FIRST® DIVE℠
presented by Qualcomm

firstinspires.org/robotics/ftc

2024-2025 *FIRST* Tech Challenge

FIRST Tech Challenge Docs

FIRST Tech Challenge Documentation

Sponsor Thank You Thank you to our generous sponsors for your continued support of the *FIRST* Tech Challenge!

Contents

Welcome to the *FIRST®* Tech Challenge Documentation! This website contains everything you need to know to create a competition robot! There is information and tutorials on how to use the *FIRST* Tech Challenge software and robot control system. There is also information for coaches and mentors.

FIRST Tech Challenge is a robotics program for middle and high school students. It's way more than building robots, see *About the FIRST Tech Challenge* and *Gracious Professionalism®* to see why.

Chapter 1

About the FIRST *Tech Challenge*

It's way more than building robots. *FIRST* Tech Challenge teams (up to 15 team members, grades 7-12) are challenged to design, build, program, and operate robots to compete in a head-to-head challenge in an alliance format.

Guided by adult coaches and mentors, students develop STEM skills and practice engineering principles, while realizing the value of hard work, innovation, and working as a team.

The robot kit is reusable from year to year and can be coded using a variety of levels of graphical and Java-based programming. Teams design and build robots, raise funds, design and market their team brand, and do community outreach to earn specific awards. Participants are encouraged to explore future education, enlistment, and other employment pathways.

Each season concludes with regional championship events and an exciting *FIRST* Championship.

About FIRST Tech Challenge (2021)

1.1 Start a *FIRST* Tech Challenge Team

FIRST Tech Challenge teams design and build a robot using a reusable kit of parts and compete within a common set of game rules to play an exciting field game and complete the specific season challenge. The robot game changes every season and is always a blast!

Student and adult team members are encouraged to bring any skills they already have, like programming, electronics, metalworking, graphic design, web creation, public speaking, videography, and many more. *FIRST* Tech Challenge welcomes every student, with or without special skills.

If you are looking to incorporate *FIRST* into your classroom or after-school programming, learn more about *FIRST* Class Pack, a flexible implementation option for up to 24 students.

Continue on to learn about the essential steps to starting a *FIRST* Tech Challenge Team!

1.2 About *FIRST* Tech Challenge Kahoot

This is a fun self-led *FIRST* Tech Challenge Kahoot to test and build your knowledge. Learn about the many facets of *FIRST* Tech Challenge and the Core Values of *FIRST* in the *FIRST* Tech Challenge Kahoot.

Chapter 2

Gracious Professionalism®

Gracious Professionalism® is part of the ethos of *FIRST*. It's a way of doing things that encourages high quality work, emphasizes the value of others, and respects individuals and the community. *Gracious Professionalism* is not clearly defined for a reason. It can and should mean different things to everyone.

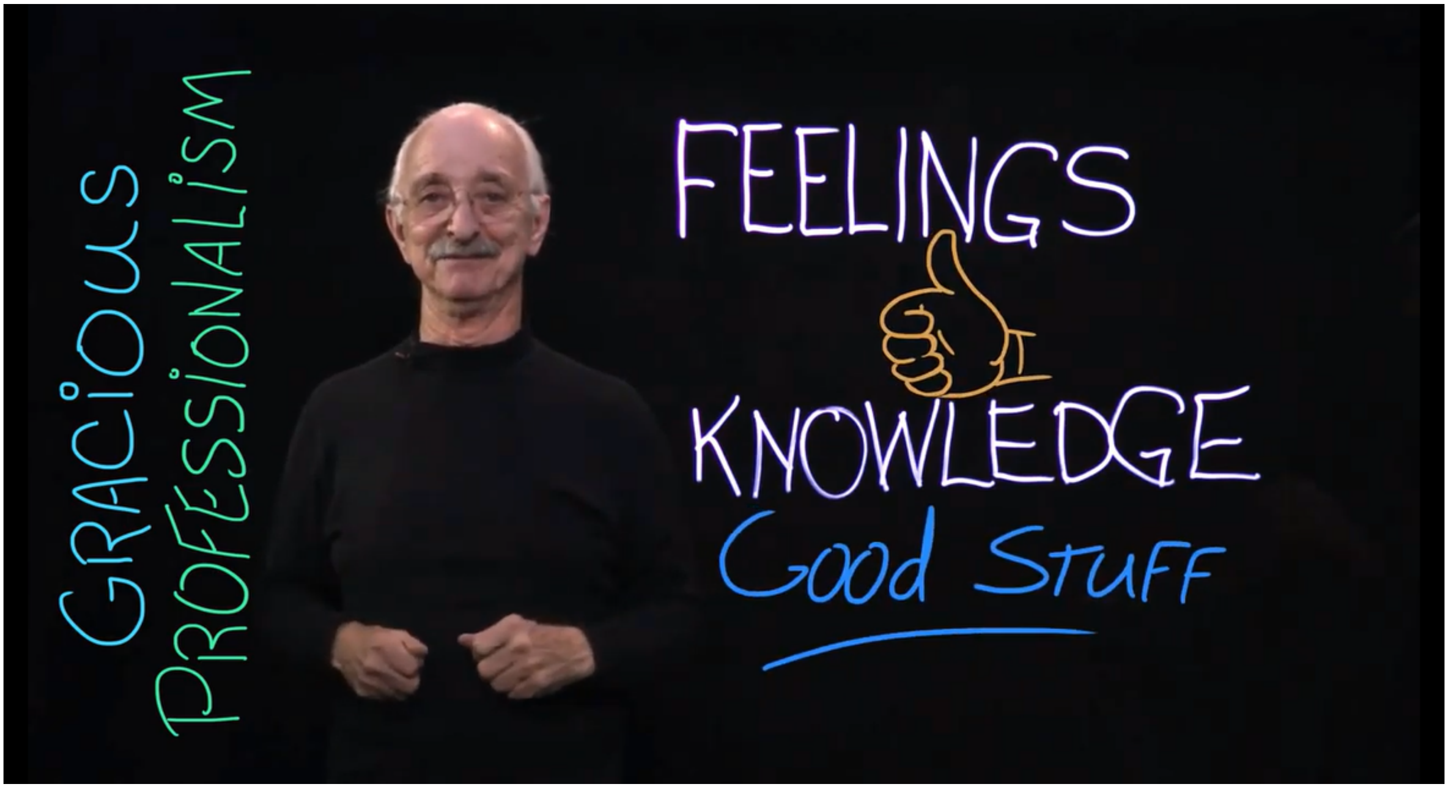
Some possible meanings of *Gracious Professionalism* include:

- gracious attitudes and behaviors are win-win,
- gracious folks respect others and let that respect show in their actions,
- professionals possess special knowledge and are trusted by society to use that knowledge responsibly, and
- gracious professionals make a valued contribution in a manner pleasing to others and to themselves.

In the context of *FIRST*, this means that all teams and participants should:

- learn to be strong competitors, but also treat one another with respect and kindness in the process, and
- avoid leaving anyone feeling as if they are excluded or unappreciated.

Knowledge, pride, and empathy should be comfortably and genuinely blended. In the end, *Gracious Professionalism* is part of pursuing a meaningful life. When professionals use knowledge in a gracious manner and individuals act with integrity and sensitivity, everyone wins and society benefits.



The FIRST spirit encourages doing high-quality, well-informed work in a manner that leaves everyone feeling valued. Gracious Professionalism seems to be a good descriptor for part of the ethos of FIRST. It is part of what makes FIRST different and wonderful.

- Dr. Woodie Flowers, (1943 – 2019), Distinguished Advisor to *FIRST*

It is a good idea to spend time going over this concept with your team and reinforcing it regularly. We recommend providing your team with real-life examples of *Gracious Professionalism* in practice, such as when a team loans valuable materials or expertise to another team that they will later face as an opponent in competition. Routinely highlight opportunities to display *Gracious Professionalism* at events and encourage team members to suggest ways in which they can demonstrate this quality themselves and through outreach activities.

2.1 In Memoriam

In October 2019, Dr. Woodie Flowers, an innovator in design and engineering education and a Distinguished Advisor to *FIRST* and supporter of our mission, passed away. As thousands of heartfelt tributes to Woodie have poured in from around the world, it is clear his legacy will live on indefinitely through the gracious nature of our community and our ongoing commitment to empowering educators and building global citizens.

Fig. 1: Dr. Woodie Flowers, (1943 - 2019)

Chapter 3

New Teams

Welcome to *FIRST* Tech Challenge! Resources have been organized by type to help your team stay organized and be successful throughout the season. Get started by exploring our robot building resources, control system and the game. You may also find the Coach's Playbook, a weekly schedule of activities, helpful to organize the whole team under Team Management. Just click on the button for the resource you want to explore!

FIRST Core Values

We express the *FIRST* philosophies of Gracious Professionalism and Coopertition through our Core Values.

Gracious Professionalism

Coopertition (external)

Core Values (external)

Programming Resources

Look for programming resources here.

Choosing a Programming Tool

Blocks Programmming Tutorial

Programming Resources

Robot Building and Control

Look for robot and control system resources here.

The FTC Control System

Robot Controller

Driver Station

Hardware Component Overview

Robot Building Resources (external)

Competition Manual

Be sure you're following all of the rules of the competition! The Competition Manual is an essential document.

Competition Manual

Playing Field Resources

Game Question and Answer System (external)

Team Management

Links to team management resources like team registration, mentor training/resources, team budget and fundraising, preparing for a competition, and more.

Team Management (external)

CAD Resources

Look for resources for Computer-Aided Design (CAD) software.

CAD Resources

Event Info

FTC events and event results.

FTC Events (external)

Awards

Know the awards criteria before the event.

FTC Awards (external PDF)

Frequently Asked Questions

Commonly asked team questions.

Frequently Asked Questions

Chapter 4

Returning Teams

Welcome back to *FIRST* Tech Challenge! Resources have been organized by type to help your team stay organized and be successful throughout the season. These resources are tailored to teams with experience in robotics looking to elevate their skills. The technical resources are a stepping stone towards industry standard certifications. Just click on the button for the resource you want to explore!

Programming Resources

Look for Java programming resources here.

OnBot Java Tutorial

Android Studio Tutorial

AprilTag Introduction

Programming Resources

CAD Resources

Look for resources for Computer-Aided Design (CAD) software.

CAD Resources

Team Management

Team management resources including marketing, community and industry outreach.

Team Management (external)

Awards

Know the awards criteria before the event.

FTC Awards (external PDF)

Chapter 5

Coach (Administrative) Resources

Welcome to the *FIRST* Tech Challenge coach's page! Resources have been organized by type to help your team stay organized and be successful throughout the season. These resources are focused on the coach or administrator needs to manage the team while promoting *FIRST*'s ethos. Just click on the button for the resource you want to explore!

FIRST Core Values

We express the *FIRST* philosophies of Gracious Professionalism and Coopertition through our Core Values.

Gracious Professionalism

Coopertition (external)

Core Values (external)

FIRST Dashboard Links

Discover registration and purchasing FAQs. Team Registration requires *FIRST* Dashboard account access on firstinspires.org.

Register Your Team (External Login)

Youth Registration (external)

Business Plan and Budget

A simple guide to managing your team's budget.

Sample Budget (External Excel Spreadsheet)

Team Management

Resources to provide your team a well-paced and successful season.

Team Management Resources (External)

Coach Guidance

Discover the best practices for new coaches.

Mentor Manual (External PDF)

Pre-Event Checklists

Simple checklists to prepare for competition.

Preparing for Competition (External)

Chapter 6

Technical Mentor Resources

Welcome technical mentors to *FIRST* Tech Challenge! Resources have been organized by type to help you stay organized and be successful with teams throughout the season. These resources are tailored to teams and mentors with experience in technical fields looking to elevate their skills. The technical resources are also a stepping stone towards industry standard certifications. Just click on the button for the resource you want to explore!

Control System Resources

Look for Control System resources here.

FTC Control System

Mechanical Resources

Mechanical engineering and robot building resources

Robot Building Resources (external)

Programming Resources

Links to programming resources.

Programming Resources

CAD Resources

Look for resources for Computer-Aided Design (CAD) software.

CAD Resources

Chapter 7

FIRST *Tech Challenge Blog*

The FIRST Community Blog hosts the blogs for all *FIRST* programs.

FIRST Tech Challenge program specific information can be found on the FIRST Tech Challenge Blog. The blog hosts articles that go beyond what can be covered in the Team Email Blasts .

Chapter 8

FIRST *Tech Challenge Tech Tips*

Started in the 2023-2024 season, Tech Tips are a weekly segment released in the *FIRST* Tech Challenge Team E-mail Blast. Sometimes the Tech Tips are included in whole in the email blast, but sometimes there is more content than is reasonable in the email blast so partial content is included in the blast with the rest of the content here. Blasts are ordered on this page chronologically, with the newest content at the top of the page.

Just click to expand the Tech Tip you'd like to read.

Week of 11/06/2024 "Android Studio 2024.2.1 LadyBug Update and the FTC SDK"

Android Studio 2024.2.1 LadyBug Update and the FTC SDK

This is an important message for teams who use Android Studio to program their robots. Teams who use Blocks or OnBot Java are not impacted.

On October 1, 2024 Android Studio released a new version of their software, 2024.2.1 codenamed "LadyBug", which brought a major user interface change as well as several other changes (bundled software and tooling changes) that affects how Android Studio builds projects. Unfortunately these tooling changes broke the native compatibility with the *FIRST* Tech Challenge Software Development Kit (SDK), most notably with the FtcRobotController project. *FIRST* Tech Challenge teams who use Android Studio with software projects version 10.1 and older are not able to use Android Studio "LadyBug" without performing additional steps to restore compatibility.

Teams do not need to update Android Studio to "LadyBug" to continue building current software, however if they do, a new version of the FtcRobotController project (10.1.1) has been released which is designed to work with Android Studio "LadyBug." Users will be required to upgrade their Android Studio software minimally to Android Studio 2024.2.1 "LadyBug" in order to use the 10.1.1 version of the SDK and newer. There are no feature updates to SDK 10.1.1, it is merely a compatibility update which updates the build tools used by the SDK - including the underlying Gradle tools and the Android Gradle plugin - and eliminates the need to perform any additional steps to use Android Studio "LadyBug" and newer. It is expected that future updates of the SDK will build upon this update, and will minimally require "LadyBug." Teams who are using older versions of Android Studio who upgrade to SDK 10.1.1 will receive notifications within Android Studio to update the version of Android Studio, which may require an internet connection to update.

Teams are encouraged to read the Managing an Android Studio Project article on ftc-docs for tips on managing their projects using GitHub and the git version control system. Teams managing software projects outside of GitHub and git may re-download the project, reapply their changes, and copy over their TeamCode folder. Teams who need technical assistance may use the ftc-community forums to receive technical help and advice.

Week of 10/10/2024 "AprilTag Localization"

AprilTag Localization

This week's Tech Tip is all about AprilTag Localization. How can your robot determine where it is on the field by looking at an AprilTag? A new set of APIs have been added to SDK 10.0 to provide just that information, and it works for any static (immobile) AprilTag on the competition field. Check out the AprilTag Localization documentation on ftc-docs!

Week of 09/09/2024 "AI Innovation Corner - Google AI Studio"

AI Innovation Corner - Google AI Studio

This week's Tech Tip of the Week launches a new initiative in *FIRST* Tech Challenge, an AI Innovation Corner. Generative AI has taken the world by storm, becoming commonplace now in everything from personal assistants, search engines, recipe curation, music innovation, and vehicle maintenance! Machine Learning AI has been a part of *FIRST* Tech Challenge in some way for the past six years, and we're now transitioning to help teams learn how to use and incorporate Generative AI in their *FIRST* Tech Challenge experience (while we're learning ourselves!).

The first step (or *FIRST* step?) to getting the most out of AI is choosing a model. What do I mean by model? Every AI is a neural network that has been trained with specific knowledge with the ability to do specific things based on that knowledge. Each version of this neural network is stored in a "model". Each different company has different models available for different purposes, though most models are variations on their flagship model (Gemini from Google, ChatGPT 4-o from OpenAI, Claude from Anthropic, and so on). Each company has different web-based and API interfaces for interacting with their models, and everyone has their favorite. In *FIRST* Tech Challenge, the standard tool we use is Google AI Studio to interact with Gemini.

Google AI Studio is free to use, but requires a Google account to access - virtually all models require a login or API token of some kind to use. Google AI Studio is our favorite for its list of examples (Prompt Gallery) and its easy to use interface to save prompt sessions and resume them later. With Google AI Studio, you also can select the specific model you want to use, and when available you can choose to use preview versions of up and coming models.

Future AI articles will be released not under the "Tech Tip of the Week" headline but under the "AI Innovation Corner" headline. Keep an eye out for future AI tips via the Team Blast and ftc-docs website!

Week of 08/19/2024 "REV Driver Hub Batteries"

REV Driver Hub Batteries

This week's Tech Tip of the Week focuses on the REV Driver Hub. Sure, we already did a pretty thorough deep dive on the REV Driver Hub in the 11/06/2023 Tech Tip "Driver Hub or Smartphone?", but we never really covered the batteries used in the Driver Hub themselves - and, of course, this topic was recently brought up in a team question. The question was, "Why aren't batteries for the REV Driver Hub interchangeable?"

Well, that wasn't the actual question, as the team didn't know the question they SHOULD have been asking, but that was the root of the issue. The team in question had purchased an extra REV Driver Hub battery, charged it, and was using it as a spare. We've also heard anecdotes from teams who attended events where FTAs would also purchase spare batteries (or pull batteries from spare Driver Hubs) and let teams with depleted batteries use their charged batteries. However, in each case the teams noted that the spare battery never lasted as long as their "regular" batteries, often significantly shorter (half or less). The issue is actually not specific to the REV Driver Hub, but in the batteries themselves.

I noticed the same thing a few years ago when I owned a smartphone that had user-replaceable batteries. My phone battery stopped holding a charge, so I bought a battery online to replace it. However, I noticed that the replacement battery had a significantly lower "lifespan", meaning it would go from full charge to near-dead in a shorter period of time versus the original battery. Over time the battery seemed to "last longer", until after about a dozen charge cycles it was very close to the original battery's performance. Did the battery get better, or did my phone adapt to the battery?

What I didn't know was that minor variances in how batteries are manufactured, especially in lower-voltage Li-Ion batteries, can affect the voltage stability of the battery as it depletes (how the voltage of a battery changes as it's used). In order to know how much battery power is left, the device needs to know the "charged" voltage, the "depleted" voltage, and generally needs to understand how the battery voltage changes from one extreme to the other. Unfortunately this isn't linear, and differences in a battery's specific internal resistance and other factors will cause each battery to have different behavior (this occurs in all batteries, but higher capacity batteries with low internal resistance tend to show this difference less). The REV Driver Hub performs a calibration phase as it charges a battery, and stores the battery charge characteristics - that helps it know how the battery should behave when it's being depleted. In this way the Driver Hub "learns" how to interpret the battery it's charging so that it can create an accurate charge profile for the battery as it's used by the device.

When a team replaces the primary battery with a spare, the Driver Hub doesn't necessarily know that this has happened, and can only apply the stored discharge characteristics for the primary battery to the new battery. Unfortunately this often leads to the device misinterpreting the battery and shutting down before the battery has fully depleted, or thinking there's more battery left when there really isn't. If the new battery is then charged, the Driver Hub will calibrate to the new battery, and changing the battery again will cause the Driver Hub to mischaracterize the original battery if replaced.

It is highly recommended that all teams use an external USB Battery Pack connected to the USB-C port on the Driver Hub to provide consistent power (use USB-A to USB-C cables only). The battery pack will sustain your Driver Hub and keep it from being additionally depleted by any high-power-drain gamepads (such as the Sony DualShock and Sony DualSense gamepads) that your team may be using.

Week of 07/29/2024 "Servo Power Injectors"

This week's Tech Tip of the Week is intended to be a short treatise on Servo Power Injectors. Servo Power Injectors have been used in FIRST Tech Challenge for several years now, but do you really understand what they are and how they work? What is a Servo Power Injector and how might servos behave differently when used with one?

A servo connection is a 3-wire combination that combines power, ground, and a signal. The actual command signal for the servo travels on the signal wire, and the power used to power the servo travels on the other two wires. A servo power injector is a device that removes the power provided by the servo controller (REV Control Hub or REV Expansion Hub for FTC) and provides a new, usually higher wattage power source. Both the REV Servo Power Module and Studica Servo Power Block replace the 5V/10W power provided by the REV Control/Expansion Hubs with a 6V power source with a higher maximum wattage.

So what does higher voltage do for a servo? Servos operate on a power range; the more power they get, the faster and stronger they can become, up to a certain limit. Servos operating at 5V get a noticeable boost in speed and output power when used at 6V. The same servos may seem superhuman at higher voltages!

The downside of servo power injectors is that teams are now responsible for managing their own power usage. On the REV Control Hub, for instance, each servo port pair is limited in how much power it can draw (at least there's a limit on how long it can draw high loads). When using a servo power injector, the pool of power for a servo is much larger and less restricted since it pulls its power directly from the robot battery - using power injectors means you could consume all of the power on the robot just from the servos alone! This will result in the robot power system browning out (resulting in loss of communications or loss of power to the control system) or even blowing the 20A battery fuse.

Using a servo power injector can also expose different behaviors in servos that were not present when using the REV Control/Expansion Hub directly. The biggest behavior is the "Lost Signal" behavior. When an OpMode ends, the REV Control/Expansion hubs stop the signal and also cut power to the servo ports - this leads to the servos "going limp" as they lose power. With a servo power injector, the servos never lose power, and so "lost signal" behaviors will often then take over which may cause the servo to move to a "default" position (which is virtually never advantageous for robots but definitely advantageous for R/C planes for example). The Axon MAX+ servo and several higher-power HiTec servos have this behavior, the Axon MAX+ behavior is at least configurable with a servo programmer.

Finally, when using a servo power injector it's of VITAL importance that you cover unused ports with tape or other debris-limiting measures to protect the ports. It's very easy to get metal swarf in open servo ports, and that metal can short out the power output pins - especially lower-cost power injectors cannot tell when they're being used, or don't have protections against short circuits, but they still have all output pins powered. This can quickly turn your servo power injector into an expensive paperweight when the power regulator overloads and burns out.

Week of 06/24/2024 "Calculating Motor and Servo Power"

In this Tech Tip of the Week we'll be exploring mechanical and electrical power, why some types of power are calculated differently, and how to use this calculated power to compare servos. This Tech Tip was written and fact-checked with the help of Google Gemini 1.5 Flash using Google AI Studio.

The fundamental concept we need to understand is power. We are generally concerned with two similar but different kinds of power, so let's look at the two most common types. In a motor, **electrical power** is the energy supplied by the electrical current flowing through the motor's windings. This electrical energy is transformed into **mechanical power**, which is the rate at which the motor performs work by rotating a shaft. Both kinds of power are measuring different aspects of the motor; electrical power deals with the movement of electrical charges, and mechanical power deals with the movement of objects due to forces. Both of these measurements are expressed in the same unit, Watts (W), because power, in general, is defined as the rate of energy transfer or work done. No matter the form of energy (electrical, mechanical, thermal, etc.) the fundamental concept of power remains the same. Even though these two power measurements carry the same unit, they are calculated differently and **cannot be used interchangeably (or together!)**.

Motors and servos are constructed similarly - both are electromechanical devices that convert electrical energy into mechanical energy - but there are big differences in how they're used. Motors are often used in applications requiring continuous power, such as pumps, fans, and conveyor systems. Motors are typically rated for **continuous power output**, meaning they can sustain that power level indefinitely without overheating. Servos are commonly used in robotics and precision positioning systems, where controlled movement and precise positioning are essential. Servos are designed for intermittent operation - typically cycling through on/off periods to control movement - and are often rated for their **stall torque** and **no-load speed** reflecting their ability to hold a position against a force and how fast they move when unloaded. While electrical power is calculated generally the same for both types of devices, these design and use differences have an impact on how mechanical power is determined.

Both motors and servos calculate **electrical power** the same, using the standard electrical power formula:

- *Electrical Power(W) = volts(V) x amps(A)*

For example, a typical REV Smart Servo is supplied with 6V when used with a REV Servo Power Module (SPM) or 5V when used with a Control or Expansion Hub. Per the servo's specs, at 6V the servo will pull at most 2A at stall (when the servo cannot physically move to the position it's being commanded to). This means the maximum electrical power the servo will consume is 12Watts of power when plugged into the REV SPM and being commanded to a position it cannot reach. The REV SPM supplies 90W of maximum electrical power, so the maximum number of fully-stalled REV Smart Servos the SPM can supply full power to is 7 (90W divided by 12W, ignoring the remainder).

Motors and servos also generally calculate mechanical power similarly.

- *Mechanical Power(W) = torque (N-m) x angular speed (rad/s)*

Mechanical Power for a DC motor generally follows a very specific curve, based on its efficiency, stall current, stall torque, speed, and a bunch of other factors. The general performance curve of a DC motor can be seen in Figure 1.

From this we can see that the Peak Power is found at the intersection of 1/2 Stall Torque and 1/2 Speed. Even though a servo is used different than a generic motor, this approximation is still good for calculating the maximum mechanical power of a servo. Simplified, we can use this formula:

- *Servo Max Mechanical Power(W) = 0.25 x stall torque(N-m) x no-load speed(rad/s)*

Using this approximation the REV Smart Servo, when being provided 6V, produces a maximum Stall Torque of 13.5kg-cm (1.33N-m) and a time of 0.14s per 60 degrees of travel (7.48rad/s) yielding an approximate max servo mechanical power of 2.48W.

Tip: It's important to point out that a high speed motor or servo that is loaded past its maximum power point will actually do worse than a slower motor or servo with the same load. It's all about getting the maximum mechanical power by operating the motor at the max power point.

One of the most difficult parts of calculating Servo Mechanical Power is working with unit conversions, especially since servo manufacturers use lots of different units. In order to calculate servo mechanical power correctly the speed unit MUST be converted to radians-per-second and the max stall torque unit MUST be converted to Newton-meters. Below is a handy

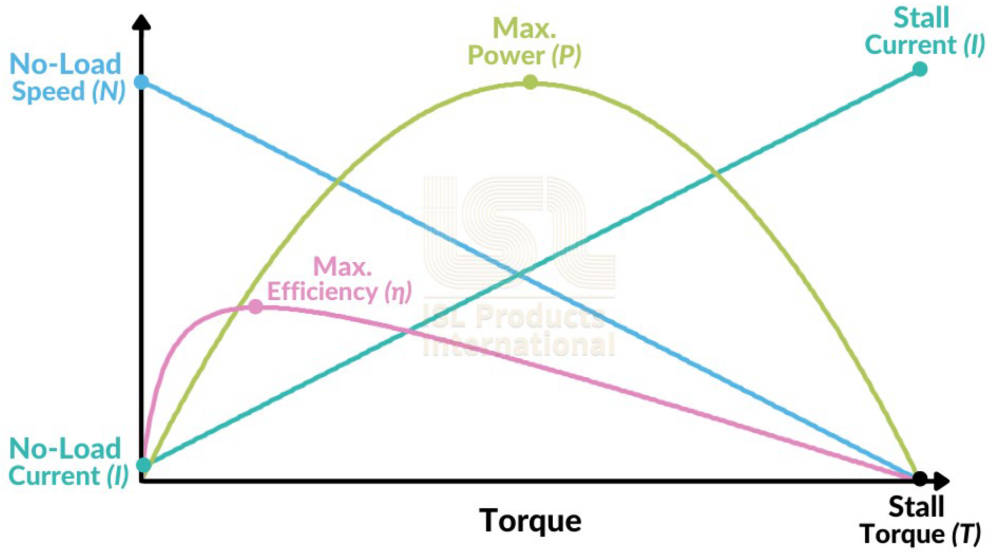


Fig. 1: Figure 1: General DC Motor Performance Curve

calculator that you can use to automatically perform the necessary conversions and calculate Servo Mechanical Power (*Thank you to Orion DeYoe for providing this tool*).

Tip:

- For Speed, use the radio button to choose the unit type that the manufacturer has provided - for most servos this will be listed in a period of time per 60 degrees (such as with the REV Smart Servo example) or perhaps the manufacturer may provide an angular velocity, such as rotations-per-minute (RPM). Enter the no-load speed value and unit as the manufacturer has provided.
- For stall torque, provide the value and select the unit as specified by the manufacturer. If the manufacturer merely provides kg, assume kg*cm.

The calculator automatically recalculates on any changes, there is no button to press in order to trigger a calculation.

Here is a handy table of some common servo mechanical power values:

Table 1: Common Servo Mechanical Power Values (@6V)

Description	Speed	Torque	Stall Current	Max Power	Cost ($USD)
Tetrix MAX Standard (HiTec HS-485HB)	0.18 s/60°	6 kg-cm	1.2 A	0.86 W	$29.50
REV Smart Servo	0.14 s/60°	13.5 kg-cm	2.0 A	2.48 W	$30.00
goBILDA 2000 Series Speed Servo	0.09 s/60°	9.3 kg-cm	2.5 A	2.65 W	$33.99
Axon Robotics Micro+	0.075 s/60°	7.8 kg-cm	2.2 A	2.67 W	$63.79
goBILDA 2000 Series Torque Servo	0.20 s/60°	300 oz-in	2.5 A	2.77 W	$33.99
Studica Multi-Mode Smart Servo 200	0.046 s/60°	5 kg-cm	2.7 A	2.79 W	$24.99
goBILDA 2000 Series Super Speed Servo	0.043 s/60°	4.7 kg-cm	2.5 A	2.81 W	$33.99
AndyMark am-4954 High Torque Servos	0.20 s/60°	22 kg-cm	1.7 A	2.82 W	$34.00
Studica Multi-Mode Smart Servo	62 RPM	20 kg-cm	1.8 A	3.18 W	$23.99
AndyMark am-4955 High Speed Servos	0.05 s/60°	7 kg-cm	2.7 A	3.59 W	$30.00
FeeTech FT5335M-FB	0.20 s/60°	35 kg-cm	4.0 A	4.49 W	$52.95
HiTec HS-805BB	0.14 s/60°	24.7 kg-cm	6.0 A	4.53 W	$49.99
HiTec HSR-M9382TH	0.17 s/60°	34 kg-cm	2.7 A	5.13 W	$199.99
Power HD GTS3	0.083 s/60°	20 kg-cm	4.0 A	6.19 W	$120.00
Axon Robotics MINI+	0.09 s/60°	25 kg-cm	3.8 A	7.13 W	$79.99
Axon Robotics MAX+	0.115 s/60°	34 kg-cm	4.0 A	7.59 W	$79.99

Week of 06/10/2024 “Updating the SDK Manifest”

This week’s Tech Tip of the Week comes to us from an amalgamation of emailed questions asking about allowed ways to update an FtcRobotController SDK project. An approximate summary of the emailed questions along this topic is as follows:

- *“Is merely editing the Android Manifest file in the TeamCode directory of the FtcRobotController SDK project an acceptable way of easily updating the SDK? And would this violate RS08 in Game Manual Part 1?”*

Manually editing the Android Manifest file in the TeamCode Directory of the FtcRobotController SDK software is not a violation of RS08, merely because RS08(b) only protects the binary .AAR files. The manifest file is not part of the .AAR binary, and thus it’s not protected.

Even though it’s not forbidden, that doesn’t mean you should do it – like putting pineapple on pizza (sorry, the door was open, I couldn’t stop myself). Seriously, though, 4 times out of 5 you can likely get away with updating the SDK through editing the Android Manifest to point to the latest version of the SDK libraries. However, that assumes that all the Tech Team does is update the SDK libraries, which is never ever the case. In addition to also updating programming samples, often enough the Tech Team must also update tooling, dependencies, and other build items in addition to the SDK libraries, and simply updating the Android Manifest is going to get you into real trouble (things will appear to work, until they don’t, and you won’t know why). As a corollary, you can choose to simply only put gas in your car and ignore all the other fluids, but eventually you’re going to wish you hadn’t.

The proper way of updating your SDK is to use Git/GitHub to update your robot source each time the SDK software updates. The Tech Team always updates the FtcRobotController in-place (meaning the same repo is always updated each version), so if you’re using Git you can easily pull the changes made upstream and accept the changes within your code. You should never be manually updating files, like the Android Manifest file, because Git can tell you all of the files you need to update and can do that for you. If you use Git or GitHub, we highly recommend reading our guide on ftc-docs for *managing your Android Studio project repositories*.

For example, check out these changelists. The FtcRobotController v9.0 commit/change is everything that needs to be changed to upgrade from version 8.2 to 9.0 – there are 75 changed files there, which include samples, a core interface module change, gradle dependencies, and in that changelist the Tech Team also rearchitected the asset structure. However, the FtcRobotController v9.0.1 and FtcRobotController v9.1 pull requests only changed a handful of files (mostly samples), and the core changes are in the AndroidManifest.xml and build.dependencies.gradle files. In general our major version releases (where we increase the first number in the version string) are the big ones, and then the dot-releases are almost always fairly small targeted releases. The Tech Team tries very hard not to make big-scale changes to build systems or major dependencies during the season.

In summary, teams should never simply change the Android Manifest, they should be updating the software appropriately – as Voltaire warned, with great "Android Studio" power comes great "GitHub" responsibility.

Week of 05/20/2024 "Wi-Fi Bands, Part 3"

Welcome back to the Tech Tip of the Week, this is Part 3 of a 3-part series talking about Wi-Fi bands and why you might be shooting yourself in the foot by not selecting (and designing your robots for) the right Wi-Fi band. In Part 1 we discussed the physical characteristics and properties of frequencies in each of the 2.4GHz and 5GHz bands. In part 2 we talked about the history of the bands, described sources of interference (e.g. other devices!) on each band, and how Wi-Fi improvements have made 5GHz more efficient to use.

Robot design - and more aptly "Control Hub placement" - is THE critical factor in influencing the Wi-Fi frequency/band you should be using. Remember Wi-Fi is a line-of-sight technology, that means Wi-Fi does best when there's a straight unobstructed path from the antenna on the Control Hub to the antenna on the Driver Hub. Where is the antenna in a Control Hub? It's right under the plastic on the "face" of the hub on the logo side. If the Control Hub can be mounted so that its antenna is generally not covered/surrounded/blocked by metal, 5GHz should be your target band. However, if your Control Hub is buried deep inside the robot and surrounded by metal, the 2.4GHz band may be your only option (remember, the lower frequencies of 2.4GHz might be able to "bend around" metal obstacles slightly better). Unfortunately exposing the "back side" of the Control Hub instead of the "front side" of the hub is not going to yield similar results, as there is a PCB with metal traces between the antenna and the "back side" of the Hub that will block/reflect/absorb signals.

Does that mean your Control Hub needs to be mounted unprotected on the outside of the robot in order to get good signal reception? Not necessarily, fortunately not all materials are the same. Plastics are generally the most "invisible" to Wi-Fi frequencies, or at least their absorption/blocking/reflection (also known as attenuation) is generally minimal enough to not sufficiently matter. Wood, especially thin birch commonly used in many robot designs, is slightly more attenuating but definitely still a great option. Metals, however, will greatly attenuate Wi-Fi frequencies and are the worst materials for Wi-Fi transmission. Yes, I'm looking at YOU teams who use hook-and-loop to mount your robot battery to the top of the Control Hub - stop doing that! And for those looking for inspiration in this upcoming season, water is also an incredibly poor medium for transmission of Wi-Fi frequencies.

But how do you know for sure how well your robot's Wi-Fi is performing? You can monitor the Wi-Fi signal's strength through the Driver Station App. Check out the 2024/02/15 Team Blast Tech Tip for info on how to view and understand Wi-Fi Signal Strength. If your signal is strong when using 5GHz at maximum field range (from the Driver Hub) and in all robot orientations, you should be good to go on 5GHz! Feel free to compare the performance on 5GHz and 2.4GHz, and if they're comparable you should stick with 5GHz for better interference reduction.

In summary, the vast majority of robots should be using 5GHz as this is the optimal channel in terms of interference reduction, device crowding, and channel utilization by the Wi-Fi standards. Robot design - specifically Control Hub placement - might necessitate the use of 2.4GHz if the line-of-sight path to the Control Hub antenna in the robot is too greatly obstructed by metal, especially motors. By monitoring the robot's Wi-Fi signal strength, you can determine which frequency band yields the best Wi-Fi signal performance for your robot.

Week of 05/06/2024 "Wi-Fi Bands, Part 2"

Welcome back to the Tech Tip of the Week, this is Part 2 of a 3-part series talking about Wi-Fi bands and why you might be shooting yourself in the foot by not selecting (and designing your robots for) the right Wi-Fi band. In Part 1 we discussed the physical characteristics and properties of frequencies in each of the 2.4GHz and 5GHz bands. In this part we'll talk about sources of interference.

You might have realized this, but wireless devices are all the rage. The FCC (in the USA) doesn't just let any device broadcast on any frequency they want. Instead, there are licensed and unlicensed radio frequency bands. Some frequencies are uniquely licensed to private operators, for example radio stations pay a lot of money to the FCC for the exclusive rights to broadcast on specific frequencies. HAM radio operators undergo special training to be allowed to broadcast on a range of licensed frequencies (some reserved only for HAM radio, some not). The FCC also sets aside frequencies that are unlicensed, meaning the operators themselves (like you, your neighbor, or the kid down the street) don't need training or licensing to operate devices that broadcast on those frequencies. The devices themselves must adhere to specific regulations, but those requirements are generally easy to meet.

Wi-Fi uses portions of the radio frequency spectrum designated as unlicensed - remember that these frequencies are available to the general public to use - so anyone can broadcast signals over it. And boy howdy do they. The 2.4GHz frequency band was opened to the public in 1985, and devices began using that frequency for use. Wi-Fi emerged in the late 1990's. The 2.4GHz frequency band became extremely crowded, and by devices using different protocols - think about trying to have a conversation with a friend in a crowded room, but some people are talking "normally", some are using air horns, and others are mimicking nails on a chalkboard. The resource was very narrow, but at least interference was just a matter of distance - though not everyone lives in the deserts of Arizona where they can carry out their conversations in relative peace.

By the turn of the 20th century, the 5GHz space was opened up for unlicensed use. This required different hardware, as the 2.4GHz devices couldn't simply just start using 5GHz. The 5GHz band was much larger, and it took longer for it to become crowded as more devices came onto the market that could use it. 5GHz already had a bunch of legacy systems that used portions of it, and so the FCC grandfathered those systems and made special regulations for using those frequencies (most manufacturers designed their devices to only use the portions of the 5GHz band with the least rules and regulations). Some uses of 2.4GHz could not move to 5GHz because of the frequency wave propagation behaviors (that we talked about previously, e.g. reflections and wave bending), but many systems like Wi-Fi found the greatest use in 5GHz. The number of channels and the frequency space was much larger in 5GHz, and 5GHz Wi-Fi technologies learned to use the 5GHz space more efficiently and robustly.

When you consider which frequency you should use, you have to consider many factors. How obstructed is the path from the radio to the receiver? How crowded might the frequency space be that you're trying to use? Has the event organizer worked with the venue to clear specific channels for robots to use? What advanced technologies might the device you're using be capable of utilizing on specific frequency bands?

In Part 3 of this series we'll talk about how robot design can influence the Wi-Fi frequency you should be using, how to design for the best possible outcome, and how to characterize your optimal band.

Week of 04/29/2024 "Wi-Fi Bands, Part 1"

Welcome to the Tech Tip of the Week, where this week hopefully "Bandwidth of Robots" will be your new favorite way to refer to groups of wireless robots. Today we'll be starting a three-part series talking about Wi-Fi bands and why you might be shooting yourself in the foot by not selecting (and designing your robots for) the right Wi-Fi band. And at the end of the day how do you truly know which band you should be using?

If you're anything like the average team, Wi-Fi bands are something nebulous that you don't really understand or even give a second thought to. At least, until "bad things" start happening and you're grasping at straws trying to resolve them. So let's start this discussion by talking about radio frequency bands and then the two Wi-Fi bands we have access to, 2.4GHz and 5GHz.

What are the important properties of Wi-Fi frequencies we should know? To explain Wi-Fi frequencies, let's look at something most of us might already be more familiar with - AM and FM radio frequency bands (which share similar behaviors, ignoring modulation differences).

AM radio stations are assigned carrier radio frequencies between 540kHz-1600kHz. For example WGHM 900 AM out of Nashua, NH, is licensed to broadcast at 900kHz. AM radio station signals travel very far very easily mostly because the frequencies in AM radio have very large wavelengths - 900kHz, for example, has a full wavelength of 333m (just over one fifth of a mile) - and because of this they can bend around obstacles very easily (buildings, mountains, curvature of the earth, etc). However, long wavelength AM radio is more susceptible to interference and static than shorter wavelength transmissions, like FM.

FM radio stations are assigned frequencies between 88.1MHz-108.1MHz. For example, WEVS 88.3 FM also in Nashua, NH broadcasts at 88.3MHz. FM radio frequencies are higher frequency, and have a shorter wavelength - 88.3MHz is about 3.4m (about 11 feet) in wavelength - and cannot bend around obstacles as easily. Shorter wavelength frequencies also tend to be absorbed/reflected (comparatively) much easier by obstacles as well.

Hence when driving through the mountains and forests of NH I am more apt to be able to cleanly listen to the AM station uninterrupted but not the FM station, even though they're broadcasting at roughly the same power and from very similar locations.

Frequency bands used for Wi-Fi share very similar characteristics, but because the frequencies for Wi-Fi are much higher some characteristics are more exaggerated. As an analogy, for the purposes of this discussion, we can say that 2.4GHz is to 5GHz as AM is to FM. 2.4GHz frequencies have a longer wavelength (starting at ~0.125m or ~5 inches) than 5GHz frequencies (starting at ~0.05m or ~2 inches), and because of that 2.4GHz radio waves can bend around objects better than 5GHz ones but are much more susceptible to interference than 5GHz. Similarly 5GHz frequencies will also tend to be reflected/absorbed much easier by solid objects, and so 5GHz tends to perform better with an unobstructed line of sight between antennas.

In Part 2 of this series we'll talk more about the challenges Wi-Fi faces because unlike AM and FM radio, Wi-Fi doesn't have dedicated frequency space. This can cause legitimate issues due to the number of existing devices and services that already use frequencies that Wi-Fi has to share.

In Part 3 of this series we'll talk about how robot design can influence the Wi-Fi frequency you should be using, how to design for the best possible outcome, and how to characterize your optimal band.

Week of 04/08/2024 "What makes Battery Voltage Sag? Part 3"

This Tech Tip of the Week is Part 3 in a 3-part series surrounding a question that we get asked at events all the time - "What makes battery voltage sag?". As a battery is heavily used, teams will notice that the voltage of the battery temporarily decreases from its starting voltage during periods of heavy use, and then generally raises back up once the heavy use has subsided. So what causes this?

There are LOTS of reasons why battery voltage will sag during use. In Part 1 we talked about battery chemistry to give an idea how a battery works, and we talked about how motor torque is inversely proportional to the power consumption (given a constant load). Part 2 covered cell health and battery temperature, both of which can affect a battery's performance and longevity. This week, we'll cover another major factor which is Internal Resistance (IR).

Understanding IR requires talking about the discharge rate of a battery. The discharge rate is a measure of how quickly the battery can deliver its stored energy. Most NiMH batteries used in FIRST Tech Challenge are rated at a nominal 12V and a maximum discharge rate of 30A, though that rate is limited by the 20A fuse. A battery's IR refers to any opposition to that flow of electric current within the battery itself. Resistance can come from a number of sources, such as resistance within the battery's chemistry (such as a breakdown of the conductive electrolyte within the battery), changes to the resistance of the electrodes (such as a buildup of crystals around the electrodes), resistance added due to connectors and wiring, and others. Rising IR affects the battery performance primarily in decreasing the Voltage and Current that the battery can provide, and causes the battery to generate excess heat when used. The starting IR of a battery can vary among different manufacturing processes and batches, so much that batteries should have their IR measured (using a CTR Battery Beak, West Mountain Radio CBA, or similarly capable battery tester - **YOU CANNOT MEASURE INTERNAL RESISTANCE DIRECTLY WITH A MULTIMETER, ATTEMPTING TO DO SO WILL BLOW YOUR FUSE AND MAY DAMAGE THE MULTIMETER!**) at "birth" (when "new" at time of purchase) and the IR then should be tracked over time. Once the battery's IR increases by 50% from when it was "born", the battery is universally considered ready for replacement.

> **Danger:** You cannot measure the internal resistance of a battery directly with a multimeter. Please do not even try. Doing so will certainly blow your multimeter's fuse, and may even damage the multimeter. Please do not attempt. Internal resistance can only be measured indirectly using a load-measuring device like a CTR Battery Beak.

What can teams do to slow the increase in a battery's IR? Naturally the battery's IR will change as the battery ages, increasing due to chemical changes and wear and tear. The temperature of the battery can also have a negative effect on IR, higher temperatures cause higher resistance (so keep your batteries cool!). It's also important to note that the state of charge of a battery can change the IR, battery IR should always be measured fully charged. But the most important ways to keep your battery healthy are to avoid deep discharges (avoid letting your batteries drain below 10V steady-state, definitely never below 9V!), use a high-quality charger that prevents batteries from overcharging, follow the battery manufacturer's recommended charging procedures, and use low-resistance connections (thick wires and clean connectors!).

Finally, the IR of NiMH batteries can also sometimes be decreased through a process known as "battery conditioning" (also referred to as "charge cycling"). If IR within a battery is raised due to crystal formations inside the battery, this process of conditioning can help break down those crystal formations and improve Voltage and the flow of current in a battery. Some chargers have automatic conditioning modes, but always refer to your manufacturer's recommended procedure for charge cycling your NiMH batteries.

Week of 04/01/2024 "What makes Battery Voltage Sag? Part 2"

This Tech Tip of the Week is Part 2 in a 3-part series surrounding a question that we get asked at events all the time - "What makes battery voltage sag?". As a battery is heavily used, teams will notice that the voltage of the battery temporarily decreases from its starting voltage during periods of heavy use, and then generally raises back up once the heavy use has subsided. So what causes this?

There are LOTS of reasons why battery voltage will sag during use. In last week's Tech Tip we talked about battery chemistry to give an idea how a battery works, and we talked about how motor torque is inversely proportional to the power consumption (given a constant load). In this week's Tech Tip we'll cover two more common reasons - cell health and battery temperature. In subsequent Tech Tips we'll cover other reasons, such as the internal resistance of the battery.

Battery cell health is an important factor in the overall health of a battery. An NiMH battery used in FIRST Tech Challenge is a multi-cell battery, meaning it's composed of individual smaller batteries connected together. Each cell contributes to the overall power output of the battery. As a battery ages, individual cells in the battery may age at different rates - this aging can lead to degradation of cell material, electrolyte breakdown, and creation of dendrites that can eventually puncture the cell wall from inside the cell among others. Most often this cell breakdown is accelerated due to improper storage, overcharging, deep discharging, excessive temperatures, or physical damage (especially due to dropping). When a cell fails, it can lead to a reduced capacity of the battery pack, and the battery will not last as long on a single charge nor will it be able to provide the peak power output that it previously could. Failed cells can cause other cells to fail prematurely, primarily due to overcharging and imbalanced voltage due to the fact that NiMH batteries and chargers for NiMH batteries do not contain a load-balancing management system for individual cells. In some cases, failed cells can cause short circuits, overheating, and increased risk of fire/explosion! If you're suspicious of a battery, get it tested before using it again.

Battery temperature is also an important consideration. When a battery is being charged, it will likely become warm and even slightly hot to the touch - this is expected and natural due to the process of recharging a battery. NiMH batteries deliver their best performance at moderate temperatures. When a battery is hot from charging, its internal resistance increases (we'll cover internal resistance in a future segment) which can lead to reduced power output. Allowing the battery to cool down before use helps to ensure optimal performance. This process of allowing the battery to cool down before use can also prolong the life of the battery. This advice should also be tempered with the knowledge that most modern NiMH batteries are generally designed to handle some degree of heat; if you need to use the battery immediately after charging, it's usually safe to do so as long as the battery is not excessively hot to the touch. However, understand that it may not provide the maximum level of power output as it would have if it had cooled first.

Week of 03/25/2024 "What makes Battery Voltage Sag? Part 1"

This Tech Tip of the Week is a short one, Part 1 in a 3-part series surrounding a question that we get asked at events all the time - "What makes battery voltage sag?". As a battery is heavily used, teams will notice that the voltage of the battery temporarily decreases from its starting voltage during periods of heavy use, and then generally raises back up once the heavy use has subsided. So what causes this?

There are LOTS of reasons why battery voltage will sag during use. In this week's Tech Tip we'll cover the most common two reasons - battery chemistry and heavy use. In subsequent Tech Tips, we'll cover other reasons, such as battery cell health, battery temperature, internal resistance, and other factors to be aware of!

The first thing to remember is that a battery is a chemical reaction factory, and does not exactly work the same as the typical "gas tank" analogy makes it seem. The chemical reactions at the electrodes create a potential difference (voltage) between them. This voltage drives the flow of electrons generated by hydrogen and hydroxide ion creation and transfer. In NiMH batteries this reaction is reversible but it takes time and energy. What's important to understand is that the chemical reaction can happen only at a specific rate (the rate is based on a number of factors which we'll discuss later); if the demand exceeds the rate of reaction for the battery, the voltage and current will drop until the reactions can replenish the battery output (this temporary drop is known as "sag"). As the materials at the electrodes are gradually consumed, the overall battery charge will deplete and can no longer sustain the flow of electrons, and the battery will need to be recharged or replaced.

So what is the biggest reason why batteries will sag? On a FIRST Tech Challenge robot, this reason is actuator (motor and servo) current draw. Motors and Servos can pull a considerable amount of current when they're being used, especially when they're being used in low-torque configurations. Motors that are geared closer to 1:1 gear ratio can spin faster - they can propel your robot's drivetrain across the field much faster - but have less torque because of the lower gear ratio. Motor configurations that have less torque consume significantly more current to operate (when driving the same load) than motor configurations with more torque. Systems being driven by actuators that have more friction or less torque will cause the motors to consume larger amounts of current, and this can cause even healthy batteries to have their voltages "sag" during periods of high use. Teams must consider their power consumption very carefully when optimizing their battery and motor utilization during a match, even though that's often an afterthought for most teams.

Week of 03/18/2024 "Battery Fuses"

Welcome to the Tech Tip of the Week, where hopefully after reading you don't blow a fuse. Yup, you guessed it, we're talking today about fuses - more specifically, we're talking about the fuses on your Main Robot Battery.

Every legal Main Robot Battery in FIRST Tech Challenge is required to have an in-line replaceable fuse on the battery, you'll find the fuse housing on the red (positive) cable on your battery between the battery and the connector (the top lifts off, exposing the fuse). This fuse helps protect your battery and your electronics from prolonged or excessive over-current. The fuse used with all legal batteries is a 20A Automotive-Mini (ATM) blade-style fuse, and can be found in virtually every auto parts store. It has a yellow-colored housing which easily identifies it as a 20A fuse. If you find that your battery's voltage suddenly drops to zero (when tested using a battery tester or multimeter) it's probably because you've blown your battery's in-line fuse.

A fuse is a short span of specially-designed electrical wire intended to carry electrical loads up to a very specific amount of current. When the current loads exceed the rating, the wire within the fuse begins heating up - the more the load exceeds the rating, the hotter the wire will get. Eventually the wire will heat up so much it self-destructs and melts or burns up, breaking the circuit. This fuse-melting condition is often called "Blowing a Fuse"; the fuse is thus destroyed and is no longer usable, but it protected the electronics in the circuit as its last selfless act.

How does a fuse battery get blown? These are two of the most common reasons why a fuse can be blown:

Overcurrent Conditions - The Robot has components (generally actuators, like servos and motors) that can pull a combined current that is more than the robot's electrical circuit can safely carry. The main electrical power wires on a robot are required to be a minimum 18AWG, which can easily continuously carry up to 16A of current. When components pull a combined current far exceeding this limit, generating unsafe heat in excess of what the wires can tolerate (risking melting the wire insulation which could lead to short circuits and fire), the fuse blows to protect the circuit. The wire size and fuse limit has been carefully selected for the safety of the robot's electrical system. Short Circuits - Usually this happens if unshielded wires of opposite polarity touch each other in the robot's electrical system, like when performing electrical maintenance on switches or wires (ALWAYS unplug the battery before performing any maintenance on a robot!). Other causes can be failed electronics and damaged components. This causes an extremely high current load to travel through the battery, near-instantly causing the fuse to blow. When replacing the connector on a battery, ALWAYS remove the fuse prior to performing any work - this protects the person doing the maintenance AND protects the fuse!

Always make sure your main battery fuse is replaced with the proper fuse (20A for FIRST Tech Challenge) and make sure you're always following all safety guidelines when working with your robot's electrical system!

Week of 03/11/2024 "Signal Filtering with Ferrite Cores"

For those about to use sensors, we salute you - with our Tech Tip of the Week! This week's Tech Tip focuses on signal noise and how to eliminate it with ferrite cores.

When deciding to use a sensor on a robot, we're normally worried about how accurate the sensor's detection is, how much the sensor costs, or how the sensor's protocol will interface with the control system. It isn't until the device is being mounted to the robot before we consider how outside electrical noise already present on the robot might significantly impact the performance of the sensor. This electrical noise almost exclusively comes from the electric motors and other sources of electric fields on a robot, such as power wires, power supplies, some sensors (especially ultrasonic sensors and cameras), radio frequency generators (like the Wi-Fi on the robot), and other places. This electrical noise can generate unwanted currents through electromagnetic induction in nearby wires, especially sensor wires, and these unwanted currents can wreak havoc (create "noise") within the signals from your sensors. The amount of current induced in the wire depends on several factors including the strength of the magnetic field, the rate of change of the field, and the orientation of the wire.

Some buses and wiring are more sensitive to electrical noise than others. On a FIRST Tech Challenge robot, long signal-carrying wires (such as Servo wires or I2C sensor wires) are most susceptible to induced noise. So how can we eliminate this noise? The easiest way to remove noise is through the use of a Ferrite Core. Ferrite Cores, also known as Ferrite Beads, are made of a ceramic material called ferrite that has incredibly useful magnetic properties. When a Ferrite Core is clipped around a signal-carrying wire, the induced "noisy" alternating currents in the wire generate electrical fields in the ferrite that act to oppose those currents - this has the effect of canceling out or removing the high-frequency noise. It's not typically required to "loop" the cable around the ferrite core, but doing so could increase the efficiency of the noise filtering in cases where excessive noise is being generated. You can find ferrite cores already installed in cables meant for high-noise environments or highly sensitive devices such as USB webcam cables and monitor cables. It's best to place ferrite cores on the wire closest to the connector leading into the Control/Expansion Hub port.

Week of 03/04/2024 "Motor Modes"

This week's Tip of the Week is the first in a series for all you who love diving deep into the FIRST Tech Challenge SDK and exploring interesting lesser-known behaviors of well-known interfaces. Today we're talking about motor modes. The REV Robotics documentation for encoder feedback has a really good description of the four primary run modes, namely:

- DcMotor.RunMode.STOP_AND_RESET_ENCODER mode
- DcMotor.RunMode.RUN_WITHOUT_ENCODER mode
- DcMotor.RunMode.RUN_USING_ENCODER mode
- DcMotor.RunMode.RUN_TO_POSITION mode

The first two modes do exactly as their names suggest, and generally no more. STOP_AND_RESET_ENCODER stops the motors and resets the encoder count to zero. RUN_WITHOUT_ENCODER more or less blindly controls the motor power using a calculated percentage of the available battery power through the motor's .setPower() method. There's really no more to see here.

The last two modes are a bit more interesting. These two modes use a feature of the Control/Expansion hub firmware to externally (from robot code) control the motors. Using this feature you can do a lot more with the motors such as set the maximum velocity of the motor (nominally in encoder-ticks-per-second) using the .setVelocity() method, and *change the actual PIDF algorithm* being used by the motor mode (using the .setPIDFCoefficients() methods). Because these two motor modes rely on knowing specific motor characteristics, it's VERY important to set the correct motor type for the motor in the Robot Configuration!

Finally, one final note about RUN_TO_POSITION. When setting a Power or a Velocity for the motor in RUN_TO_POSITION mode, the value is intended to be unsigned. When using RUN_WITHOUT_ENCODER and RUN_USING_ENCODER the sign of the value of the Power or Velocity denotes direction; positive values mean run the motor "forwards" and negative values mean run the motor "backwards." However, with RUN_TO_POSITION, the current encoder value and target encoder position are already known - and thanks to the motor setting in the Robot Configuration it knows everything about the motor - therefore the controller already knows which direction to run the motor and does not need a signed value indicating direction.

Week of 02/26/2024 "Robot Controller Source Code"

Have you ever been programming your robot (especially in Blocks and OnBot Java) using FTC SDK APIs and wished you could see the source code under the hood that executes the commands you're calling? Welcome to the Tech Tip of the Week, where we're going to explore the Extracted-RC GitHub repository. Note that Android Studio users can already view source code within Android Studio!

Several years ago, FIRST Tech Challenge gave permission for the OpenFTC project to extract AAR's from our SDK releases and publicly post an extracted version of the Robot Controller source code. The Extracted-RC repository has branches that contain source code for each release of the SDK, as far back as SDK 5.2 through SDK 9.0.1. You can look up how setPower() works on a Continuous Rotation Servo, how REV Core Hex motors are defined, how Blocks OpModes are started, and even see the built-in driver for the HuskyLens vision camera.

The Extracted-RC repository will not accept Pull Requests (PR's) since the repository has no actual development purpose - it is only to allow interested folks the ability to read the source code and see how things are implemented. Only FIRST staff and Tech Team members have access to the development source. Are you interested in joining the FIRST Tech Challenge Tech Team? Let us know by filling out this survey!

Week of 02/19/2024 "Robot Wi-Fi Link Speed"

In last week's Tech Tip of the Week we talked about Wi-Fi Signal Strength. This week's Tech Tip rounds out the Wi-Fi reporting features and introduces Link Speed and the Signal Bar Graph, both found on the *FTC Driver Station App*.

Link Speed is the speed (in Mbps) at which a Wi-Fi connection can communicate, and it generally ranges from a snail-like 1Mbps through about 100Mbps, which is the maximum practical rate for an 802.11ac/b/g/n/w Wi-Fi network (when using a Control Hub and Driver Hub). It's important to understand the difference between Signal Strength and Link Speed. Signal Strength is often used to describe how "loud" a connection is, and Link Speed is used to describe how "fast" a connection can communicate. Link Speed can also be a secondary indicator of how much "noise" or "interference" a communication channel has; the "louder" the signal and "clearer" the communication channel, the "faster" the devices can generally communicate. Wi-Fi link speeds are automatically renegotiated periodically and they're most often affected by noise, channel congestion (too much happening at once), and distance.

A Wi-Fi channel is like a room where only one person/device is ever allowed to talk at a time. If each person/device can talk in short, fast bursts (fast link speed) then everyone has an opportunity to speak within a short duration of when they want to speak. However, if one or more devices are speaking slowly (slow link speed) then all devices have to wait for them to finish before they can talk REGARDLESS of their own link speeds - this invariably introduces communications lag. This example highlights the fact that even though it's important for a given device to have a strong signal and a fast link speed, it's important for ALL devices communicating on a channel to have a strong signal and fast link speed. As the idiom goes, it only takes one rotten apple to spoil the whole bunch.

Finally the Signal "Bar" Graph attempts to combine the Signal Strength and Link Speed into an easy to understand graphical meter. The more bars, the stronger and clearer the signal and the faster the communications.

NOTE: The Driver Hub has a known bug where the Link Speed indicator only shows the initially negotiated link speed, and the link speed indicated does not change when the Wi-Fi device renegotiates different link speeds. This means the Link Speed indicator and the Bar graph are not represented accurately on Driver Hubs, but are represented accurately on all legal phones.

Week of 02/12/2024 "Robot Wi-Fi Signal Strength"

Welcome to the Tech Tip of the Week! One common question we get is how to determine the Wi-Fi signal strength between the Driver Station and the Robot. Because there are a lot of factors that can play into your robot performance on the field, it's important to know that your robot is getting the strongest Wi-Fi signal possible.

Wi-Fi signal strength is measured in dBm (decibel-milliWatts) and is always negative. Typically the range for Wi-Fi is -30dBm to -90dBm; -30dBm is the maximum possible signal strength, and -90dBm is considered too weak of a signal to support Wi-Fi communications. dBm is measured on a logarithmic scale, so comparing dBm values differs from what you would normally consider on a linear scale. Increments of 3dBm indicate doubling/halving signal strength, and increments of 10dBm indicate 10x change in signal strength. For example, a signal strength of -40dBm is twice as strong as a signal strength of -43dBm, and a signal strength of -67dBm is one-tenth the signal strength of -57dBm. Signal strengths around -40dBm are Amazing, but rarely achievable in match play. A strength of -60dBm is still considered Very Good. -67dBm is considered Good. -70dBm is considered Okay. Anything less than -80dBm is considered unusable.

To see the Signal Strength between your Driver Station and the Robot Controller, first ensure that the robot is connected within the Driver Station App. At the top of the Driver Station App is a readout that shows the connected network name, and under it are Ping times and the Channel number of the Wi-Fi connection. Tap that area of the app, and the display will change and instead show the signal strength under the connected network name. Tap again to swap back.

Knowing your Signal Strength can help you understand how metal on your robot might be affecting your Wi-Fi connection, understand how your robot's signal may vary depending on the orientation of the robot to the Driver Station, and how external factors (like placing your Driver Station on a metal music stand) can degrade the signal strength. Remember that ensuring a strong Wi-Fi signal strength is just one factor in maintaining optimal robot health. Tune in next week to learn about Link Speed, which is the other piece of information provided by the Signal Strength readout.

Week of 02/05/2024 "Gamepad Calibration and Drift"

Welcome to the Tech Tip of the Week. Over the past couple weeks we've had an abnormally large number of questions regarding gamepad calibration hit our support lines, both at FIRST and at REV Robotics, though question submitters had no idea that gamepad calibration was the issue - so let's cover the topic!

How does a joystick know where "center" is on a gamepad? On virtually all gamepads the analog joysticks have an electrical device (usually a potentiometer) that electrically measures the motion of the stick. If the electrical device's value at "center" does not coincide with the value the gamepad thinks should be center, the stick will have a non-zero value at its center position; this is called drift. In a video game, drift is what causes your character to walk left (or right, etc) even though you're not moving the joystick. For a robot, this can cause ghost turning or unwanted motor or servo motion. So how is this "center" value determined?

Some gamepads, like the Logitech F310 gamepads, simply read the value of the analog joystick when it's first powered on and assumes the sticks are always "centered" at that time. If the analog stick is NOT centered when powered on, for example if it's upside down on a table or otherwise resting against something that is deflecting the analog stick, the "center" value will include some amount of drift. In order to correct this, ensure the gamepad analog sticks are centered and simply unplug and replug the joystick. When replugged, the gamepad will again read the current analog stick value as "center" and correct the drift.

Other gamepads, like the Sony DualShock 4 (PS4) or Sony DualSense (PS5), can be calibrated using online tools such as https://dualshock-tools.github.io/ (this is not an official Sony calibration method).

Week of 01/29/2024 "REV Driver Hub Tips"

Welcome to the Tech Tip of the Week! This week is a long one, filled with great REV Driver Hub tips. Most everything here can be found in REV's Driver Hub Troubleshooting tips page, we've just annotated a few of these for the most common scenarios you'll potentially experience with the REV Driver Hub. Understand that this Tip of the Week is not meant to disparage the REV Driver Hub in any way - no device is perfect, but the REV Driver Hub can provide you trouble-free performance if you can understand its nuances and take a few additional steps to keep it running optimally.

1. Make sure your REV Driver Hub time/date is set correctly! This is the cause of a number of inspection nightmares and Robot Controller log file confusion, the first step should always be to check to make sure the Date/Time on the Driver Hub is set correctly. This is set through the normal Android System Settings by pulling down the Android Quick Settings pull-down twice, tapping the Gear Icon, selecting System, and then selecting "Date & Time".
2. USB wall chargers are all the same, right? Wrong. A/C-to-USB wall chargers can range drastically in power (measured in Watts) - the REV Driver Hub comes with an A/C-to-USB wall charger, and that is the recommended wall charger to use to charge the REV Driver Hub. Can you use another device to charge the REV Driver Hub? Maybe, but it's best to stick to either the one that ships with the REV Driver Hub or a fully-charged USB Battery Pack like the Anker 10,000mA Power Bank which can keep a Driver Hub fully charged all day without ever needing to put the Driver Hub to sleep.
3. Rechargeable Lithium batteries don't necessarily work the same way that other batteries work, they all have a slightly different usable Voltage range. The REV Driver Hub needs to calibrate to the Voltage range of the internal lithium battery plugged into it, and to do that there's a full calibration process that has to be followed for any new battery, along with a verification step. DO NOT simply "replace" a drained battery with a new charged one when it gets low, the new battery is NOT guaranteed to have the same calibration as the first and it is not guaranteed to perform optimally. If you're having problems keeping the REV Driver Hub internal battery charged, consider a USB Battery Pack like the Anker 10,000mA Power Bank.
4. Battery safety in any Lithium Battery system is paramount, and the REV Driver Hub has battery safety features that most teams will likely run into at least once. The most commonly experienced safety feature is the Battery Lockout system. If a REV Battery depletes to a level below its recommended safe level, or the battery is overcharged, the REV Driver Hub will enter lockout mode to protect the battery. In this mode, the REV Driver Hub will not power on when the battery button is held down. The process for recovering from Battery Lockout can take several minutes, but it's better than the alternative. It's not recommended to leave a REV Driver Hub on charge unattended for more than 8-10 hours, and definitely NOT for multiple days.
5. When a user puts the REV Driver Hub to sleep, or if it goes to sleep on its own because the Driver Station App main screen is not actively running in the foreground, it goes to sleep pretty easily. However, when the REV Driver Hub returns from a sleep state, sometimes the Wi-Fi and the gamepads will not reload correctly or automatically; this requires you to unplug and replug the gamepads from the REV Driver Hub before you can use them again, or perform a hard reboot in order to bring Wi-Fi connectivity back. Many veteran teams use a fully-charged USB Battery Pack, like the Anker 10,000mA Power Bank, and leave the Driver Station App main screen running all day without putting the device to sleep.
6. Keep the REV Driver Hub safe by using 3M Dual-Lock or hook-and-loop fasteners (like those sold by Velcro Brand) to mount the Driver Hub to a Driver Station Carrier. This prevents your REV Driver Hub from being placed on the floor (where team members may step on it) and prevents you from accidentally dropping the Driver Hub on the floor - dropping the Driver Hub is the #1 cause of all Driver Hub damage! Some teams have designed their own custom Driver Carriers, be creative and have fun!
7. When the REV Driver Hub is not in use (not at competitions, not in use during practices) it should be turned OFF and have all sources of power disconnected. Do not put the Driver Hub to sleep, but actually turn it off - press the power button for 1-2 seconds and then use the drop-down menu to turn off the device. The Driver Hub uses power even in sleep mode, and that can lead to a dead battery and you may have to perform Battery Lockout Recovery before you can turn it back on.
8. Sometimes teams may experience "random power loss" on the REV Driver Hub. This is usually due to a battery fitment issue within the battery box on the device (the battery momentarily stops making a connection with the power pins on the device), and can be mitigated using techniques from the REV Troubleshooting tips. Some teams have been known to operate their REV Driver Hubs without a battery inserted at all, and simply run the Driver Hub using a fully-charged USB Battery Pack, like the Anker 10,000mA Power Bank. The jury is still out on whether that's a good idea, but worth

considering if you're having problems that you're desperate to solve and REV Support is unable to help you resolve (because of time pressures) before your big event.

9. Ensure your REV Driver Hub is fully updated. Firmware 1.2.0 solves a host of REV Driver Hub issues, and it makes sense to use the on-board updater (once connected to Wi-Fi) to perform all updates on the Driver Hub.
10. This isn't specifically a REV Driver Hub tip, but it's a question we get asked all the time. Did you know that the Robot Wi-Fi network name (Robot Controller Name) and the Wi-Fi passwords can be managed straight from within the Driver Station app? With the Driver Station App connected via Wi-Fi to the Robot Controller, click on the three dots menu on the upper-right and select "Program and Manage", then use the hamburger menu on the upper-left and select "Manage". On this page you'll find all of the same settings as you'd find on the webpage by logging in to the controller on a laptop!

Week of 01/22/2024 "REV Grounding Strap"

This week's Tech Tip of the Week is dedicated to the REV Resistive Grounding Strap; the REV Resistive Grounding Strap (RGS) is the only FTC-legal means of providing a grounding option for your robot frame or connected structural elements. Static electricity has two basic behaviors depending on whether it's building up on a conductive or non-conductive surface; on non-conductive surfaces like polycarbonate or other plastics static electricity builds up in "pools", on conductive surfaces like most metals static electricity spreads and distributes across the entire surface of the material. Aluminum extrusion used on robots typically has a clear non-conductive anodized layer used to prevent corrosion of the aluminum but the aluminum under the layer is conductive. When using the RGS, it's important to connect the RGS to surfaces where you want to mitigate static buildup. If mounting the RGS to aluminum on your robot, it's recommended to use a multimeter to test the continuity between the ring terminal on the RGS to different places on the robot to determine if the static buildup will be mitigated by the RGS. If testing for resistivity, remember that the REV Grounding Strap has a 470 Ohm resistor (with a ~5% tolerance) in-line in the strap - if not using an auto-range multimeter, be sure to select a range above 600 Ohms to ensure the resistivity is measured properly. It may be necessary to scrape the aluminum to create a conductive path between multiple segments of aluminum, just remember that a non-conductive oxide layer will eventually form on the exposed aluminum. Remember that if you're probing aluminum extrusion to check for continuity or resistivity, those areas need to be scraped to expose bare metal in order to ensure electrical connectivity. "Jumper wires" screwed to aluminum elements can also be added to ensure conductivity between components.

Week of 01/08/2024 "OnBot Java Backups"

This week's Tech Tip of the Week is for all those who program in OnBot Java. Have you ever been worried that your OnBot Java programs could suddenly magically vanish? Has it ever happened to you? One lesser-known feature of OnBot Java is automatic backups - each time you "compile all" in OnBot Java the system saves a copy of all source code, up to 30 compilations deep. In order to find these backups, you must connect to the Control Hub via USB from a Windows machine and navigate to the "FIRST" folder on the device's internal storage. In this folder you'll find a "java" folder, and within that is the "srcBackups" folder. Here you'll find zip files containing each backup with a time/date stamp. Happy Programming!

Week of 12/25/2023 "Protect your Robot with a Password"

This week's Tech Tip of the Week is a gentle reminder that strong passwords and regular backups make for good competition. Even when competing at a Scrimmage before your competition season starts, be sure to change your Wi-Fi password on your Control Hub from the default password of "password" to something only your team knows. Anyone who knows your password can easily gain access to your robot and change or delete your programs, change important settings, or even force your controller to revert to factory settings! And with that said, it's ALWAYS a good idea to keep backups of your programs - it's especially important to regularly *download all Blocks* and OnBot Java programs that are normally only stored on the robot in case anything happens!

Week of 12/18/2023 "Automatic Auto to Driver Control Program Switching"

Did you know that it's possible for the Driver Station to *automatically load your Driver Controlled OpMode* as soon as your Autonomous OpMode has completed? Lots of teams go into panic mode immediately after Autonomous has completed - they're trying to navigate and select the proper Driver Controlled OpMode, Initialize, and Run the OpMode while also picking up their gamepads and preparing to drive. Skip all that panic and confusion and let the Driver Station queue up your Driver Controlled OpMode for you! This week's Tech Tip of the Week focuses on how to *designate a Driver-Controlled OpMode* that is to be loaded once an Autonomous OpMode has completed. You still have to initialize and run the OpMode at the proper time, but at least the Driver Station can do the heavy lifting of swapping and loading the OpMode for you!

Week of 12/11/2023 "Using Servos with the Control/Expansion Hubs"

In case you missed it (ICYMI) there was a fantastic question on the FTC-QA that prompted an in-depth discussion about servos in FIRST Tech Challenge - the question was in regard to servo compatibility and operation/performance on a REV Control Hub, REV Expansion Hub, and REV Servo Power Module. While the full explanation was too much for a Q&A answer, the complete answer was provided on the FTC-Community forums. If you are using servos (or want to use servos) on your robot, the full answer contains an explanation of how servos are managed on a Control and Expansion Hub that you cannot get anywhere else!

Week of 12/04/2023 "Using Encoders"

This week's Tech Tip of the Week highlights proper encoder use within the FIRST Tech Challenge SDK. Encoders are the devices that track how much a motor shaft has rotated, which the vast majority of motors used in FIRST Tech Challenge have built-in. The encoders on the motors can help track a motor, but they can also be used to help synchronize and control motors via "Motor Modes" built into the Control and Expansion Hub firmware. Did you know that most programmers use these motor modes incorrectly? More on these "Motor Modes" and the correct way to use them can be found on the REV Robotics Encoder documentation.

Week of 11/27/2023 "HuskyLens Intro"

This week's Tech Tip of the Week comes to us from Chris Johannesen, 2023 *FIRST* Tech Challenge Volunteer of the Year and author of many ftc-docs tutorials. Have you heard of the HuskyLens and want to learn how to properly connect one to a Control Hub, learn how to use it to detect Team Props, and use the HuskyLens samples included with SDK 9.0.0 and newer? Chris has this and more in his *HuskyLens Tutorial* on ftc-docs, check it out!

Week of 11/13/2023 "Robot and Driver Station Self-Inspect"

This Week's Tech Tip of the Week is here to help teams prepare for inspection at their events. Aside from making sure that your robot is within the Maximum Starting Size, ensuring that your robot code can correctly pass Field Inspection, and other tasks in the Robot Inspection Checklist, teams need to make sure their robot software and hardware apps are updated to the latest and greatest versions and that their hardware is configured correctly. There is a tool within the Driver Station App 3-dot menu called the "Self-Inspect" feature that can help teams perform a quick check to ensure their hardware and software is configured correctly. Depending on your hardware configuration the Self-Inspect screens may be formatted differently or have different options listed, so *there is a handy reference on ftc-docs* that can help you understand the Self-Inspect tool. Make sure you're ready for inspection!

Week of 11/06/2023 "Driver Hub or Smartphone?"

REV Driver Hub or Smartphone?

This week's Tech Tip of the Week briefly discusses the pros and cons of Smartphones versus the Driver Hub. Which one should you use? Are there hidden benefits or perils for using one over the other?

The REV Driver Hub is the standard *FIRST* Tech Challenge Driver Station hardware device. It boasts three USB-A ports for plugging in gamepads, a USB-C port used for communication and charging, a large touch screen, and an unused Ethernet port (for future-proofing). This device runs the Android operating system, maintained by REV Robotics, and uses Wi-Fi to communicate with the REV Control Hub.

Driver Hub Pros

- Driver Hub and Control Hub combo use 802.11w for communications. No approved Smartphone supports 802.11w communications.
  - 802.11w offers encryption of control packets, which prevents many Wi-Fi attacks by remote routers/devices.
- Driver Hub is a "standard" *FIRST* Tech Challenge Driver Station device, which provides long-term support for *FIRST* Tech Challenge. The average SmartPhone is deprecated within 2 years after being released, but the Driver Hub is supported as long as it's legal to use in *FIRST* Tech Challenge.
- Driver Hub has a USB-C port, which allows for charging while it's being used.
  - USB-C port allows use of external battery packs, which are necessary for sustained use of PS4 and PS5 gamepads which leech power from the Driver Station to charge their own internal batteries.
  - A single 10,000mAh External battery pack allows Control Hub to be used non-stop over the course of an entire day.
- Driver Hub has 3 USB-A ports, so no external USB hubs and additional cables are required for using multiple USB gamepads. This makes the Driver Hub very compact and easy to manage.

Driver Hub Cons

- Driver Hub still has Power Management issues
  - Driver Hub needs battery compartment tweak to ensure internal battery makes good connection. Foam insert in battery compartment helps, but doesn't always perfectly fix the problem.
  - Driver Hub cannot boot if the internal battery is too low, even if plugged into external battery. If battery dies, troubleshooting requires removal of battery to power device.
  - Power Management bugs can drain battery while charging.
- Driver Hub USB ports are fragile
  - Teams carrying their Driver Hubs around without a Driver Station tray (NOT RECOMMENDED) have dropped their Driver Hubs with gamepads plugged in, and impact can damage USB-A ports.
- Display screen ribbon cable comes loose
  - If the screen stops working, opening the back of the device and re-seating the screen ribbon cable can sometimes fix screen issues.
- Turning off the display unloads gamepad drivers, but turning the display back on does not reload them. USB devices must be re-plugged in order to trigger USB driver loading.
- USB-C to USB-C cables do not work with Driver Hub. USB-A to USB-C cables are required in order to use the USB-C port.

On the other hand, several off-the-shelf SmartPhones are supported, including the Motorola Moto E4 and Moto E5 phones. These devices, like the REV Driver Hub, run the Android mobile operating system and use Wi-Fi to talk to the REV Control Hub (therefore no SIM card or cell plan is required). SmartPhones use USB-OTG to interface with gamepads and external USB hubs necessary for operating multiple gamepads.

SmartPhone Pros

- SmartPhones are typically cheaper than Driver Hubs, and generally survive being dropped better.
- SmartPhones don't have the same power management issues that Driver Hubs are known to have.
- Some teams report having better Wi-Fi consistency with SmartPhones than Driver Hubs, though that has not been verified or debunked in any way.

SmartPhone Cons

- There are only a small number of approved Android Smartphones, none of which are still supported by the manufacturers of the phones.
  - SmartPhones are deprecated typically within 2 years after being released. Security updates and OS updates are not guaranteed.
  - The number of approved SmartPhones are dwindling, and SmartPhones are becoming increasingly difficult to obtain. New SmartPhones are not being approved to replace older ones.
- Android is not a consistent platform in the Mobile Phone industry. Each manufacturer, and sometimes even within product families, will produce their own "flavor" of Android which has different software requirements and behaviors. Supporting the different manufacturers in the changing Android landscape is near impossible.
  - There is very little consistency between smartphones of the same model sold in different countries - each will have their own firmware with their own quirks, often impossible to debug or avoid.
  - *FIRST* Tech Challenge is not enough of a volume consumer to be able to set requirements or have partnerships with SmartPhone manufacturers.
- SmartPhones cannot use 802.11w for encryption of Wi-Fi control packets, which makes the connection between devices vulnerable. Rogue Access Point Detection and Quarantine features within venue network security systems (like within schools and other venues) can interrupt these communications seemingly randomly, making connections difficult to maintain.
- SmartPhones cannot be used at the same time they're being charged, so teams frequently run down the internal batteries on the phones during the course of an event. Careful battery management is required.
  - PS4 and PS5 gamepads with internal batteries will further drain the SmartPhone batteries, as they leech power from the Driver Station in order to maintain a full charge level for their own batteries.
- SmartPhones require USB-OTG cables and external USB Hubs are also required in order to use multiple gamepads, and each cable/connection and device is a potential source of failure. Extreme care must be taken to ensure the connections remain solid.

Week of 10/30/2023 "Computer Requirements"

This week's Tech Tip of the Week focuses on required computer hardware for *FIRST* programs. If you're looking to buy a laptop and want to make sure you meet the minimum requirements for the program you're participating in, like *FIRST* Tech Challenge, this tech tip is for you! There is a new *Computer Requirements* document on ftc-docs that provides a cross-program view of the laptop requirements for all *FIRST* programs. It also has examples of the different laptops and a list of the required features needed for each program. Check it out!

Week of 10/23/2023 “Control and Expansion Hub Tips”

This week’s Tech Tip of the Week provides useful tips when using Control and Expansion Hubs.

- The RS485 data cable ports that provide data between Control and Expansion Hubs are redundant - you can use two data cables utilizing both ports to ensure that if one cable fails communications aren’t lost.
- Encoder ports 0 and 3 are hardware-counted, but ports 1 and 2 are software-counted. This means higher counts-per-revolution encoders (like the REV Through-Bore Encoder) should be placed on Ports 0 or 3 to ensure counts aren’t missed, and lower counts-per-revolution encoders (like the goBILDA Odometry Pods or most motors) can be connected to any port.
- Servo port pairs (0,1), (2, 3), and (4,5) each share a common power supply, so if you’re using higher-current servos (like a goBILDA torque servo) directly on the Control or Expansion Hub you should only use ports (0, 2, 4) or (1, 3, 5) in order to maximize the power available to each servo. If you need to use more than 3 high-current servos per hub, consider using a REV Servo Power Module.
- Each Digital and Analog sensor connector on the Control and Expansion Hub each have 2 signal channels. Some REV sensors are only designed to be configured and used on the N or N+1 channels. Read the documentation for each sensor carefully!
- The USB 2.0 port shares the same USB bus as the internal Control Hub radio. ESD or other electrical interference that affects devices (like webcams) plugged into that port may cause a loss of communications. When using a USB webcam, use the USB 3.0 port first.
- USB C-to-C cables do not work properly with the Control Hub, only USB A-to-C cables do.
- If you’re utilizing the onboard IMU, Do not plug I2C devices into Port 0 unless absolutely necessary. Port 0 shares an I2C bus with the IMU, and misbehaving devices (or devices that don’t “play well with others”) plugged into Port 0 can cause the IMU to stop communicating.

Week of 10/16/2023 “Battery Maintenance Tips”

This week’s Tech Tip of the Week is an extension to our first-ever Tech Tip of the Week regarding battery maintenance. Nickel-Metal Hydride (NiMH or Ni-MH) batteries, like those used in FIRST Tech Challenge, do require periodic maintenance to keep them healthy! Every day, NiMH batteries lose on average 1% of their charge capacity at normal room temperature - at colder temperatures this decline slows a bit but does not stop it. This means that every 2-3 months it’s important to recharge your batteries to keep them healthy - there is no off-season for batteries! It’s also recommended to mark your batteries with tape and a sharpie to mark (1) Your team number (never lose a battery at a competition!), (2) What year the battery was purchased, (3) Give your batteries names so you can differentiate batteries easily, and (4) optionally provide a tick mark each time the battery is recharged. NiMH batteries can generally last 200-300 recharge cycles before their internal resistance declines to the point where it’s time to replace them, and keeping track of charge cycles is an easy way to track how “used” the battery is before needing to have its internal resistance checked.

Week of 10/09/2023 “Hardware Connection Diagrams”

Have you ever asked, “How does that get connected?” when working with *FIRST* Tech Challenge control system components? This Tech Tip of the Week highlights Stefen Acepcion of *FIRST* Robotics Competition Team 3161 - he has graciously compiled several connection diagrams that demonstrate different ways that common components can be connected within the *FIRST* Tech Challenge control system. *Driver Station connection diagrams* (both Driver Hub and Android Smartphone configurations) and *Robot Controller connection diagrams* (both Control Hub and Android Smartphone configurations) can be found on ftc-docs. Stefen has contributed additional diagrams this season, including a new Advanced REV Control Hub connection diagram and a new Advanced Smartphone connection diagram. These diagrams are chock full of helpful tips, connection techniques, and information you otherwise can’t find in one place - check them out!

Week of 10/02/2023 "Choosing the right Webcam and Calibration Crowd-sourcing"

When using AprilTags, choosing the right webcam can save you from having to *perform your own calibration* before being able to use it for obtaining *AprilTag Pose information*. This week's Tech Tip of the Week explores the new *Webcams for VisionPortal* document that highlights several commonly used webcams that have calibration data built-in to the SDK itself. Maximum frame rates, field of view, and supported resolutions with calibration data are all covered for each of the most common webcams in *FIRST* Tech Challenge. Short on time? Be sure to check out the handy *quick summary* at the bottom of the page! Did you calibrate your own camera and determine lens intrinsics for it? Please check out this FTC-Community post to contribute to the crowd-sourcing effort for calibration data!

Week of 09/25/2023 "3D Printing Resources on FTC-Docs"

Do you wish you knew more about 3D printers, filament, and choosing and maintaining a 3D printer? This week's Tech Tip of the Week highlights ftc-docs community contributions from *FIRST* Tech Challenge teams 16461 and 1002 introducing *3D printing in FIRST Tech Challenge*. Once you've got a 3D printer, be sure to check out *Computer Aided Design (CAD)* also on ftc-docs to find a CAD package and start designing and printing parts for your robots!

Week of 09/18/2023 "Technical Update video by AJ Foster"

This week's Tech Tip of the Week is a Video Tech Tip of the Week from AJ Foster, *FIRST* Tech Challenge World Championship FTA and Orlando Robotics League All-Star Volunteer. AJ gives a great synopsis on many of the key technical updates for the CENTERSTAGE presented by RTX season and some background on those changes. Watch his video on the *FIRST* Tech Challenge YouTube Channel here: https://www.youtube.com/watch?v=uOcVGwdhG3E&feature=youtu.be.

Week of 09/11/2023 "Updating the Robot Controller App"

This week's Tech Tip of the week is all about updating software on your Control Hub. If you use *Android Studio*, did you know that you're not supposed to use the REV Hardware Client to update the Robot Controller (RC) App? Blocks and OnBot Java programs are stored on the Robot Controller (SmartPhone or Control Hub) differently than Android Studio programs, and this has a major effect on how updates can be managed on the device. Read more about this at *Updating the Robot Controller (RC) App*.

Week of 09/04/2023 "Battery Charging"

The *FIRST* Tech Challenge Tech Tip of the week this week is all about Battery Charging. There are *three robot main batteries* that are legal to use in FIRST Tech Challenge, and they are all 3000mAh NiMH batteries with an attached 20A fuse. However, the manufacturers of the batteries have different battery chargers and different recommended charging settings for the batteries. When charging the TETRIX MAX 12-Volt battery, on the battery the manufacturer recommends charging at the 0.9A charge rate (the lowest setting on most selectable battery chargers) using the Global NiMH battery pack charger. The Matrix 12-Volt battery with the same form factor is recommended to be charged with the goBILDA 12V battery charger, which does not have a user-selectable charge rate switch but has a max charge rate of 1.0A. However, the REV 12-Volt Slim Battery is recommended to be charged with the REV Battery Charger using the 1.8A charge rate setting. To ensure safety, proper charging, and a long battery life, make sure you're charging your batteries at the manufacturer's recommended charge rates!

Chapter 9

AI Innovation Corner

These articles are part of the *FIRST* Tech Challenge AI Innovation Corner. This is a place where we'll post custom and curated articles relevant to *FIRST* Tech Challenge as it relates to AI and its impact on our daily lives and the world around us. We would like to thank Google for their generous contributions to *FIRST* Tech Challenge to increase access to our program in underserved communities and for providing sponsorship and occasional technical direction for this content.

Articles are ordered on this page chronologically, with the newest content at the top of the page expanded. Just click to expand any other articles you'd like to see.

Week of 10/21/2024 "AI for Social Media"

AI for Social Media

This week in this AI Innovation Corner we're going to be exploring using AI for helping you take the first steps that every *FIRST* Tech Challenge team needs to do, which is to increase your team's social media presence. Specifically, we're going to focus on social media campaigns and team marketing, which can be a critical step to increase the visibility of your team in the community. This can help you with recruitment of team members and mentors, spreading awareness of yourselves and *FIRST*, and is one great step towards helping you with fundraising (which we'll cover in more detail in a future article).

If you've never started a social media campaign for a *FIRST* Tech Challenge team, you might have some ideas about how to get started but you need some direction and some more "concrete" examples pertaining to where to start, what to create content about, and even where to post/host your content. Gemini can be a great start in your brainstorming process. This simple prompt can yield loads of valuable feedback:

- *I have a high school FIRST Tech Challenge Team. What are things I need to consider when starting my first social media campaign?*

Once you've got a presence and start working through content, through social media you'll engage with lots of folks. It can sometimes be difficult to find the right way to interact with the community at large, and responders who love (or are critical of) your content. AI to the rescue! AI can help you find different ways of interacting - though remember the ultimate responsibility on what to post is up to you, AI should only be used as a means of brainstorming and/or refining what you want to say.

- *Write a friendly, one-sentence response to the following social media comment that shows appreciation for the comment and encourages the user to turn on post notifications for more content from our FIRST Tech Challenge team.*

Can't think of prompts to ask/use, or are you looking for some inspiration? AI can help you there as well. You can ask AI to help you with building AI prompts (asking the AI what to ask the AI is a pro tip!), brainstorming ways to use AI for social media, and ways to accentuate what you do on social media. Be sure to keep " *FIRST* Tech Challenge " in your prompts so that the AI relates what you're asking it to the competition so that the suggestions it makes are relevant to your team and your activities.

- *Can you tell me some common AI prompts for generating social media posts for my FIRST Tech Challenge team?*

Finally, it's important to track your social media progress in order to determine what's going well, optimize your strategy for content in the future, and determining where you're getting the most "bang" for your content-production "buck." We can use the following prompt to ask AI to give us some tips on the best ways to track our online footprint:

- *How can I track my team's progress on social media?*

Remember, social media is a marathon, not a race. It requires careful planning, continuous improvement, consistency in creating content, and the desire to build your team's brand. Hard work and determination can pay off, and AI can help guide you along the way!

Week of 09/30/2024 "AI Competition Manual Assistant"

AI Competition Manual Assistant

In our first article, the Google AI Studio was introduced as a tool to interact with Google's Gemini AI. Gemini is one of several flagship Large Language Models (LLM's) that have been meticulously trained on massive amounts of text data to learn the patterns and relationships between units of language - these models have actually learned how to recognize text-based language, read and understand data, and synthesize what it learned to predict and interpret future data. This is the exact process humans make in learning and understanding the world around us! In Google AI Studio, users can interact with the Gemini AI through "prompts" to perform tasks for them. Prompts are instructions or queries given to an AI in order to generate a response - the quality of the response is often directly related to the quality of the prompt. Through these prompts, Gemini can provide responses based on the massive dataset that it has been pre-trained with, or users can also provide additional documents, text, or media that the AI has never seen before. These multimodal prompts, or prompts that include multiple types of content, can be very beneficial in interacting with an AI using content that is specific to a niche area like *FIRST* Tech Challenge. Can you think of ways to put this ability to good use in *FIRST* Tech Challenge?

In *FIRST* Tech Challenge, one of the first tasks teams have to do is to read and understand the *FIRST* Tech Challenge Competition Manual. This can be a very painstaking task, and even a skilled reader can miss subtle nuances provided by the manual. However, an AI can break down and analyze the manual in a matter of seconds, usually preserving the nuance provided in the document. Users can then interact with the AI that has analyzed the Competition Manual, and prompt the AI to provide insights - these questions might involve locating specific information likely found in the Competition Manual, summarize important rules or processes, or even involve asking the AI to make a best guess. Through a process known as "role playing" the user can prompt the AI to take on a role or persona and direct the AI to follow specific rules as it interacts with the user in subsequent prompts. The remainder of this article is a tutorial on how to set up a "role playing" session with the Google Gemini AI through Google AI Studio to analyze and answer questions based on the *FIRST* Tech Challenge 2024-2025 Competition Manual for the INTO THE DEEP presented by RTX season. While some of the nuanced elements (like AI prompting) will be shallowly covered in this article, it is something we'll cover a lot more in future articles.

Creating an AI expert using Google AI Studio is fairly straightforward - the hard part is creating the proper prompt, and there we've got you covered.

Step 1 - First, log into Google AI Studio. You can do this by clicking the "Sign in to Google AI Studio" button on the front page of the Google AI Studio home page. You will need a Google account in order to do this - getting one is left as an exercise to the reader. The Google account is used to store your Google AI Studio prompt sessions and any content you upload to the model, and to track usage of the Gemini APIs.

Step 2 - Let's download the *FIRST* Tech Challenge Competition Manual to your local computer. You can always find the latest Competition Manual PDF at the following link:

- https://ftc-resources.firstinspires.org/file/ftc/game/manual

Step 3 - In the left navigation pane towards the top of the pane, there is a circle with a plus inside it with the text "Create new prompt" next to it. Clicking on this button will start a new prompt - though if you're using Google AI Studio for the first time it's likely a new prompt is already open.

Now that we have a new prompt, you can give the prompt a name. This will allow the prompt to be saved in your "My Library" so you can come back and interact with the prompt later without having to recreate the prompt session every time.

In the bottom center of the workspace is a text field where you can enter in your prompt (it has a default prompt of "Type something"). BEFORE we enter our prompt, we want to add our Competition Manual PDF document. To add the document, click the "Plus" icon to the right of the prompt area. This will give you several options, choose "Upload to Drive". You can either click the "Browse" button to browse for the PDF of the Competition Manual that you downloaded, or you can drag the file into the window. This adds the Competition Manual to your prompt, it may take a minute or two to upload the PDF so please be patient.

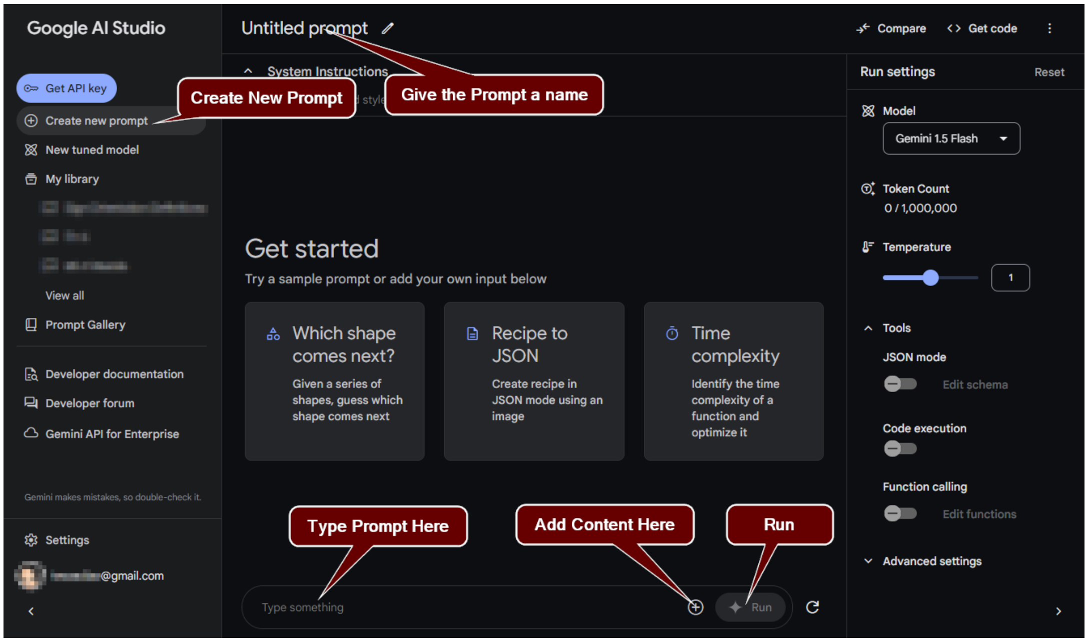


Fig. 1: Creating a prompt in Google AI Studio

Step 4 - Now that we have our document uploaded, we now want to enter our prompt. This prompt directs the AI in how to manage its responses, what information to use when developing a response, and sets up the role that the AI will attempt to play. Enter the following prompt and press the "Run" button:

- *You are a helpful AI assistant providing answers to questions about the provided PDF. Do not use any prior knowledge; you have everything you need to answer questions in the one PDF provided. Cite all references.*

Once the AI processes the initial prompt, we can then ask questions that the AI will use the Competition Manual to answer. Depending on the question, it may take the AI between several seconds up to a couple minutes to answer - be patient! Here are several questions you can ask (remember to press the "Run" button after asking each question):

Example sample questions:

- How many SAMPLES is a ROBOT allowed to CONTROL at a time?
- What are the different ways to score points?
- How large can a ROBOT be in its STARTING CONFIGURATION?
- Which awards are best for advancement?
- How do I write a strong engineering portfolio?

Some prompts that require a lot of complex understanding or strategy can yield results that are not correct, especially if there is information "understood but not supplied." For example, the following prompts provide some correct and some incorrect information:

Examples of difficult questions:

- What is the maximum score for an alliance?
- Can ROBOTS pick up an opposing ALLIANCE'S SAMPLES?
- How many matches does a team play at an event?

This example was specific to FIRST Tech Challenge, but this process can be used for virtually any documents or media. Using AI as an analysis assistant can help you summarize news articles, find specific instructions in user manuals, review books, and more! Remember that the quality of the responses the AI provides is directly related to the quality of the prompt provided - even so, the AI isn't always going to be able to provide correct answers so it's up to you to verify the correctness of all answers provided by an AI.

Week of 09/09/2024 "AI Innovation Corner - Google AI Studio"

AI Innovation Corner - Google AI Studio

This first article launched as part of the *Tech Tips of the Week*, but is the official first article for the AI Innovation Corner.

This week's Tech Tip of the Week launches a new initiative in *FIRST* Tech Challenge, an AI Innovation Corner. Generative AI has taken the world by storm, becoming commonplace now in everything from personal assistants, search engines, recipe curation, music innovation, and vehicle maintenance! Machine Learning AI has been a part of *FIRST* Tech Challenge in some way for the past six years, and we're now transitioning to help teams learn how to use and incorporate Generative AI in their *FIRST* Tech Challenge experience (while we're learning ourselves!).

The first step (or *FIRST* step?) to getting the most out of AI is choosing a model. What do I mean by model? Every AI is a neural network that has been trained with specific knowledge with the ability to do specific things based on that knowledge. Each version of this neural network is stored in a "model". Each different company has different models available for different purposes, though most models are variations on their flagship model (Gemini from Google, ChatGPT 4-o from OpenAI, Claude from Anthropic, and so on). Each company has different web-based and API interfaces for interacting with their models, and everyone has their favorite. In *FIRST* Tech Challenge, the standard tool we use is Google AI Studio to interact with Gemini.

Google AI Studio is free to use, but requires a Google account to access - virtually all models require a login or API token of some kind to use. Google AI Studio is our favorite for its list of examples (Prompt Gallery) and its easy to use interface to save prompt sessions and resume them later. With Google AI Studio, you also can select the specific model you want to use, and when available you can choose to use preview versions of up and coming models.

Chapter 10

Game Manuals

Game Manuals can be found on the Game and Season Materials page on the *FIRST* Website.

Chapter 11

FIRST Tech Challenge Game Q&A

The Game Q&A is a forum/tool that provides teams an opportunity to receive clarifications from the Game Design Committee about the current season's challenge. Rulings on the Q&A are final and binding, so all teams must make sure to check back and review the Q&A often. Game rulings made within the Q&A system are NOT guaranteed to be reflected in the appropriate Game Manual.

11.1 How to Ask Questions

Once the Game Q&A opens for the season (usually within 2-3 weeks of the challenge being announced) teams may ask questions using unique accounts provided to teams via their FIRST Dashboard accounts. Teams must use these credentials to log into the system, and then may ask questions. Anyone may create a personal account and view or tag questions, but only accounts provided to teams may ask questions.

Obtaining the team credentials for the question-asking team accounts can only be done by the Lead Coach 1 or 2 by selecting "Passwords/Voucher Codes" from the "Payment & Product" drop-down in the "Team Options" column of the team information in the FIRST Dashboard. The Q&A website credentials for the team will be listed under the "Game Q&A Forum Accounts" section of the resulting webpage.

11.2 Game Q&A Summary

The Game Q&A tool periodically provides updates to their "one-page summary" of all answered questions. This one-page summary is not guaranteed to contain all questions and answers, and is also not guaranteed to be updated on a regular basis, but it is the best way to obtain a printable format of the questions and answers on the forum.

Chapter 12

Playing Field Resources

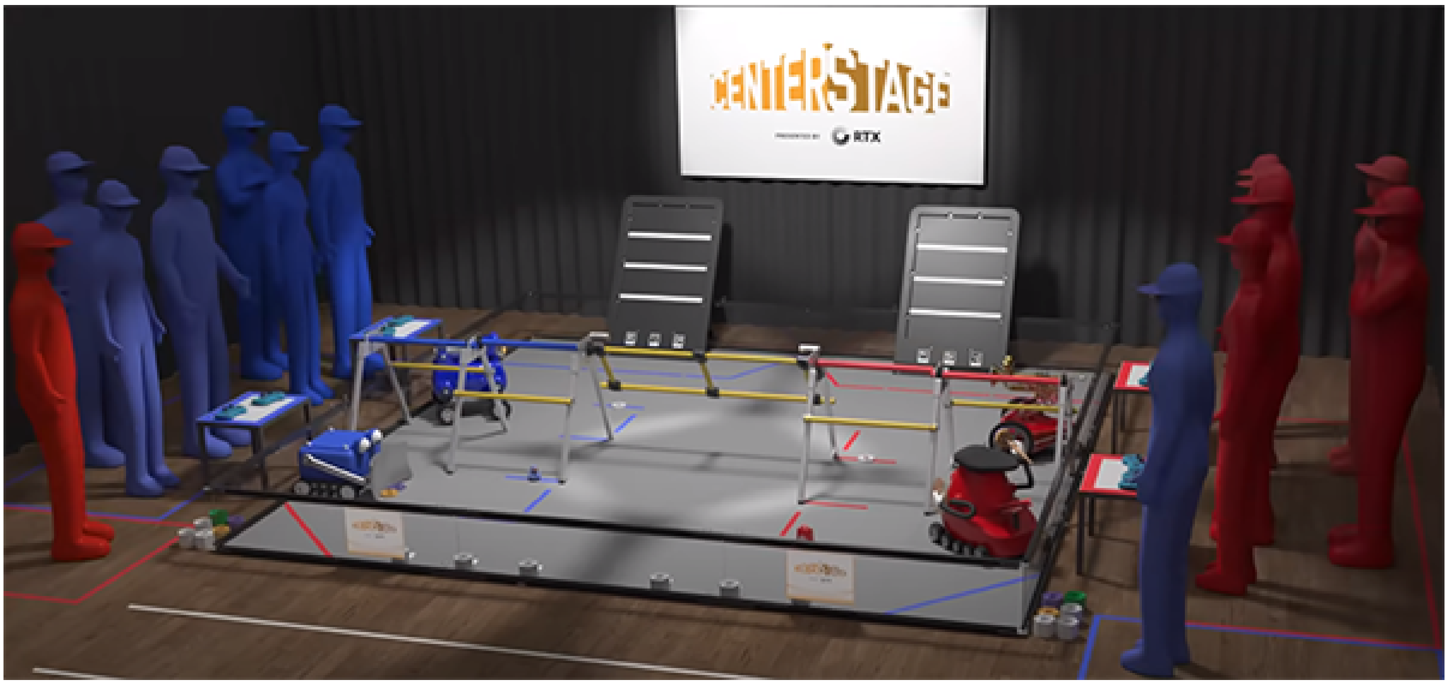


Fig. 1: Traditional Playing Field, CENTERSTAGE presented by RTX, 2023-2024

12.1 About the Playing Field

There are multiple configurations of the playing field that can be used. For traditional games, the playing field is a part of the Arena that includes the 12 ft. x 12 ft. (3.66 m x 3.66 m) field and all the elements described in the official field drawings. The base field stays the same for all games but the game elements are subject to change as per the Competition Manual.

The Competition Manual contains an Arena section that details the playing field for that years game. It includes measurements for key aspects of the field and game elements and scoring elements. For example, the height of a basket into which a scoring element can be placed. The Competition Manual can be found on the Game and Season Materials page on the *FIRST* Website.

12.2 Field Setup Guide

The Field Setup Guide has the official instructions for assembling and setting up a *FIRST* Tech Challenge field. Typically there are assembly instructions that build structures that then have setup instructions for placing on the field. There are also teardown instructions that indicate how to take apart the field for storage or transport.

The guide typically has the following sections:

- A list of all tools required for assembly and setup, some tools are only for assembly or for setup.
- Lists all the game elements and scoring elements with the quantity of each.
- Instructions for setup of the field perimeter and field tiles.
- Step by step instructions for assembling parts and setting them on the field.
- Most games have tape lines on the field to mark locations or areas of the game. There are also taped areas outside the field for the Alliances, and sometimes for game areas.
- Most games have AprilTags placed around the field that can be used for robot navigation.
- Finally, there are teardown instructions that indicate how to take the field down for storage or transport.

Use the following button link to download a PDF of the current Field Setup Guide from the *FIRST* Website:

Download Field Setup Guide PDF, 4.5 MB

Note: The Field Setup Guide has instructions for assembling an official game set as purchased from AndyMark.

A purchased game set can be full or partial. A partial game set is less expensive and also suitable for teams who want official game elements but don't have room to setup a full field.

The Game and Season Materials page also contains a downloadable PDF for the AprilTag images that can be printed and placed on the field. There is a Do It Yourself (DIY) Resources section that includes CAD models of scoring elements and DIY field and perimeter build guides.

Chapter 13

FIRST *Tech Challenge Field Coordinate System Definition*

Summary: The *FIRST* Tech Challenge Field Coordinate System is a Cartesian Coordinate System of three dimensions. The X and Y axes will refer to a position on the field and the Z axis a height above the field.

13.1 Scope

This document defines the Field Coordinate System for a *FIRST* Tech Challenge playing field. This definition can be used for consistent field-centric navigation, target localization and path planning.

13.2 Reference Frame

The reference frame for this definition is the field perimeter wall, adjacent to the red Alliance Area, known here after as the Red Wall. The definition is from the perspective of a person, standing outside the field, in the center of Red Wall, looking towards the center of the field.

Note: If the red Alliance Area is ever adjacent to two perimeter walls, the Red Wall will be the one with *most* contact with the Alliance Area. If the red Alliance Area is ever adjacent to two perimeter walls *equally*, then the most clockwise of the two walls will be considered to be the Red Wall.

13.3 Coordinate System

The Field Coordinate System is a Cartesian Coordinate System of three dimensions. X and Y will refer to a position on the field. Z will refer to a height above the field. You may use any length measure as long as the same measure is used for all three axes. The coordinates are ordered (X, Y, Z). Example: coordinate position (10, -10, 0) has X = 10, Y = -10 and Z = 0.

13.3.1 Origin

The 0,0,0 origin of the *FIRST* Tech Challenge coordinate system is the point in the center of the field, equidistant from all 4 perimeter walls (where the four center tiles meet). The origin point rests on the top surface of the floor mat.

13.3.2 X Axis

Looking at the origin from the Red Wall, the X axis extends through the origin point and runs to the right and left, parallel with the Red Wall. The X axis values increase to the right.

13.3.3 Y Axis

Looking at the origin from the Red Wall, the Y axis extends through the origin point and runs out and in, perpendicular to the Red Wall. Increasing Y values run out (away) from the Red Wall.

13.3.4 Z Axis

Looking at the origin from the Red Wall, the Z axis extends through the origin point and runs up and down in a vertical line. Increasing Z values extend upwards.

13.3.5 Rotation About Axes

When considering rotations about an axis, consider yourself looking down the axis from the positive end towards the origin. Positive rotations are then counterclockwise and negative rotations clockwise.

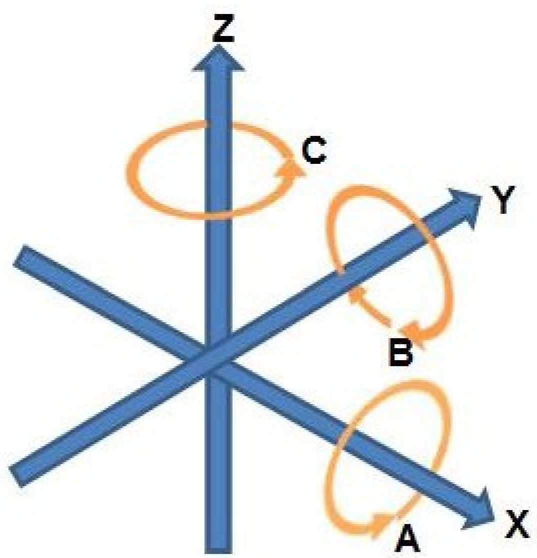


Fig. 1: Counterclockwise rotations about each axis
Imagine looking down the positive Z axis towards the origin. This would be like standing in the middle of the field looking down. A positive rotation about the Z axis would be counterclockwise.

Example: a robot spinning clockwise on the Field is making a negative rotation about the Z axis.

13.4 Field Configuration Examples

Below are two examples illustrating the Field Coordinate System for different *FIRST* Tech Challenge field configurations.

Note: In both field configurations the red Alliance is facing out along the positive Y axis, and the Z axis points up from the center of the field.

13.4.1 Diamond Field

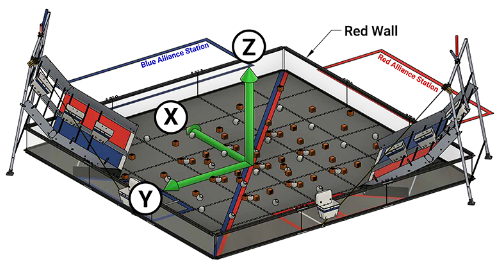


Fig. 2: The FIRST RES-Q game field

In a diamond field configuration the two Alliance walls are adjacent. The field is rotated 45 degrees such that both Alliances face the audience. From the audience perspective the field forms a diamond shape. The Red Wall will be on the right as seen from the audience. The Y axis points across the field as seen from the Red Wall. The X axis points to the Blue Alliance.

13.4.2 Square Field

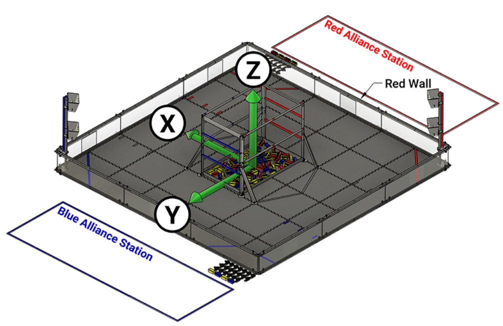


Fig. 3: The Into The Deep game field

In a square field configuration the two Alliances face each other across the field. The field is oriented such that the Red Wall is on the right as seen from the audience. The Y axis points across the field from the Red Wall towards the Blue Alliance. The X axis points away from the audience to the rear of the field.

13.5 Coordinate Position Example

Let's consider the coordinates (0, -24, 26) in inches on the Into The Deep field, which is a square field. Given the order of coordinates then X = 0, Y = -24, and Z = 26.

The X axis value of 0 is located at the origin in the center of the field. The Y axis value of negative 24 would be located closer to the Red Wall, away from the origin by the width of one tile. This the center of the wall of the submersible structure on the red side of the field.

The Z axis value of 26 indicates the coordinates refer to the center and top of the Red Alliance "high chamber" (which is the higher of the two red crossbars).

13.6 Measured Values

The following metric values have been measured from a 2016 competition field. They are representative only, and should not be assumed to be exact, or guaranteed.

- Distance between opposite inside faces of panels: 3580 mm, (if the field is assembled well: the straps give some adjustment tolerance)
- Polycarbonate transparencies have a visible opening height of 255 mm
- The top edge of transparencies is 30 mm from the top of the perimeter
- Total perimeter height is 313 mm
- Tiles are 13 mm thick

So, for a diamond field configuration, the corner of the field closest to the audience, at a height equal to the top of the perimeter wall, would have a coordinate position of: (-1790, 1790, 300) in millimeters.

13.7 Additional Information

See this Wikipedia article on Cartesian coordinate system in three dimensions. The Field Coordinate System rotation convention comes from the right hand rule of classic geometry.

Robots with a webcam can use *AprilTags* to determine where an *AprilTag is located* with respect to the robot. Since AprilTags are in known locations on the field, you can also determine the *location of the robot* on the field.

Robots can use an inertial measurement unit (IMU) to measure rotations about axes with respect to the robot. See *IMU axes definition.* The yaw value from the IMU, also known the heading, measures rotation about the Z axis which points up from the robot. You can use the IMU to determine which direction a robot is facing.

Chapter 14

Computer Requirements for FIRST *Programs*

FIRST® programs, such as *FIRST*® LEGO® League, *FIRST*® Tech Challenge, and *FIRST*® Robotics Competition, are as unique as the teams that participate in them. This uniqueness is partially due to the wide variety of vendors that provide technologies to the programs, the hardware and software necessary to manage each program's distinctive goals, and the constantly evolving landscape of tools and techniques that teams find useful to participate and excel. One commonality between programs is the need for teams to have a computer platform for software development, design, and collaboration. This document serves as a recommendation for the hardware and operating system requirements for that computer system.

Of the many factors that can affect the minimum requirements for a computer, these are the ones that affect those requirements most heavily:

- Any role-specific tasks that the computer may perform in the program
- Type of Computer-Aided Design (CAD) software that may be used on the computer
- Software development and hardware update requirements
- Vendor-specific application requirements and limitations

14.1 Program-Specific Requirements

Each program has a unique set of requirements, but each of those requirements can be met with a minimum computer configuration. This section attempts to identify the minimum requirements for each program's roles. Specific recommended hardware that meets each of the requirements are listed in the "Recommended Hardware Sets" section.

14.1.1 Recommended Computer Hardware for *FIRST*® LEGO® League

FIRST LEGO League has two divisions which use programmable platforms: *FIRST* LEGO League Challenge which uses the LEGO® Education SPIKE™ Prime platform, and *FIRST* LEGO League Explore which uses the LEGO® Education SPIKE™ Essential platform. Both platforms have virtually the same computer requirements, differences are noted below. These platforms are some of the most accessible, as they are supported by most computer configurations.

Recommended for Software Development:

- *Windows Standard Laptop*

Also Supported:

- *MacOS Standard Laptop*
- *Chrome OS Standard Laptop*
- *iOS Standard Tablet*
  - LEGO® Education SPIKE™ Essential hub cannot be updated with iPad

- *Android Standard Tablet*
  - LEGO® Education SPIKE™ Essential not supported

It is also recommended to have an active internet connection to access the Google Play Store (for Chromebook Android apps), to download in-app content, to access teacher support materials, and to use certain features such as live weather data.

14.1.2 Recommended Computer Hardware for *FIRST*® Tech Challenge

The predominant hardware platform used in *FIRST* Tech Challenge is the REV Control Hub and REV Driver Hub. These platforms have unique operating system and application requirements, though it is possible to perform most of the basic functions with most hardware platforms (albeit with more manual steps). Teams in *FIRST* Tech Challenge use computers for two basic purposes – software development and CAD – and team preference in these two uses shapes the required hardware.

Recommended for Software Development and CAD:

- *Windows Performance Laptop*

Recommended for Software Development Only:

- *Windows Standard Laptop*
  - Only cloud CAD solutions are recommended
    - OnShape, SolidWorks 3D Experience, etc.

Also Supported:

- *MacOS Standard Laptop*
  - REV Hardware Client not supported
    - Must update manually using browser-based interface
- *Chrome OS Standard Laptop*
  - REV Hardware Client not supported
    - Must update manually using browser-based interface
  - Android Studio not supported
    - Only Blocks and OnBotJava supported

It is also recommended to have an active internet connection during software development. Access to https://github.com is required by the REV Hardware Client to download and install required season software updates and is required for Android Studio users to download software templates.

14.1.3 Recommended Computer Hardware for *FIRST*® Robotics Competition

The predominant hardware platform used in *FIRST* Robotics Competition is the NI roboRIO. This platform has a unique set of requirements for computer hardware in competition that may be different than requirements for software development depending on the programming environment. Like *FIRST* Tech Challenge, teams in *FIRST* Robotics Competition use software development computers for two basic purposes – software development and CAD – and team preferences in these two uses shape the required hardware. However, in *FIRST* Robotics Competition there are two roles that computers can serve, such as Software and Design Development platforms and/or Driver Station platforms, and those roles also shape the requirements of the computer hardware.

It is recommended to have two separate computers, one to use for the Driver Station platform and another for Software and Design Development, though one laptop can be used for both purposes if necessary.

Driver Station

The driver station computer is used as the primary interface to the robot, is used to interface with the Field Management System (FMS) at an event and is limited by the software tools used to communicate with the hardware and software platform on the robot. Teams find it advantageous to have separate computers for the Driver Station role and for the Software and Design Development role to enable them to segregate the duties and physical demands of the systems at an event. Budget-conscious teams can certainly use a single computer for both roles if that computer at least meets the minimum requirements of the Driver Station role. Note that the Driver Station role requires a Windows operating system, as the applications required to perform the role's duties are Windows-only applications.

Recommended for the Driver Station role:

- *Windows Standard Laptop*

Also Supported:

- *Windows Performance Laptop*

Software Development and Design

Like *FIRST* Tech Challenge, *FIRST* Robotics Competition teams use Software Development and Design laptops for Software Development and CAD, and depending on the use of CAD the hardware requirements are slightly different:

Recommended for the Software and Design Development role with CAD:

- *Windows Performance Laptop*

Recommended for Software Development Only:

- *Windows Standard Laptop*
  - Only cloud CAD solutions are recommended
    * OnShape, SolidWorks 3D Experience, etc.

Also Supported:

- *MacOS Standard Laptop*
  - REV Hardware Client not supported
  - LabVIEW software not supported

It is also recommended to have an active internet connection during software development. Access to https://github.com is required by the REV Hardware Client to download and install required season software and firmware updates. Additional software may have similar requirements.

14.2 Recommended Hardware Sets

These are the Recommended Hardware sets referenced by the Program-Specific Requirements. There are a few extra requirements and recommendations for all hardware platforms, such as:

Windows Operating System

- Support for Windows 10 is ending in mid-2025, so purchasing a Windows system that supports Windows 11 is highly recommended. While not all software is specifically labeled as being supported by Windows 11, virtually all the required software has been tested to work with Windows 11.

USB Ports

- Laptops should have at least 2 available physical USB-A ports.

- For *FIRST* Tech Challenge, USB-C ports on laptops are not able to work properly with the REV Control Hub nor REV Driver Hub, so it is important to have USB-A ports also available.

Bluetooth

- For *FIRST* LEGO League, it is important that laptops and tablets support Bluetooth 4.0 or above.

Physical Ethernet Ports

- While most features of hardware and software can be easily supported by Wi-Fi, in some situations (such as the Driver Station for *FIRST* Robotics Competition) having a physical RJ-45 ethernet port on the system is a huge benefit.

SSD Hard Drive

- While not specifically required, hard drives that use SSD technology (versus spinning disk technology) boot up faster and are less likely to be damaged when carrying while powered on or experiences "unexpected bumps" as is common for a *FIRST* Robotics Competition Driver Station computer.

14.2.1 Windows Performance Laptop

A laptop designed for high graphics performance containing a high-end processor, like a Dell G16 or HP Omen, with the following recommended specs:

- Processor: Intel Core i7, AMD Ryzen 7, or better
- Graphics: NVIDIA GeForce RTX 4050 or better
- Memory: 16GB RAM or more, 32GB preferred
- Storage: 512 GB SSD or greater, 1TB SSD preferred
- Ethernet: RJ-45 Ethernet Port preferred
- Ports: 2 or more USB type A ports preferred
- Bluetooth: Bluetooth 4.0 or better
- Wi-Fi: Integrated Wi-Fi, Wi-Fi 6E or better preferred
- Operating System: Windows 10 or better, Windows 11 preferred

14.2.2 Windows Standard Laptop

A standard Windows laptop, like a Dell Inspiron 15 or HP Pavilion Laptop, designed for smooth performance and everyday tasks,

- Processor: Intel Core i5, AMD Ryzen 5, or better
- Graphics: Intel or AMD embedded graphics adapter or better
- Memory: 8GB RAM or more, 16GB preferred
- Storage: 256GB or greater, 512 GB SSD preferred
- Ethernet: RJ-45 Ethernet Port preferred
- Ports: 2 or more USB type A ports preferred
- Bluetooth: Bluetooth 4.0 or better
- Wi-Fi: Integrated Wi-Fi, Wi-Fi 6E or better preferred
- Operating System: Windows 10 or better, Windows 11 preferred

14.2.3 MacOS Standard Laptop

A standard MacOS laptop, like a MacBook Air or MacBook Pro, designed for smooth performance and everyday tasks.

- Processor: Apple M1 or better, Apple M2 preferred
- Memory: 4GB RAM or more
- Storage: 2GB available storage space or better
- Bluetooth: Bluetooth 4.0 or better
- Operating System: MacOS Mojave 10.14 or newer

14.2.4 iOS Standard Tablet

A standard iOS tablet, such as an iPad Air 2 or iPad Mini 4 or newer.

- Operating System: iOS 13 or newer

14.2.5 Chrome OS Standard Laptop

A standard Chromebook, such as the Samsung Galaxy Chromebook 2, or similar.

- Processor: 1.40 GHz Intel Celeron 2955U dual-core processor or better
- Memory: 4GB RAM or better
- Storage: 3GB available storage space or better
- Bluetooth: Bluetooth 4.0 or above
- Operating System: Android 7.0 or newer

14.2.6 Android Standard Tablet

A standard Android Tablet, such as the Samsung Galaxy Tab A7 Lite, or similar.

- 8” display or larger
- Memory: 3GB RAM or better
- Storage: 3GB available storage space or better
- Bluetooth: Bluetooth 4.0 or above
- Operating System: Android 7.0 or newer

Chapter 15

FIRST *Tech Challenge Software Development Kit*

The Software Development Kit (SDK) is the collection of tools for developing software and executing it on a *FIRST* Tech Challenge robot. SDK Software includes:

- *FIRST* Tech Challenge Driver Station App
  - Includes Self-Inspect, *Robot Configuration*, and others
- *FIRST* Tech Challenge Robot Controller App
  - Includes *Blocks Programming Environment*
  - Includes *OnBot Java Programming Environment*
- Android Studio Project for building the Robot Controller App with *Android Studio*
- Javadoc Reference Documentation
- Season-Specific Assets (TensorFlow models, Vuforia databases, etc…)

All released apps/source can be found in the SDK GitHub Repository.

15.1 SDK Releases

The Software Development Kit is developed and maintained by a core group, known as the *FIRST* Tech Challenge *Technology Team*, within a private GitHub repository. This repository is kept private in order to prevent leaking the details of future *FIRST* Tech Challenge game secrets, features in development, and other aspects of development. Development and maintenance is ongoing year round.

15.1.1 Release Content

Once the SDK is ready to be released, the private SDK repository is built and exported. This build consists of:

- Built Driver Station App (`FtcDriverStation-release.apk`)
- Built Robot Controller App (`FtcRobotController-release.apk`)
- Android Studio Project source code (`vX.X.zip`, `vX.X.tar.gz`)
- Javadoc Reference Documentation
- Season-Specific Assets (TensorFlow models, Vuforia databases, etc… hosted separately)

The export is then pushed to the FtcRobotController GitHub Repository as a software release.

The FtcRobotController GitHub Repository is also updated with the exported Android Studio Project source so that changes can be tracked and the GitHub repository can be forked or cloned by teams. This update is a one-way push, however, which is why public contributions (Pull Requests) to the FtcRobotController repository are not accepted. The community is free and encouraged to create issues at the repository for the *Technology Team* to consider and address, however.

Note: Some season-specific assets, such as TensorFlow models and Vuforia Databases, are not included directly in the FtcRobotController GitHub repository. Instead, they are packaged in an .AAR hosted on Maven Central. When using the Robot Controller App, these assets are included in the app. When using Android Studio, these assets are downloaded and included in your project the first time you compile the project (so an active internet connection is necessary).

15.1.2 Release Schedule

These releases happen on a regular schedule, even if the exact dates aren't specifically defined:

- **Kickoff SDK Release** - Generally released within a week or two of the *FIRST* Tech Challenge Kickoff. The Kickoff SDK is typically the minimal software version required for use during the season.
- **Update / Patch Releases** - These are typically released during the *FIRST* Tech Challenge season, when critical issues or helpful features are available for teams. Update/Patch releases aren't generally required for competition unless a critical patch or bugfix is issued.
- **Offseason Release** - Offseason releases are used to prepare teams for breaking changes or to provide a technology preview for new features in the upcoming season.

Software SDK updates are announced via the FIRST Tech Challenge Blog and Team Email Blasts .

15.2 SDK Release Notes

One of the most important elements of the SDK Release is the SDK Release Notes. The SDK Release Notes contain important aspects of each release, including breaking changes, enhancements, and critical bug fixes of note.

15.2.1 Breaking Changes

Breaking changes are as the name suggests, which are changes made within the SDK's APIs or general architecture that may break existing code or configurations that may already exist. It is especially important for all users of the SDK to read the Breaking Changes section of the release notes, if one exists for a given release, and determine the impact on their existing code.

15.2.2 Enhancements

Enhancements are new features or (non-breaking) improvements made to existing features of the SDK. Enhancements might include items such as improved logging, new user interfaces (UI), better user experience (UX), new APIs, better performance, or greater reliability. Not all enhancements of the SDK are listed in the release notes, but those that have a direct user impact should be listed.

15.2.3 Bug Fixes

Virtually every release of the SDK includes bug fixes, but when the *Technology Team* wishes to elevate the visibility of an important bug fix it is included in a Bug Fixes section of the Release Notes. Sometimes team code can be affected if the bug required a workaround, and being elevated in the Release Notes is a way for the *Technology Team* to notify teams that the workaround is no longer necessary.

15.3 Updating SDK Software

It is important for teams to update the SDK software. Updates mid-season may not be required. Teams can check the minimum software version required for a game in Game Manual 1. It is recommended to use the REV Hardware Client to update hardware, if a 64-bit Windows computer is available. If not, then alternate methods provided can be used to update the software.

- *Updating the REV Hardware Client*
- *Updating the Driver Station App*
- *Updating the Robot Controller App*
- *Updating the Driver Hub OS*
- *Updating the Control Hub OS*
- *Updating the Hub Firmware*

Chapter 16

Updating Components of the Control System

Certain components of the *FIRST* Tech Challenge Control System will periodically receive updates. Teams should make sure to update each component of the Control System to the latest released version.

16.1 Installing and Updating the REV Hardware Client

The REV Hardware Client is a desktop app, or software tool, that simplifies updating software on devices used in *FIRST* Tech Challenge. Unfortunately the REV Hardware Client is currently Windows-only, Apple/Mac users must use alternate methods of updating software. In this tutorial, some steps ask to download software and updates - doing this is not required, but will save time during updates.

To install, use the following steps on a **64-bit** PC or laptop running Windows 7 or newer.

Apple/Mac users should skip these steps.

Tip: Not sure about 64-bit? In Windows Explorer, right-click "Computer" (Win 7) or "This PC" (Win 10), choose Properties, see "System type".

16.1.1 Installing the RHC

1. Connect the computer to the internet, and download RHC from the REV RHC download page. Just click the orange Download button and choose your computer's Downloads folder to store the file.

2. See the downloaded file shown at lower left (green arrow). Click that filename to begin installing the RHC app; then follow the prompts. When that's complete, the RHC icon will appear on your computer's desktop.

   If the computer is **not** 64-bit, RHC installation will fail with an appropriate error message.

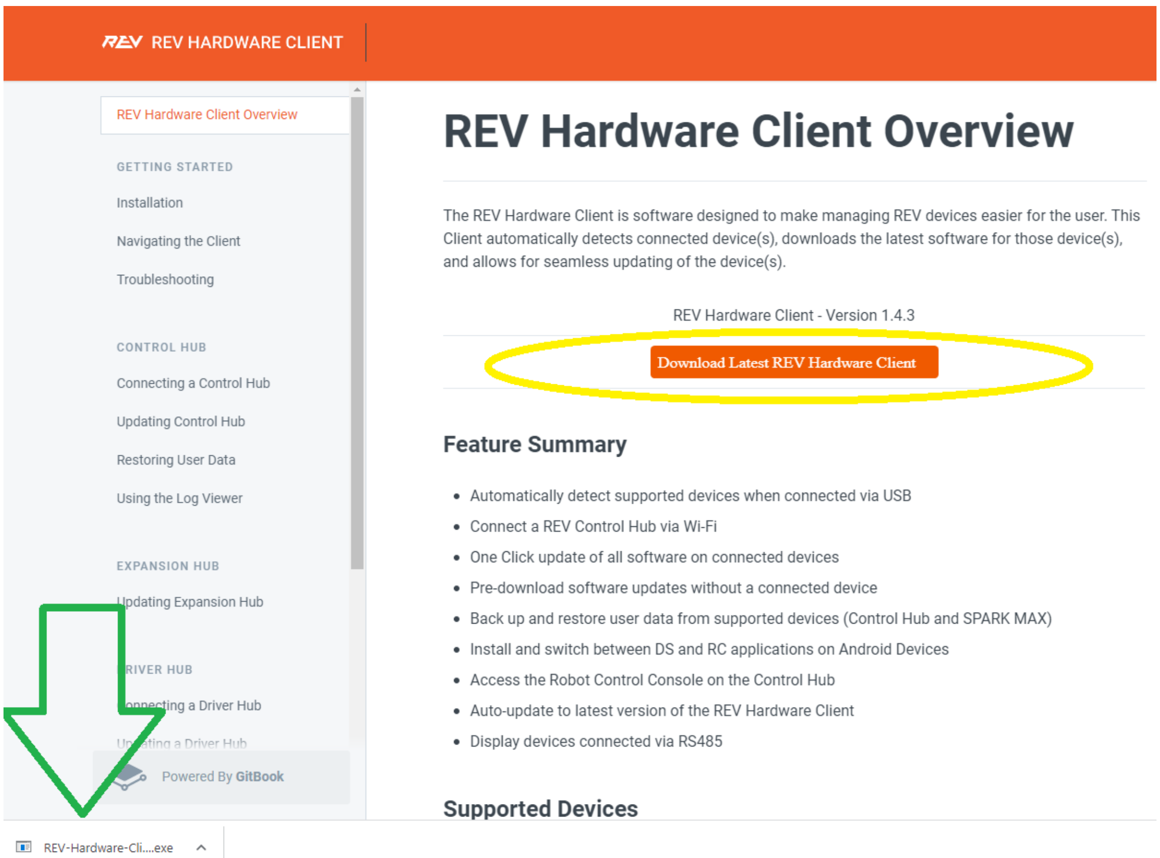


Fig. 1: Downloading REV Hardware Client

16.1.2 Downloading Initial Updates

Open the RHC app. This is a good time to **pre-download** various pieces of software you might need soon.

Why download now? Later, this computer might be connected via Wi-Fi to a Robot Controller, not to the internet. Or a good internet connection might not be available when urgently needed (Murphy's Law).

Click on the Downloads tab (top left). Under "Available Files" is a list of software for *FIRST* Tech Challenge and other software for a different program called *FIRST* Robotics Competition.

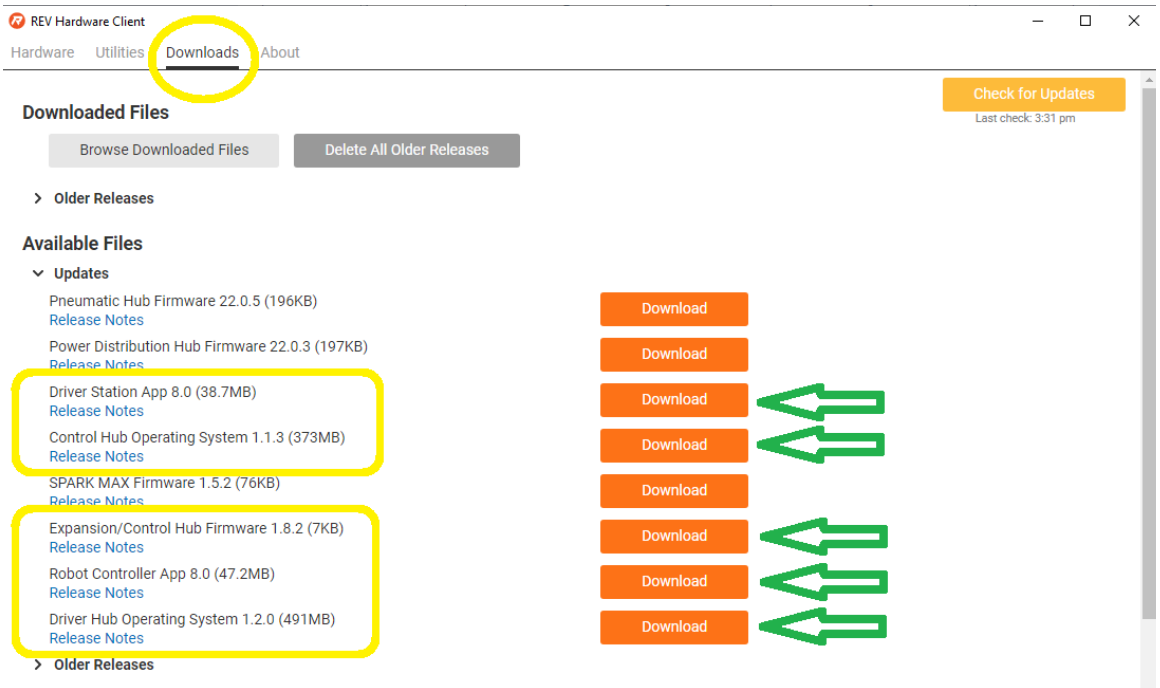


Fig. 2: REV Hardware Client Available Files

Click the orange Download button, only for the 5 *FIRST* Tech Challenge items (yellow rectangles). This may take a few minutes; the OS files are large.

You don't need to track where these files are stored; they will be available to the RHC app when needed for device update.

When complete, these 5 items will appear under the heading "Downloaded Files".

16.1.3 Updating the REV Hardware Client

1. On a Windows computer connected to the internet, open the REV Hardware Client.

2. Click the "About" tab, then click "Check for Updates" (green arrow, above). If a new version is available, click to update.

That's all for now! You will use these files later, when updating various devices. More info about the RHC is at REV Robotics' excellent documentation site.

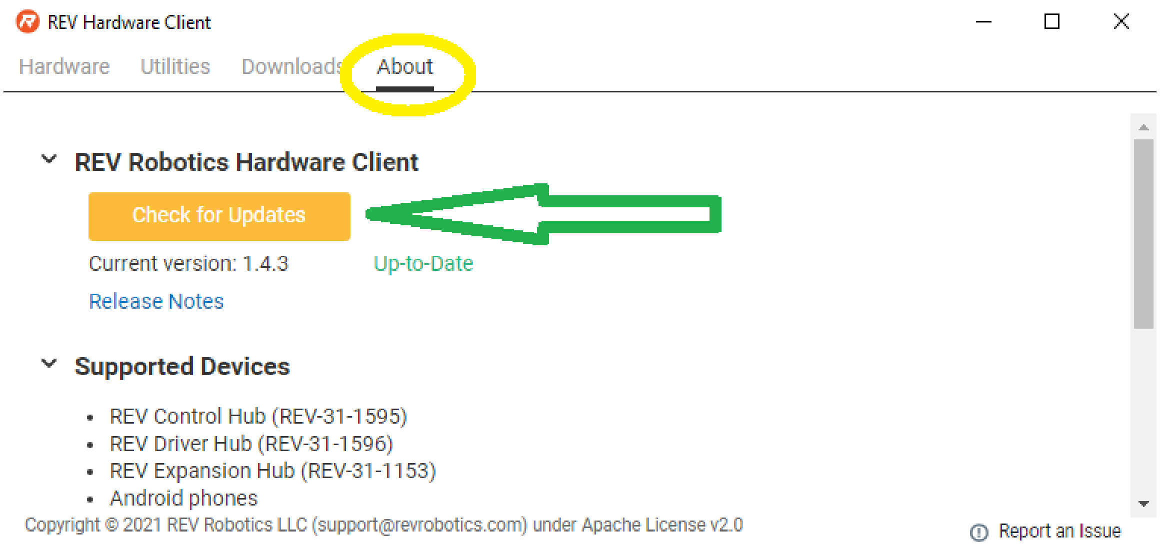


Fig. 3: REV Hardware Client Available Updates

Questions, comments and corrections to westsiderobotics@verizon.net

16.2 Updating the Driver Station App

The Driver Station App is one of the Apps provided with the *FIRST* Tech Challenge *Software Development Kit (SDK)*. The Driver Station App is the major interface for robot configuration, gamepad support, self-inspect, Team code selection and execution, and others. This app runs on the REV Driver Hub or an approved Android smartphone.

This page shows how to update the Driver Station (DS) app on these devices:

- REV Driver Hub
- An approved Android DS phone

These methods for updating the Driver Station App are the same regardless of the programming language/environment used to program robot Team Code.

Updating Driver Station (DS) app on REV Driver Hub

Here are 3 methods to update the DS app on a REV Driver Hub:

1. REV Hardware Client (RHC)
2. “Side loading” with APK
3. Software Manager on REV Driver Hub

Method 1 - REV Hardware Client (RHC) - Windows computers only

Plug the REV Driver Hub directly into the Windows computer with RHC installed and open. Use a USB-C data cable. Make sure the "Hardware" tab is active, at top left. The DS app on the Driver Hub does **not** need to be open.

Here the computer does not need to be connected to the internet, since *in Updating the REV Hardware Client* the required DS update file was previously downloaded.

The RHC app will recognize the Driver Hub, as shown here:

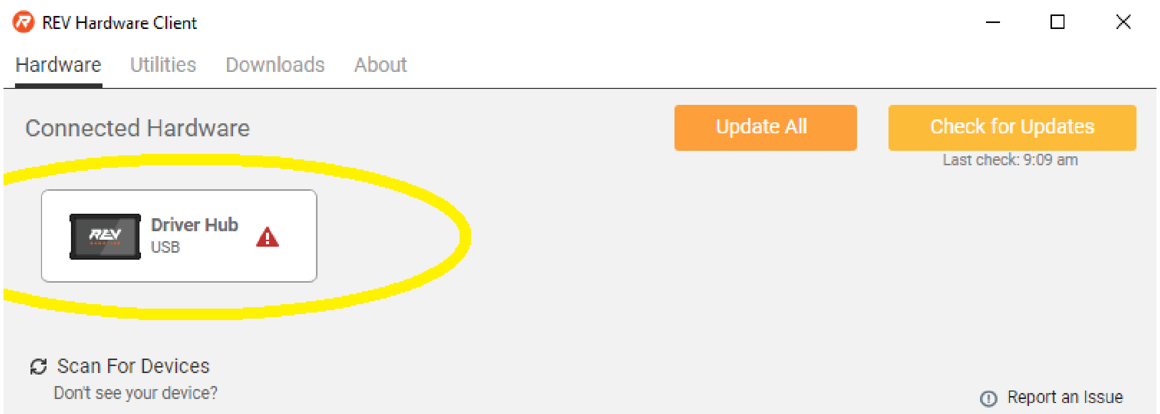


Fig. 4: Recognizing the Driver Hub

Once recognized, click on the Driver Hub's large icon/rectangle. The RHC app now displays the update status of the DS app, if any.

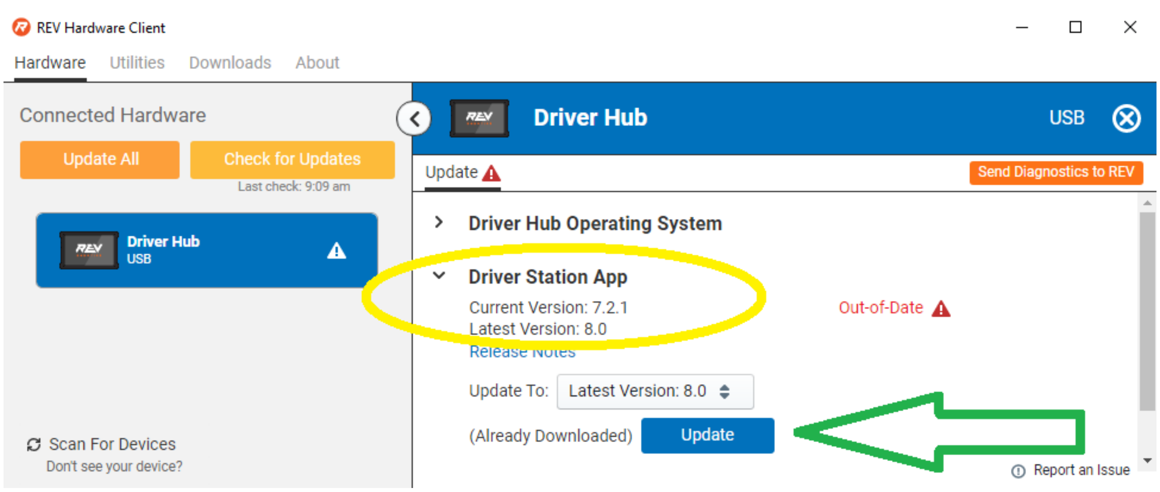


Fig. 5: Updating the Driver Hub

Simply click the blue Update rectangle (green arrow) – done!

The update was fast, because you had already downloaded the DS app to the RHC. That was noted with '(Already Downloaded)", to the left of the blue Update rectangle.

You could have selected an **older** version of the DS app, in the drop-down list just above the blue Update rectangle.

After install, drag the DS app icon from the app menu to the Driver Hub's home screen, if needed.

You may now unplug the Driver Hub from the computer, and close the RHC app. The updated DS app is ready to use.

Method 2 - Side-load APK

Here you will work directly with the Android Package or **APK file** to install the DS app on the Driver Hub. Any computer can be used, PC or Mac, old or new. This method is sometimes called "side-loading".

1. Connect your computer to the internet, open a web browser, and navigate to the SDK github repository.

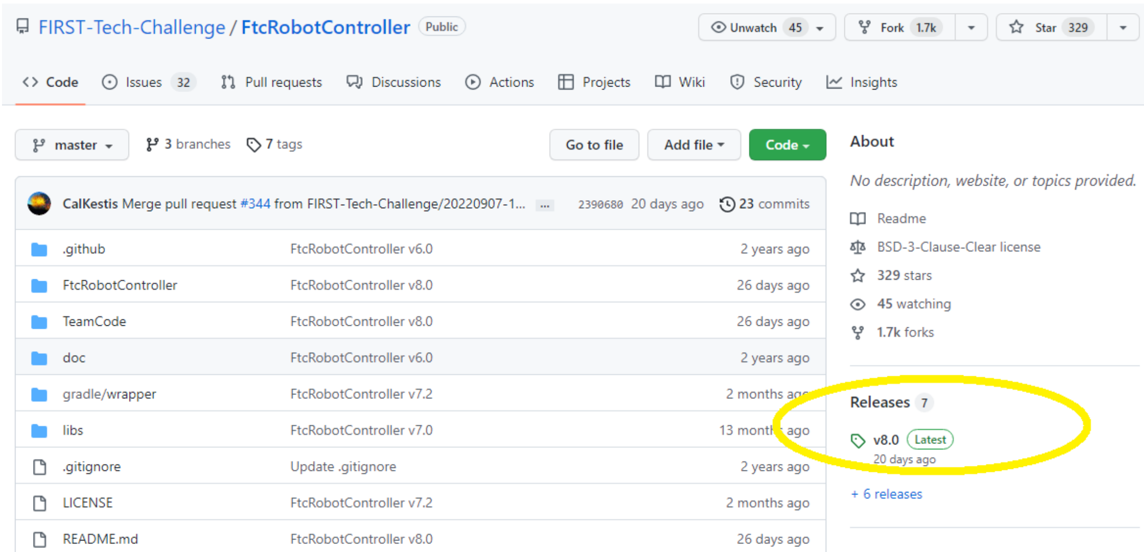


Fig. 6: SDK GitHub Repo

At the right side under "Releases", click the "Latest" icon (yellow oval, above).

In the next page, scroll down slightly in the "Latest" section, to the short list of "Assets". Click on the file "FtcDriverStation-release.apk", to download it to your computer.

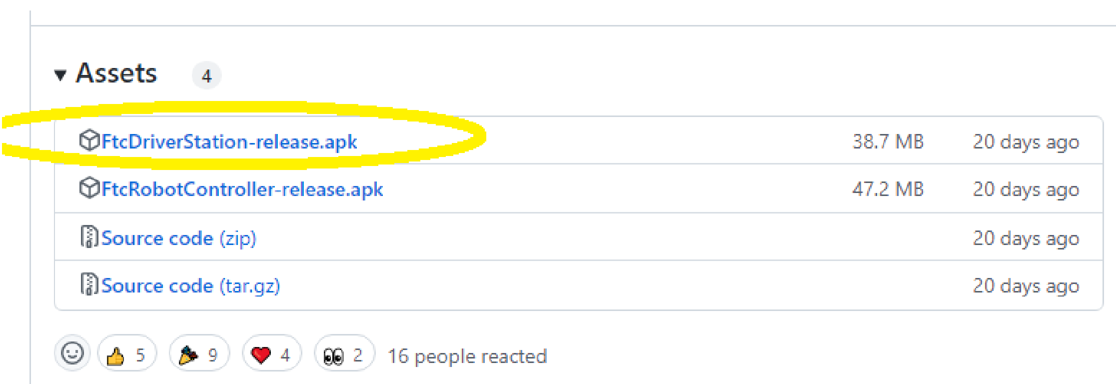


Fig. 7: SDK GitHub Releases

At this time, you could rename the file to reflect its current version number. For example, `FtcDriverStation-release-8.0.apk` or simply `DS-8.0-release.apk`. This distinguishes the file from other versions that might be stored later on that Driver Hub.

2. Transfer the APK file from the computer to the Driver Hub's Downloads folder. Use a USB-C data cable. When complete, you may unplug the Driver Hub from the computer.
3. Uninstall the existing (obsolete) DS app, by dragging its icon to the Trash/Uninstall icon. Or, touch and hold the DS icon for "App info", then choose Uninstall.
4. On the Driver Hub, navigate to the Downloads folder. This can be done in several ways:

- at the main app menu (swipe up), touch the Files icon, then three bars at top left
- use the basic file manager in Settings/Storage, then touch Files
- use a third-party app such as FX File Explorer (from the Google Play Store)

Touch the APK filename that you transferred. Respond to the prompts, to install the updated DS app.

After install, drag the DS app icon from the menu to the Driver Hub's home screen, if needed.

Done! The updated DS app is now ready to use.

Method 3 - Software Manager

The REV Driver Hub has a built-in app called the Software Manager, which can automatically update the DS app (and other related software). It requires only an internet connection.

5. Close all apps, and open the Driver Hub's Wi-Fi menu (in Settings, or swipe down twice from top of home screen). Temporarily connect the Driver Hub to the internet via Wi-Fi.
6. Open the Software Manager app at the Driver Hub home screen (left image, below).

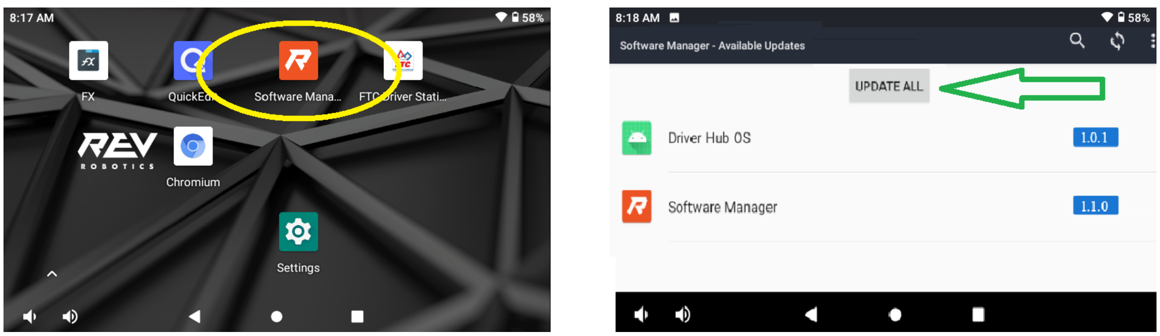


Fig. 8: REV Software Manager

7. The Software Manager will automatically check for any updates needed, and display the results (right image, above). Click the grey box to update the Driver Station (DS) app, if needed.
8. When all is complete, "Forget" the Wi-Fi network used for internet access.

Done! Now the Driver Hub is updated and ready for use.

Updating Driver Station (DS) app on Android smartphone

There are 2 methods to update the DS app on a DS phone:

1. REV Hardware Client (RHC)
2. "Side loading" with APK

Method 1 - REV Hardware Client (RHC) - Windows computers only

Plug the DS phone directly into the computer with RHC installed and open. Use a USB data cable, not a charge-only cable. Make sure the "Hardware" tab is active, at top left. The DS app on the phone does **not** need to be open.

Here the computer does not need to be connected to the internet, since *in Updating the REV Hardware Client* the required DS update file was previously downloaded.

The RHC app will recognize the phone, as shown here:

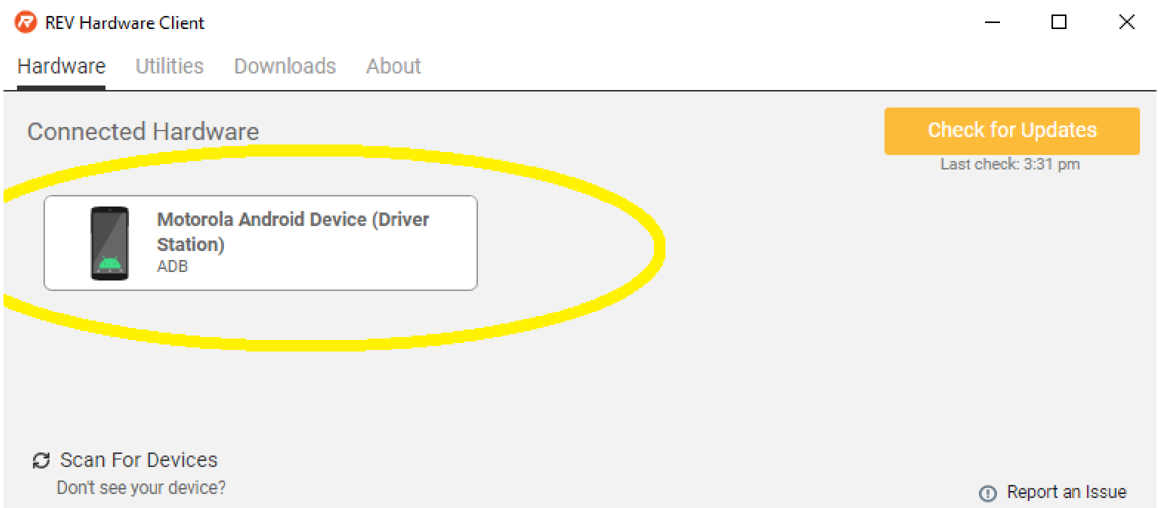


Fig. 9: Recognizing the Phone

If the phone is not recognized, ensure that the phone has *developer options* enabled. If necessary, click the "Scan for Devices" button in the lower-left of the REV Hardware Client app to force the RHC to rescan devices.

Once recognized, click on that phone's large icon/rectangle. The RHC app now displays the update status of the DS app, if any.

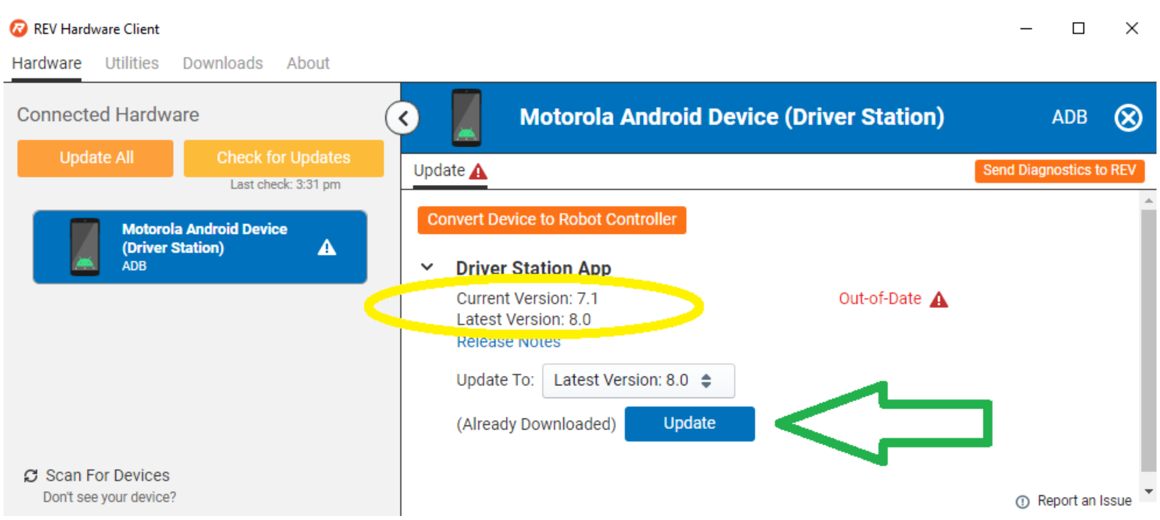


Fig. 10: Update Status of Smartphone

Simply click the blue Update rectangle (green arrow) – done!

The update was fast, because you had already downloaded the DS app to the RHC. That was noted with '(Already Downloaded)", to the left of the blue Update rectangle.

You could have selected an **older** version of the DS app, in the drop-down list just above the blue Update rectangle.

After install, drag the DS app icon from the app menu to the phone's home screen.

You may now unplug the DS phone from the computer, and close the RHC app. The updated DS app is ready to use.

Method 2 - Side-load APK

Here you will work directly with the Android Package or **APK file** to install the DS app on the Android phone. Any computer can be used, PC or Mac, old or new. This method is sometimes called "side-loading".

1. Connect your computer to the internet, open a web browser, and navigate to the SDK github repository.

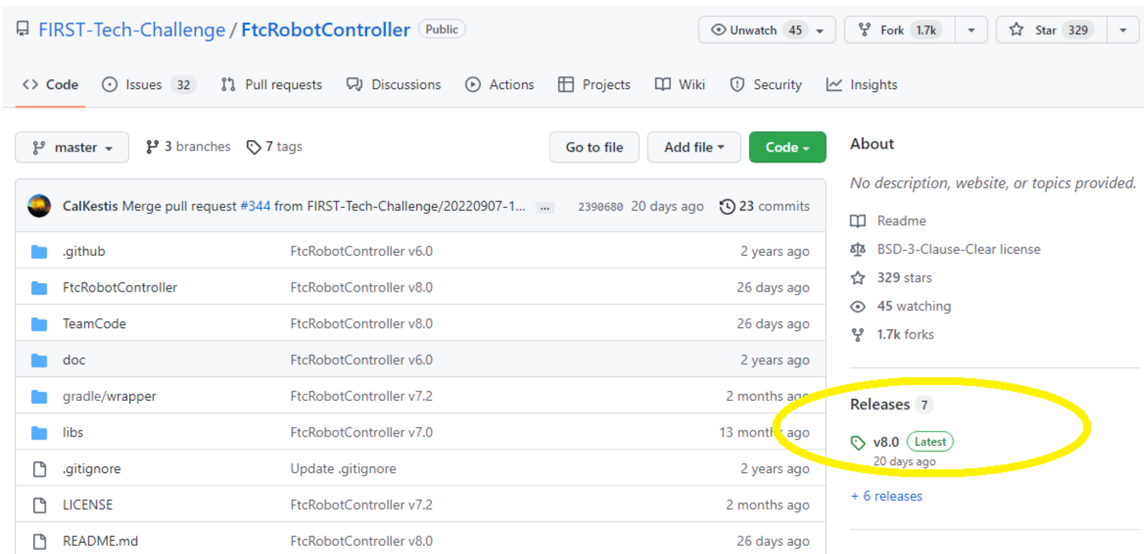


Fig. 11: SDK GitHub Repo

At the right side under "Releases", click the "Latest" icon (yellow oval, above).

In the next page, scroll down slightly in the "Latest" section, to the short list of "Assets". Click on the file "FtcDriverStation-release.apk", to download it to your computer.

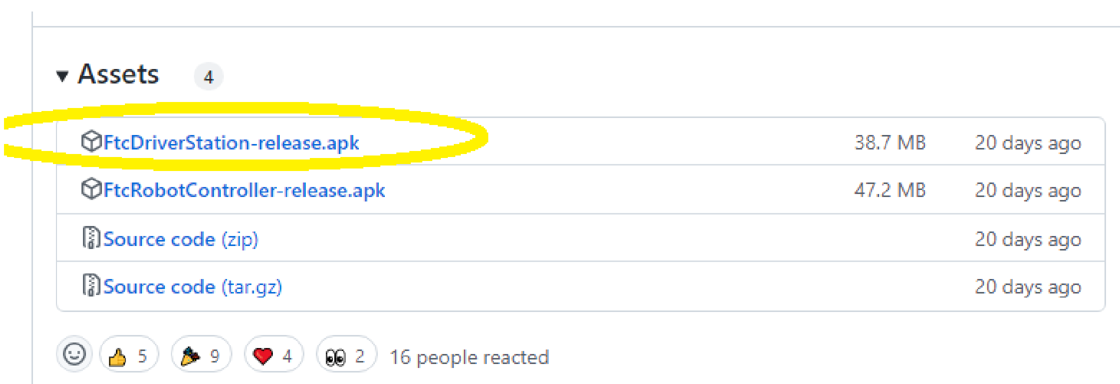


Fig. 12: SDK GitHub Releases

At this time, you could rename the file to reflect its current version number. For example, `FtcDriverStation-release-8.0.apk` or simply `DS-8.0-release.apk`. This distinguishes the file from other versions that might be stored later on that DS phone.

2. Transfer the APK file from the computer to the DS phone's Downloads (or Download) folder. Use a USB data cable (not a charge-only cable). When complete, you may unplug the DS phone from the computer.
3. Uninstall the existing (obsolete) DS app, by dragging its icon to a Trash/Uninstall icon. Or, touch and hold the DS icon for "App info", then choose Uninstall.
4. On the DS phone, navigate to the Downloads folder. This can be done in several ways:
   - at the main app menu (swipe up), touch the Files icon or the Downloads icon (if present)
   - use the basic file manager in Settings/Storage, then Explore or Files
   - use a third-party app such as FX File Explorer (from the Google Play Store)

   Touch the APK filename that you transferred. Respond to the prompts, to install the updated DS app.

   After install, drag the DS app icon from the menu to the phone's home screen.

Done! The updated DS app is now ready to use.

Questions, comments and corrections to westsiderobotics@verizon.net

16.3 Updating the Robot Controller (RC) App

The Robot Controller App is one of the Apps provided with the *FIRST* Tech Challenge *Software Development Kit (SDK)*. The Robot Controller App is the application that runs on the Robot Controller Android Device (REV Control Hub or an approved Android RC phone). This app communicates with the Driver Station App to control the robot.

This page shows how to update the Robot Controller (RC) app on these devices:

- REV Control Hub
- An approved Android RC smartphone

16.3.1 Blocks / OnBot Java vs Android Studio

Blocks / OnBot Java

The Robot Controller (RC) App contains the programming environments for Blocks and OnBot Java, and the User Programs (Team Code) developed using those environments are stored independently ALONGSIDE the RC App. This makes it possible to update the RC App independently without affecting Team Code. This incredibly simplifies updating the RC App software, since no *code* needs to be modified in order to upgrade/downgrade the RC App itself. This does mean, however, that users of Blocks and OnBot Java are limited to the "default" RC App dependencies that are shipped with the App. Blocks and OnBot Java programs can still be run with an Android Studio-built RC App, however, so some flexibility is still possible in this regard for advanced users.

Android Studio

Android Studio, in general, works exactly the opposite. The FtcRobotController repository (the Android Studio Project) contains the full source code needed to build a complete RC App; when the Android Studio Project is compiled and deployed, it's actually building a complete Robot Controller App and installing it onto the RC Android device. Team Code **and** the Robot Controller code are compiled *together*, meaning the Team Code is embedded WITHIN the RC App and cannot be updated/edited independently of the RC App. If the Android Studio-deployed RC App is replaced using the REV Hardware Client or similar process, the RC App with the Team Code embedded is removed and replaced with the default RC App - so Android Studio users should NEVER update the RC App using anything but Android Studio! However, this can complicate upgrading and downgrading software. In order to upgrade/downgrade the RC App portion of their Android Studio code, the team's *Android Studio project* must be merged properly with software releases of the FtcRobotController repository code aligning to the version of the Robot Controller App that they want to use. This must be carefully weighed when deciding to use Android Studio.

16.3.2 Updating the RC App for Android Studio

Android Studio users should **not** use the steps on this page to update the RC app, for the reasons described above. **Android Studio** users must ensure that their Android Studio Projects are up to date with the desired version of the SDK GitHub Repo.

For information on how to properly create and maintain an Android Studio project that can be updated using GitHub, please see *Using GitHub Fork and Clone*

16.3.3 Updating the RC App for Blocks / OnBot Java

These instructions are for independently updating the RC App in situations where such an independent update is supported - i.e. Blocks or OnBot Java development. Expand the following instructions that apply to your Robot Controller hardware:

Robot Controller (RC) app on REV Control Hub

Here are 3 methods to update the RC app on a REV Control Hub:

1. REV Hardware Client (RHC)
2. Manage page on computer
3. Manage page on DS phone or Driver Hub

Note: "Side loading", while possible, is not described here for the Control Hub as it requires a cumbersome procedure with extra equipment.

Method 1 - REV Hardware Client - Windows computers only

Use a USB data cable to connect the REV Control Hub's USB-C port to the Windows computer. Make sure the "Hardware" tab on the RHC is active, at top left.

Here the computer does not need to be connected to the internet, since *in Updating the REV Hardware Client* the required DS update file was previously downloaded.

The RHC app will recognize the Control Hub, as shown here:

Once recognized, click on the Control Hub's large icon/rectangle. The RHC app now displays the update status of the RC app, if any.

Simply click the blue Update rectangle (green arrow) – done!

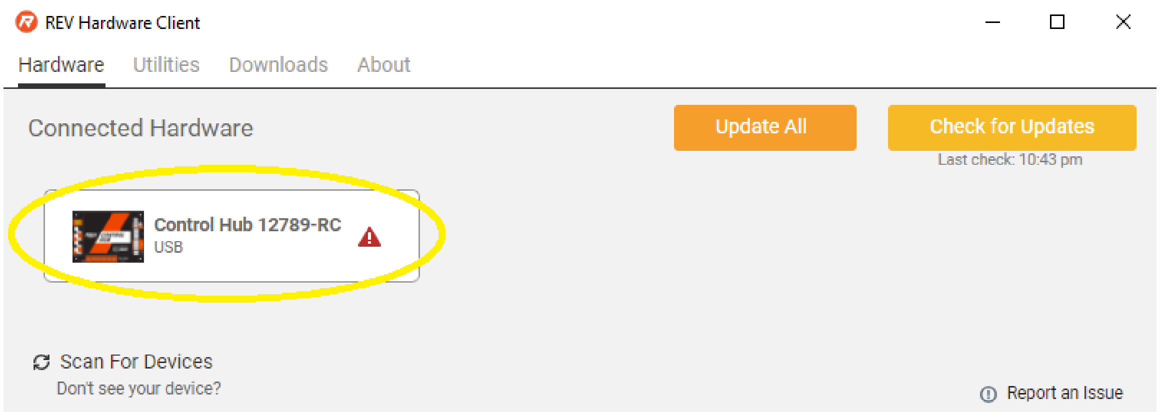


Fig. 13: Recognizing the Control Hub

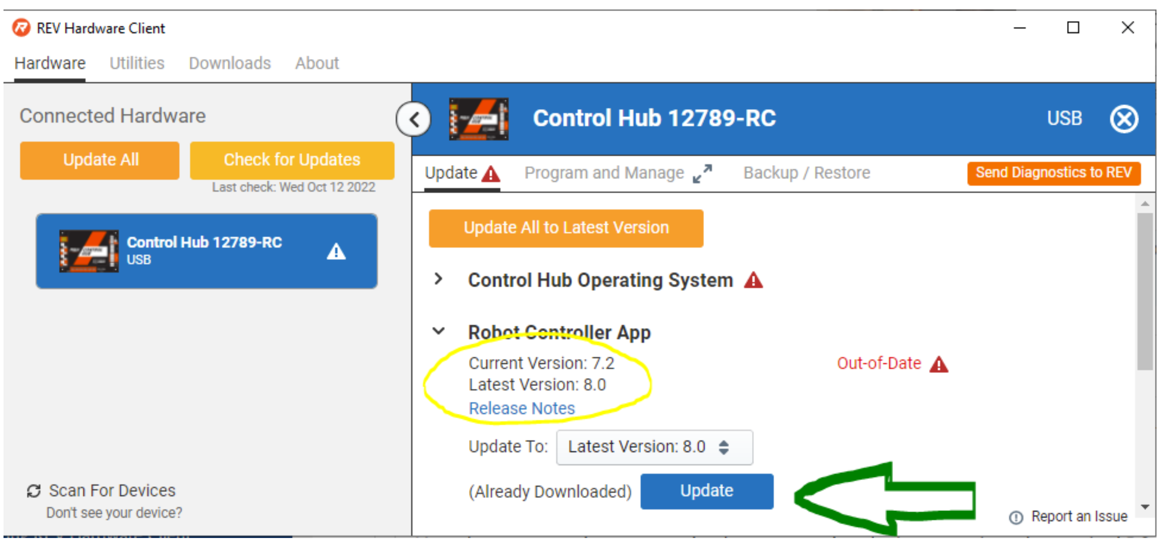


Fig. 14: Updating the Control Hub

Method 2 - Manage page on computer

1. Connect a laptop to the internet, open a web browser, and navigate to the SDK github repository.

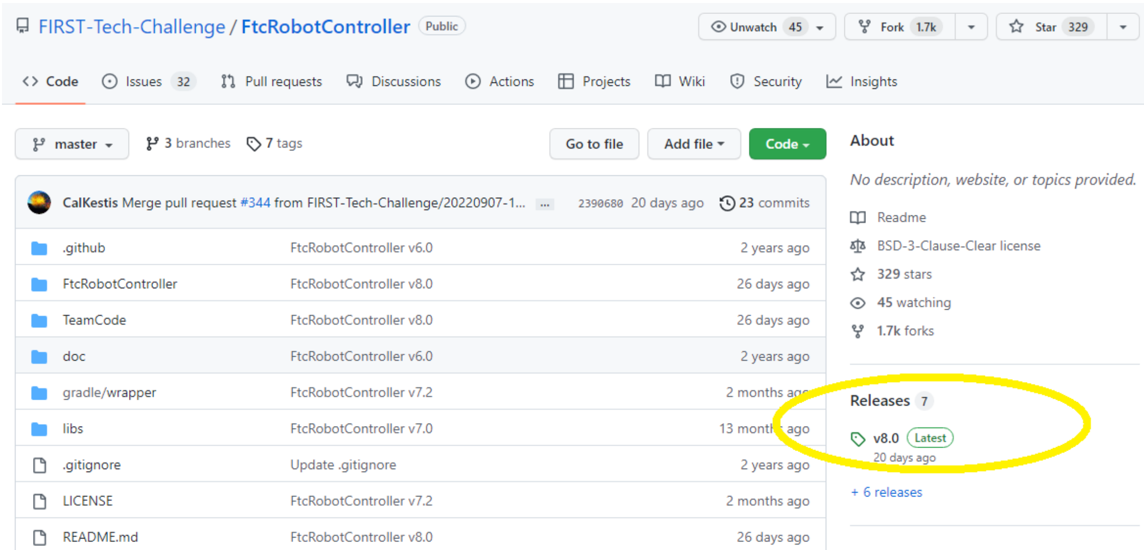


Fig. 15: SDK GitHub Repo

At the right side under "Releases", click the "Latest" icon (yellow oval, above).

In the next page, scroll down slightly in the "Latest" section, to the short list of "Assets". Click on the file "FtcRobotController-release.apk", to download it to your computer.

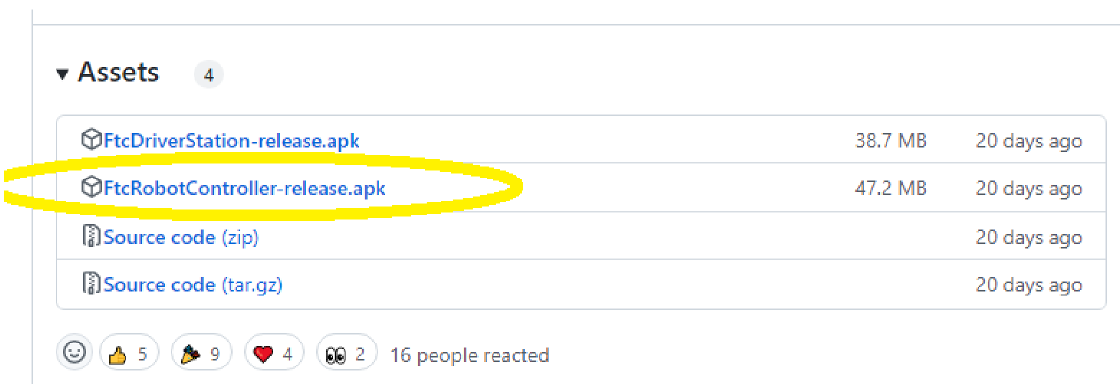


Fig. 16: SDK GitHub Releases

At this time, you could rename the file to reflect its current version number. For example, `FtcRobotController-release-8.0.apk` or simply `RC-8.0-release.apk`. This distinguishes the file from other versions that might be stored later on that RC phone.

2. Turn on the Control Hub (apply robot power), wait for green LED.

3. Connect the same laptop via Wi-Fi to the Control Hub. Open the Chrome browser, enter the usual address `http://192.168.43.1:8080`

   Click the Manage tab, then scroll down to "Update Robot Controller App".

   Click "Select App…". Navigate to the laptop folder where the RC APK file is stored, and select that file.

   Now click the "Update" button (green arrow, above).

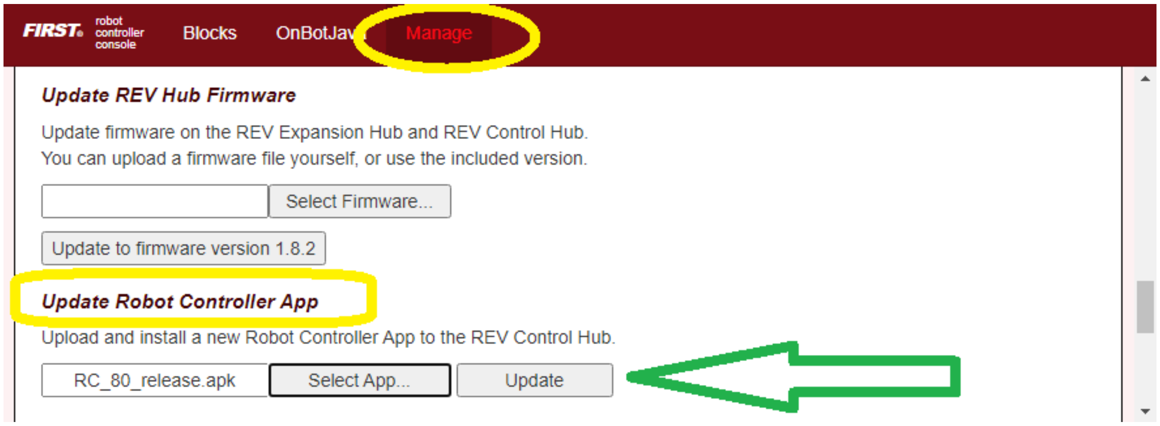


Fig. 17: Update RC App

The software will replace the existing RC app with your new updated RC app. The connection between laptop and Control Hub will temporarily be lost, then automatically restored.

When the completion message appears, the updated RC app is ready to use.

Method 3 - Manage page on Driver Hub or DS phone

This method can be used if your computer is unavailable or unable to connect via Wi-Fi to the Control Hub. For example, your desktop computer might have only a wired (Ethernet) network port, lacking Wi-Fi.

But this method does require the RC APK file to be stored in the Download (or Downloads) folder on the Driver Hub or DS phone. That's correct: **Robot Controller APK** stored on the **Driver Station** device.

First download the RC APK file from the github repo to the computer, as shown in Step 1 of Method 2. Then transfer that APK file from the computer to the DS device's Download folder, using a USB data cable. When complete, you may unplug the DS device from the computer.

Connect the DS app to the Control Hub, from the DS app's Settings menu (never with the Android device Wi-Fi settings).

From the DS app's menu, select "Program and Manage". Then touch the 3 bars at top right, and select "Manage".

This is the same Manage page that appears in a laptop browser. So the following instructions are the same as Method 2 above.

Scroll down to "Update Robot Controller App".

Touch "Select App…". Navigate to the DS device's Download folder, and select the latest RC APK file.

Now touch the "Update" button (green arrow, above).

The software will replace the existing RC app with your new updated RC app. The connection between Driver Station and Control Hub will temporarily be lost, then automatically restored.

When the completion message appears, the updated RC app is ready to use.

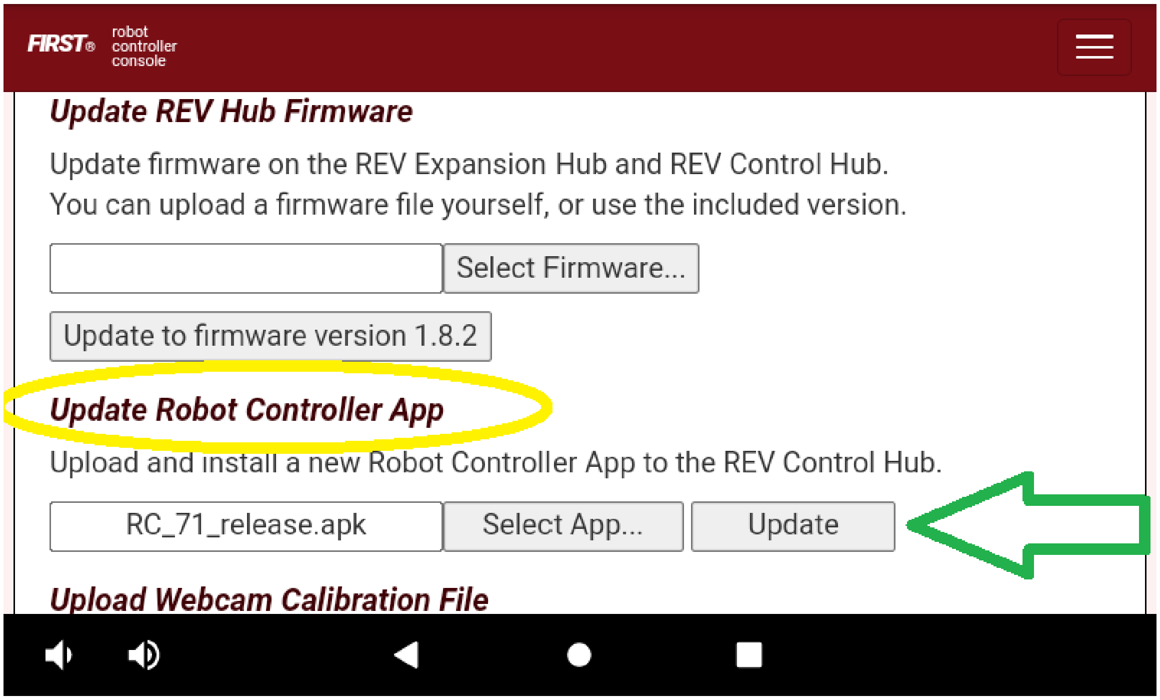


Fig. 18: Update RC App

Robot Controller (RC) app on Android phone

Here are 2 methods to update the RC app on a Robot Controller (RC) phone:

1. REV Hardware Client (RHC)
2. "Side loading" with APK

Note: The Manage page, under Program and Manage, on a computer or Driver Station device, **does not** offer updating an RC app on a connected Robot Controller phone.

Method 1 - REV Hardware Client - Windows computers only

Plug the RC phone directly into the computer with RHC installed and open. Use a USB data cable, not a charge-only cable. Make sure the "Hardware" tab is active, at top left. The RC app on the phone does **not** need to be open.

Here the computer does not need to be connected to the internet, since *in Updating the REV Hardware Client* the required DS update file was previously downloaded.

The RHC app will recognize the phone, as shown here:

If the phone is not recognized, ensure that the phone has *developer options* enabled. If necessary, click the "Scan for Devices" button in the lower-left of the REV Hardware Client app to force the RHC to rescan devices.

Once recognized, click on that phone's large icon/rectangle. The RHC app now displays the update status of the DS app, if any.

Simply click the blue Update rectangle (green arrow) – done!

The update was fast, because you had already downloaded the RC app to the RHC. That was noted with '(Already Downloaded)", to the left of the blue Update rectangle.

You could have selected an **older** version of the RC app, in the drop-down list just above the blue Update rectangle.

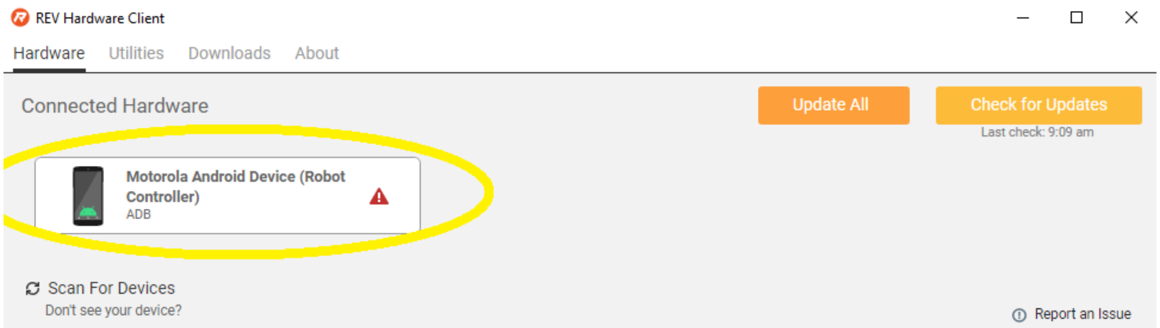


Fig. 19: Recognizing the Phone

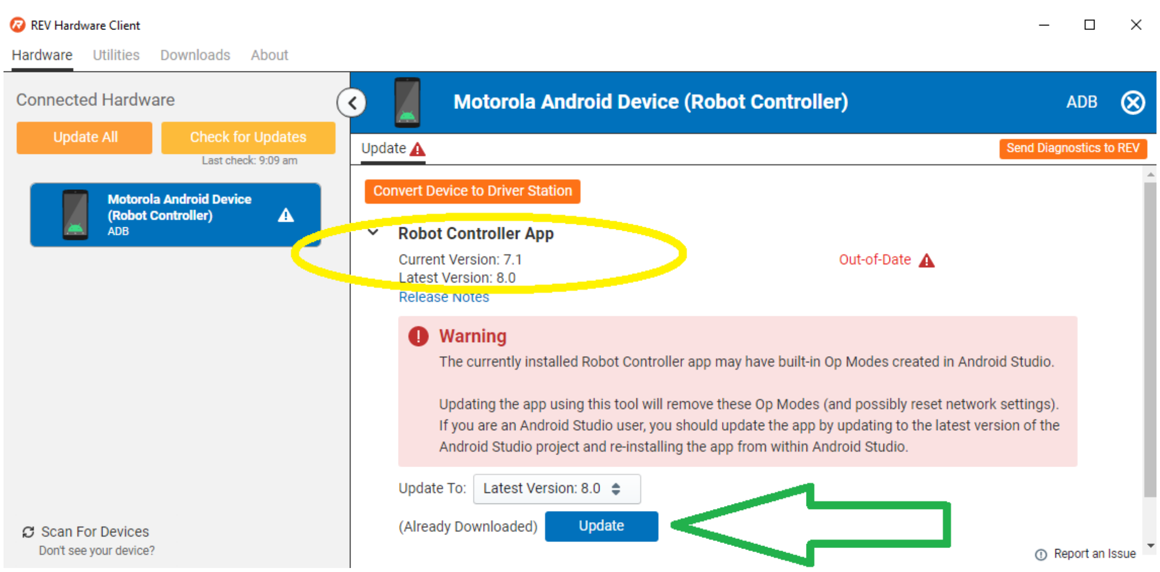


Fig. 20: Update Status of Phone

After install, drag the RC app icon from the menu to the phone's home screen.

You may now unplug the RC phone from the computer, and close the RHC app. The updated RC app is ready to use.

Method 2 - Side-load

Here you will work directly with the Android Package or **APK file** to install the RC app on the Android phone. Any computer can be used, PC or Mac, old or new. This method is sometimes called "side-loading".

1. Connect your computer to the internet, open a web browser, and navigate to the SDK github repository.

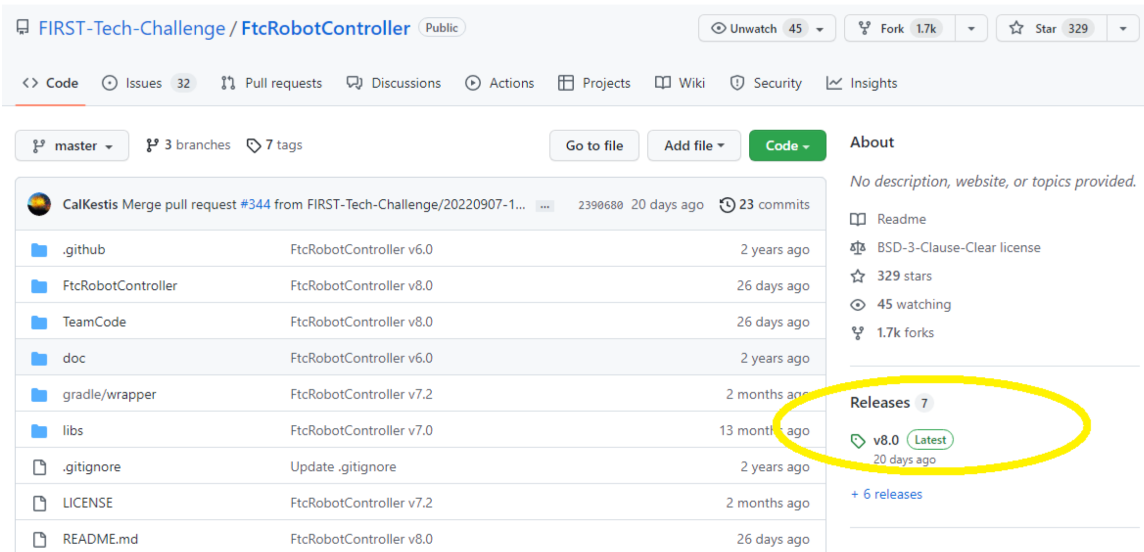


Fig. 21: SDK GitHub Repo

At the right side under "Releases", click the "Latest" icon (yellow oval, above).

In the next page, scroll down slightly in the "Latest" section, to the short list of "Assets". Click on the file "FtcRobotController-release.apk", to download it to your computer.

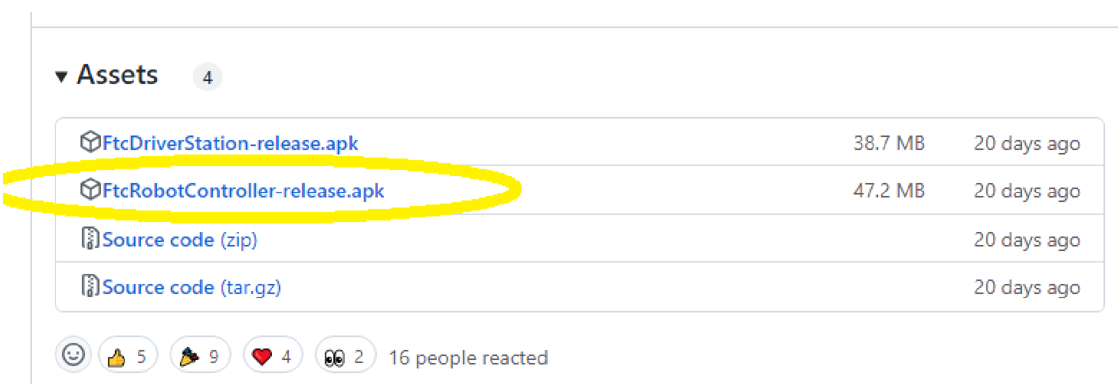


Fig. 22: SDK GitHub Releases

At this time, you could rename the file to reflect its current version number. For example, `FtcRobotController-release-8.0.apk` or simply `RC-8.0-release.apk`. This distinguishes the file from other versions that might be stored later on that RC phone.

2. Transfer the APK file from the computer to the RC phone's Downloads (or Download) folder. Use a USB data cable (not a charge-only cable). When complete, you may unplug the RC phone from the computer.

3. Uninstall the existing (obsolete) RC app, by dragging its icon to a Trash/Uninstall icon. Or, touch and hold the RC icon for "App info", then choose Uninstall.
4. On the RC phone, navigate to the Downloads folder. This can be done in several ways:
   - at the main app menu (swipe up), touch the Files icon or the Downloads icon (if present)
   - use the basic file manager in Settings/Storage: touch Explore or Files
   - use a third-party app such as FX File Explorer (from the Google Play Store)

   Touch the APK filename that you transferred. Respond to the prompts, to install the updated RC app.

   After install, drag the RC app icon from the app menu to the RC phone's home screen.

Done! The updated RC app is now ready to use.

Other descriptions of updating the RC app are at REV Robotics' excellent documentation site.

Questions, comments and corrections to westsiderobotics@verizon.net

16.4 Updating the Driver Hub OS

An Operating System (OS) is software that supports a computer's basic functions, such as scheduling tasks, executing applications, and controlling peripherals. This must sometimes be updated on the **REV Driver Hub**. While this OS update is not specifically part of the *Software Development Kit (SDK)*, the SDK requires these updates for the Driver Hub in order to perform correctly.

Here are two methods for updating the Driver Hub OS:

1. REV Hardware Client (RHC)
2. Software Manager on Driver Hub

More info about updating the Driver Hub OS is at REV Robotics' excellent documentation site.

Method 1 - REV Hardware Client (RHC) - Windows computers only

1. Turn on the Driver Hub. Plug it directly into a computer running the REV Hardware Client, with a USB-C data cable.
2. Click the Driver Hub's large icon/rectangle. Under "Driver Hub Operating System", see the current/latest mismatch, if any (yellow oval, below).

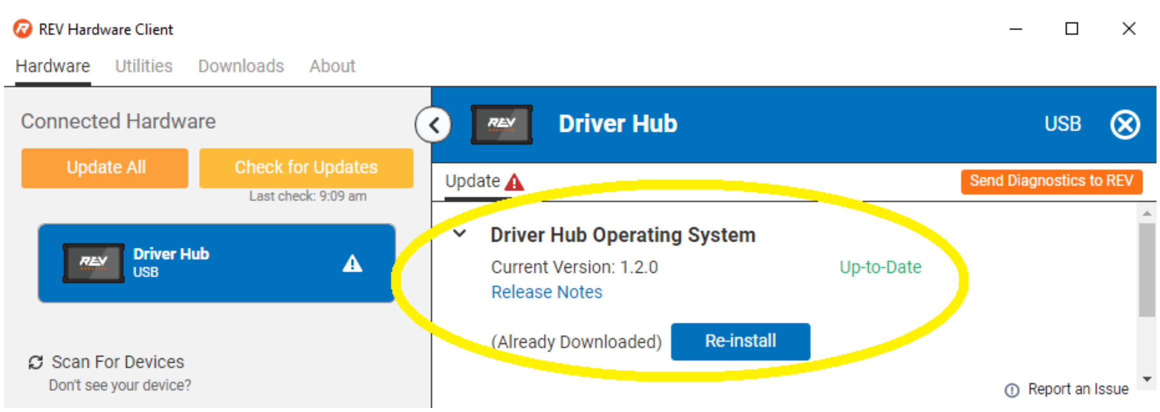


Fig. 23: Updating the Driver Hub OS

Confirm the Latest Version in the drop-down menu, if any. Then click the blue rectangle, labeled "Update" when applicable. The speed of this update is improved, since *in Updating the REV Hardware Client* the required update file was previously downloaded.

Done! The Driver Hub's OS is now updated.

Method 2 - Software Manager

The REV Driver Hub has a built-in app called the Software Manager, which can automatically update the Driver Hub OS (and other related software). It requires only an internet connection.

1. Close all apps, and open the Driver Hub's Wi-Fi menu (in Settings, or swipe down twice from top of home screen). Temporarily connect the Driver Hub to the internet via Wi-Fi.
2. Open the Software Manager app at the Driver Hub home screen (left image, below).

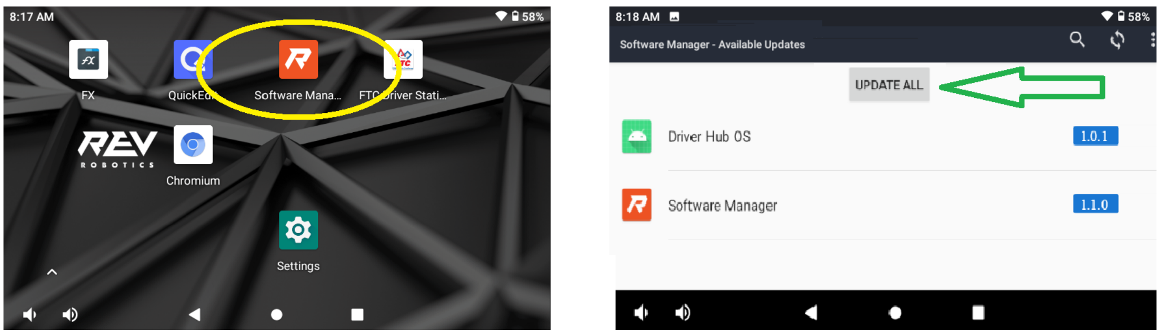


Fig. 24: Updating the Software Manager

3. The Software Manager will automatically check for any updates needed, and display the results (right image, above). Touch the grey button to perform the updates, including the Driver Hub Operating System (OS) if needed.

Note: While REV Robotics does provide a downloadable OS image file for the Driver Hub, the tools available in this tutorial do not accept providing this file for updating the OS.

---

4. When all is complete, "Forget" the Wi-Fi network used for internet access. Now the Driver Hub is ready for regular competition use.

Questions, comments and corrections to westsiderobotics@verizon.net

16.5 Updating the Control Hub OS

An Operating System (OS) is software that supports a computer's basic functions, such as scheduling tasks, executing applications, and controlling peripherals. This must sometimes be updated on the **REV Control Hub**. While this OS update is not specifically part of the *Software Development Kit (SDK)*, the SDK requires these updates for the Control Hub in order to perform correctly.

Here are two methods for updating the Control Hub OS:

1. REV Hardware Client (RHC)
2. Manage page on computer

More info about updating the Control Hub OS is at REV Robotics' excellent documentation site.

Method 1 - REV Hardware Client (RHC) - Windows computers only

1. Apply 12V robot power to the REV Control Hub.
2. Plug the Control Hub directly into a computer running the REV Hardware Client, with a USB-C data cable.
3. Click the hub's large icon/rectangle. Under "Control Hub Operating System", see the current/latest mismatch, if any (yellow oval, below).

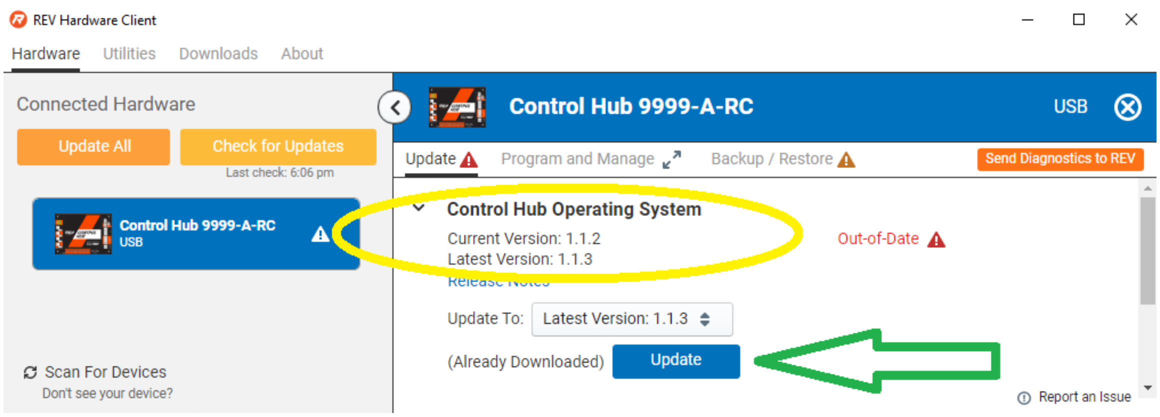


Fig. 25: Updating the Control Hub OS

Confirm the Latest Version in the drop-down menu, then click the blue "Update" rectangle (green arrow, above). The speed of this update is improved, since *in Updating the REV Hardware Client* the required update file was previously downloaded.

Done! The Control Hub's OS is now updated.

Method 2 - Manage page on computer

1. Connect the computer via Wi-Fi to the Control Hub. In the Chrome browser, open the *FIRST* Tech Challenge interface.
2. Click on the Manage tab, scroll down to Update Control Hub Operating System.

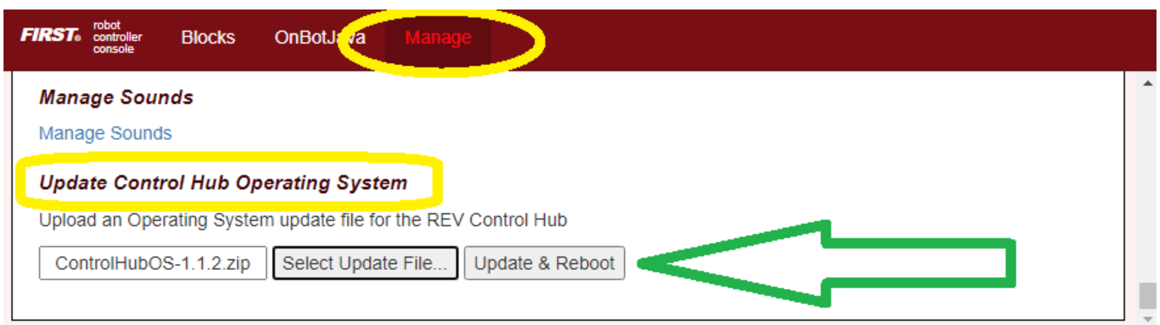


Fig. 26: Updating the Control Hub OS

3. If needed, download the latest OS file from the REV Robotics Control Hub OS web page. Do not extract or "un-zip" this file.
4. At the Manage page, click "Select Update File…" and navigate to the computer's folder where you downloaded the OS file.
5. Select that file, and click "Update & Reboot" (green arrow, above).

That's it! The Control Hub's OS is now updated.

Questions, comments and corrections to westsiderobotics@verizon.net

16.6 Updating Hub Firmware

Firmware is low-level software that controls a device's circuit boards, or electronic **hardware**. This must sometimes be updated on the REV Expansion Hub and the REV Control Hub in order for the *Software Development Kit (SDK)* to perform correctly.

Here are 5 methods:

1. REV Hardware Client (RHC)
2. Driver Station app
3. Robot Controller (RC) app - on RC phone
4. Manage page on computer
5. Manage page on Driver Station device (DS phone or Driver Hub)

Method 1 - REV Hardware Client (RHC) - Windows computers only

1. For REV Control Hub, apply 12V robot power. For REV Expansion Hub, 12V power is optional.
2. Plug the REV Hub directly into a computer running the REV Hardware Client, with a USB data cable (not charge-only). The Expansion Hub's port is Mini USB (not micro). On the Control Hub, use only the USB-C port, not its Mini USB port.
3. Click the hub's large icon/rectangle. Under "Expansion/Control Hub Firmware", see the current/latest mismatch, if any (yellow oval, below).

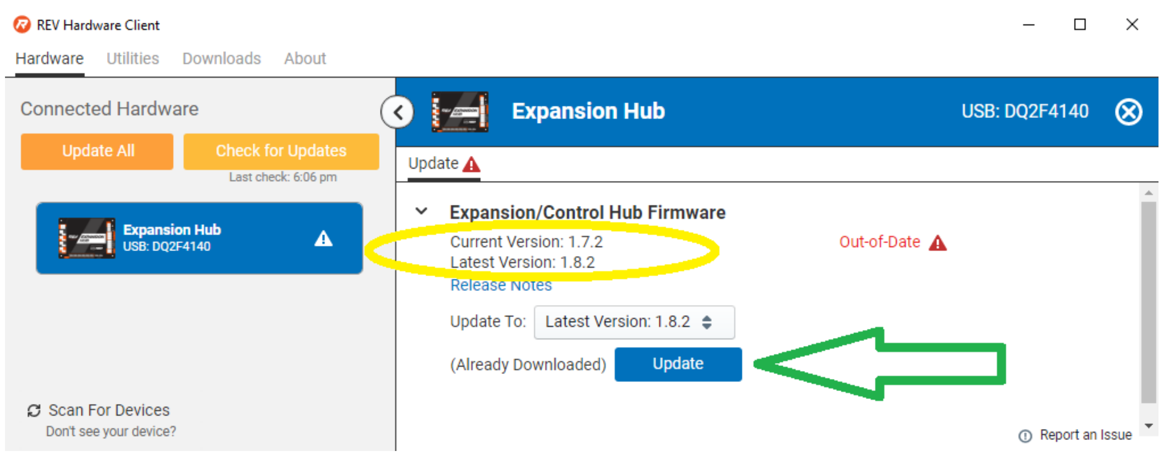


Fig. 27: Updating Firmware

Here's an example with Control Hub:

Confirm the Latest Version in the drop-down menu, then click the blue "Re-install" rectangle (green arrow, above). This is done quickly, since *in Updating the REV Hardware Client* the required update file was previously downloaded.

Done! The Hub's firmware is now updated.

More info about using the RHC to update Hub firmware is at REV Robotics' excellent documentation site.

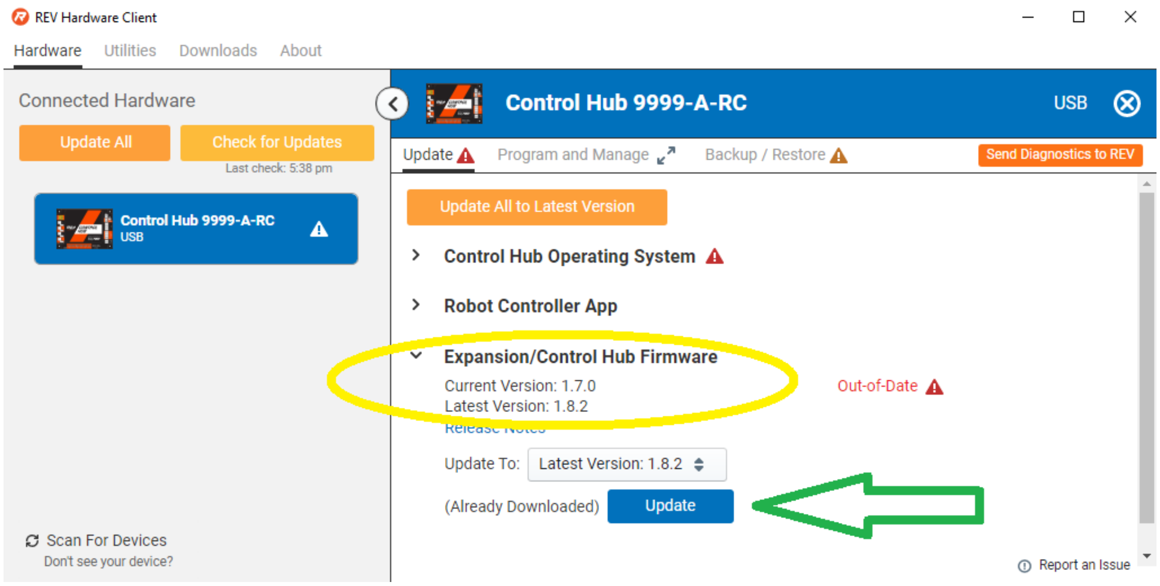


Fig. 28: Updating Firmware

Method 2 - Driver Station app

This method applies to any DS app, running on a DS phone or a Driver Hub.

1. For REV Control Hub, apply 12V robot power. For REV Expansion Hub, connect directly to Robot Controller (RC) phone, open RC app, **and** apply 12V power. The Expansion Hub being updated must be **plugged directly** into the RC phone, with no intermediate Control Hub or other (primary) Expansion Hub. After updating you can return that Hub to its secondary position, if needed.
2. Connect/pair the DS app to the RC device, from a DS phone or Driver Hub. Select DS Settings, Advanced (Robot Controller) Settings, REV Hub Firmware Update.

   Review the list of available Hub firmware, whether stored on the RC device and/or "bundled" in the app.
3. If the latest does **not** appear on the list, you can transfer the firmware file from a computer to the Robot Controller. Use a USB data cable (not a charge-only cable) to store the firmware file in the RC device's subfolder called FIRST/updates/Expansion Hub Firmware.

   Current and older firmware files can be found at the REV Robotics website.

   Then return to this list of available firmware.
4. Now select the latest firmware version and touch "Update Hub Firmware" (green arrow, above). Wait for the process to finish; do not unplug the Hub or restart the robot.

That's it! The Hub's firmware is now updated.

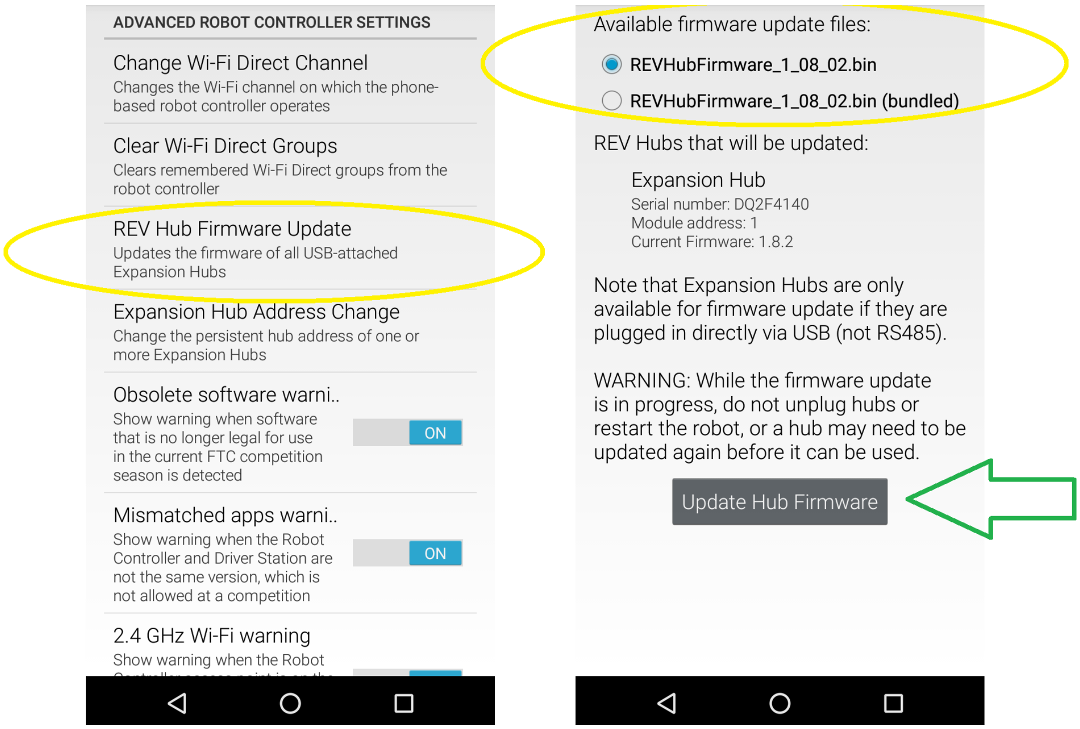


Fig. 29: Updating Firmware

Method 3 - Robot Controller (RC) app - on RC phone

This method is **exactly the same** as Method #2 immediately above, since the DS app was simply providing a portal or window to the RC app.

It's listed separately here, because it applies only to **Expansion Hub**, not Control Hub – which doesn't use an RC phone. In other words, users do not normally interface directly with the RC app on a Control Hub.

Again, the Expansion Hub must be plugged **directly** into an RC phone, with no intermediate (primary) Expansion Hub. After updating you can return that Hub to its secondary position, if needed.

Method 4 - Manage page on computer

1. Connect the computer via Wi-Fi to the Control Hub or RC phone. In the Chrome browser, open the Manage interface.
2. Click on the Manage tab, scroll down to Update REV Hub Firmware.

   See if the grey box (see green arrow, above) offers the latest firmware version, included or bundled with the RC app.
3. If not, click the "Select Firmware…" box. Navigate to the desired firmware file stored on the computer, and select it.

   As part of the update process, that selected firmware file will be stored on the Control Hub or RC phone, in a subfolder called FIRST/updates/Expansion Hub Firmware.

   Current and older firmware files can be found at the REV Robotics website.
4. Now click the box called "Update to…" or "Update using…" (see green arrow, above).
5. At the confirmation prompt, click the blue box "Update Hub Firmware". Wait for the process to finish; do not unplug the Hub or restart the robot.

That's it! The Hub's firmware is now updated.

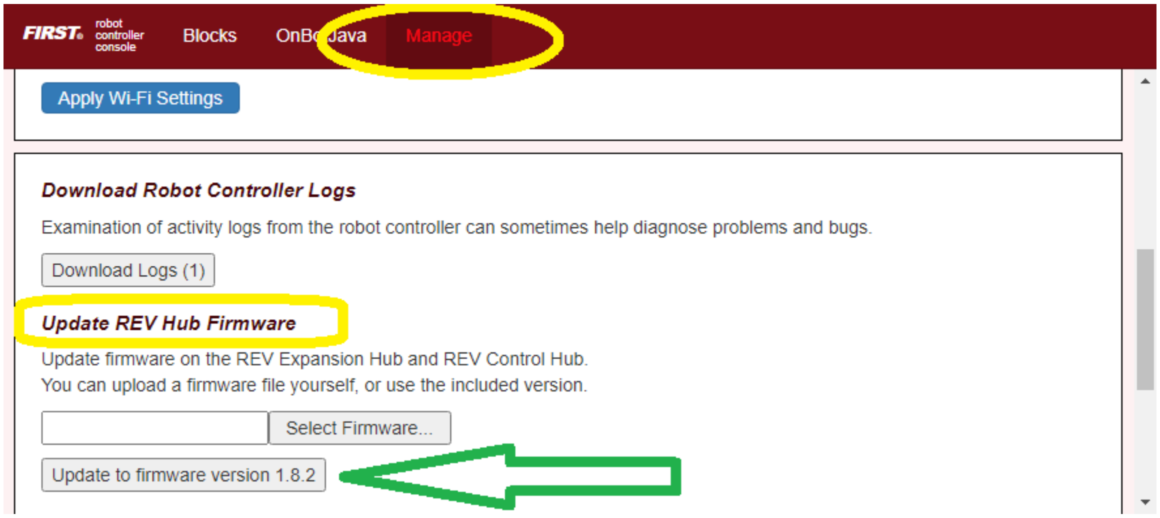


Fig. 30: Updating Firmware

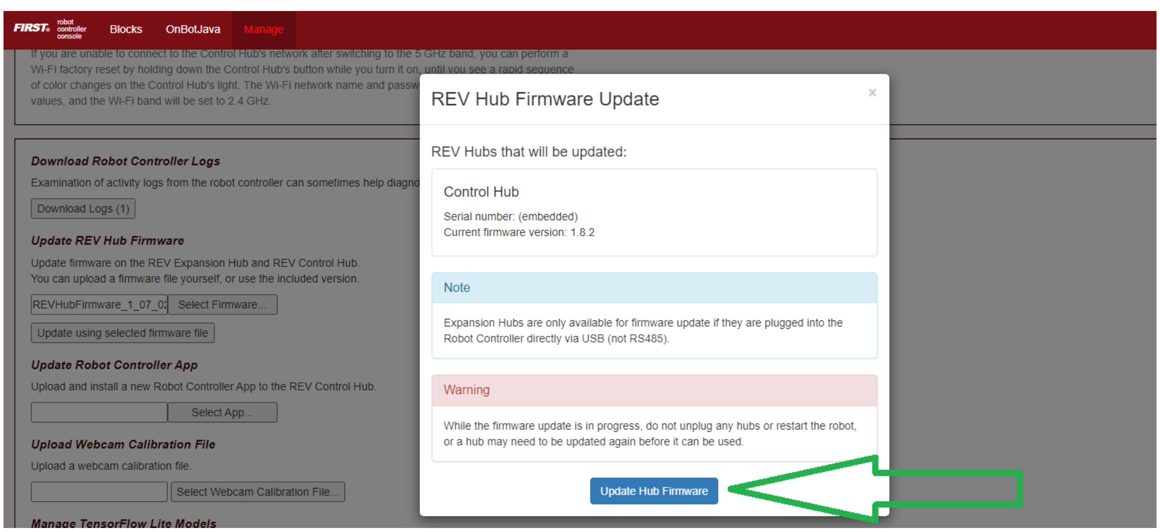


Fig. 31: Managing Firmware

Method 5 - Manage page on Driver Station device - DS phone or Driver Hub

1. Connect the DS app to the Control Hub or RC phone, from the DS app's Settings menu (never with the Android device Wi-Fi settings).
2. From the DS app's menu, select "Program and Manage". Then touch the 3 bars at top right, and select "Manage".

   This is the same Manage page that appears in a laptop browser. So the following instructions are similar to Method 4 above.
3. Scroll down to Update REV Hub Firmware.

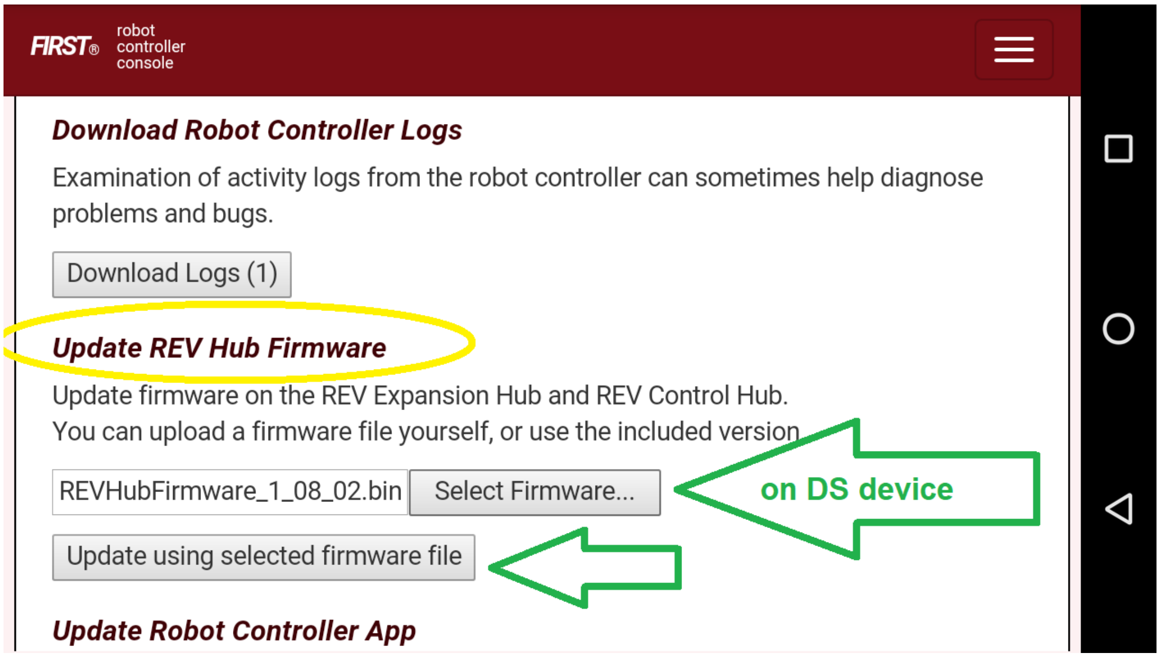


Fig. 32: Update Hub Firmware

   See if the grey box "Update to…" offers the latest firmware version, included or bundled with the DS app.

3. If not, you can transfer the desired firmware file to the **Driver Station device**.

   Yes, that's correct: transfer to the DS device, not to the RC device. This Method 5 uses a local file on the DS device, while Methods 2 and 3 (above) use a local file on the RC device.

   Use a USB data cable (not a charge-only cable) to store the firmware file in the DS device's Downloads folder.

   Current and older firmware files can be found at the REV Robotics website here.

   Then click the "Select Firmware…" box. Navigate to the DS device's Downloads folder, and select the desired firmware file.
4. Now click the box called "Update to…" or "Update using…" (second green arrow, above).
5. At the confirmation prompt, scroll down and click the blue box "Update Hub Firmware". Wait for the process to finish; do not unplug the Hub or restart the robot.

That's it! The Hub's firmware is now updated.

Questions, comments and corrections to westsiderobotics@verizon.net

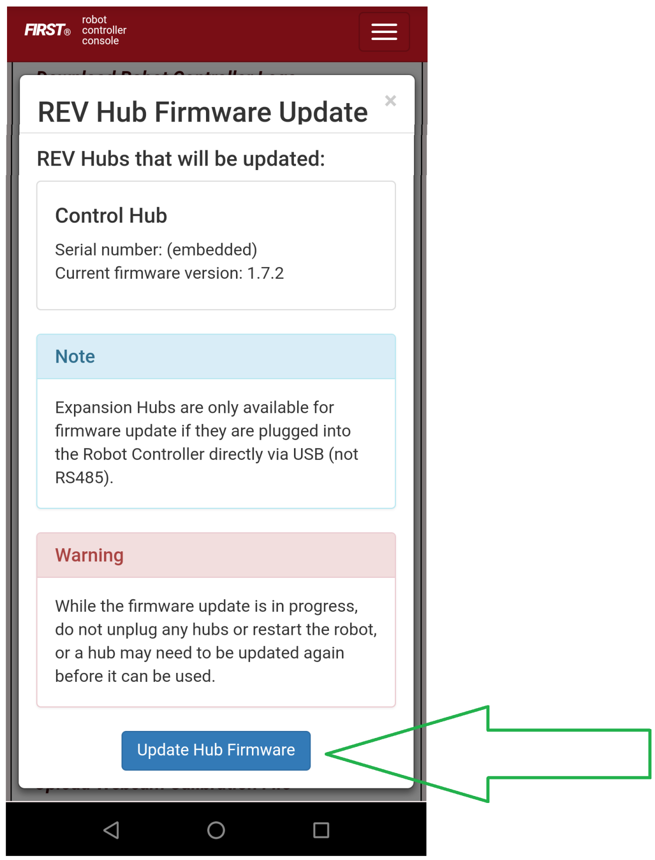


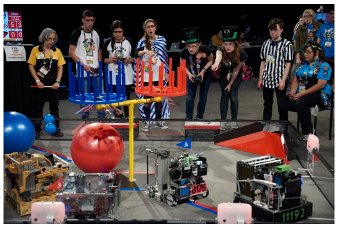
Fig. 33: Update Hub Firmware

Chapter 17

Control System Introduction

17.1 About *FIRST* Tech Challenge

FIRST Tech Challenge seeks to inspire youth to become the next generation of STEM leaders and innovators through participation in mentor-guided robotics competition. Teams who participate in *FIRST* Tech Challenge must build a robot that performs a variety of tasks. The tasks vary from season to season, and are based on a set of game rules that are published at the start of each season. The more tasks that a robot can complete, the more points a team will earn.



(Photo courtesy of Dan Donovan, ©2017 Dan Donovan / www.dandonovan.com)

17.2 AUTO vs. TELEOP

A *FIRST* Tech Challenge match has an AUTO phase and a TELEOP phase. In the AUTO phase of a match the robot operates without any human input or control. In the TELEOP phase, the robot can receive input from up to two human drivers.

17.3 Point-to-Point Control System

FIRST Tech Challenge uses Android devices to control its robots. During a competition, each team has two Android devices.

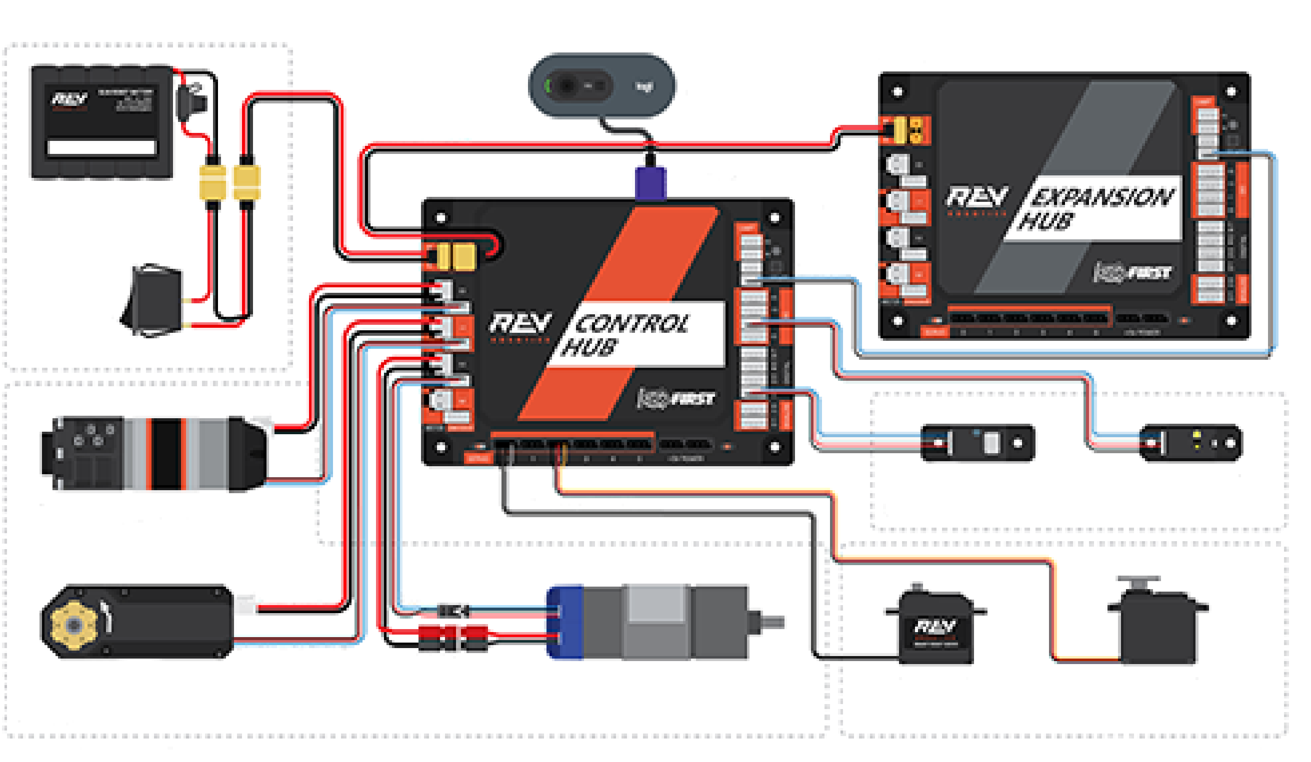
One Android device is mounted onto the robot and is called the *Robot Controller*. In most cases, the ROBOT CONTROLLER is integrated into the REV Robotics Control Hub. The Robot Controller acts as the "brains" of the robot. It does all of the thinking for the robot and tells the robot what to do. It consists of an Android device running a Robot Controller app. Many Teams will also connect a REV Robotics Expansion Hub for additional ports to connect motors, servos and sensors to the ROBOT.

A second Android device sits with the team drivers and has one or two gamepads connected. This second device is known as the DRIVER STATION. The DRIVER STATION is like a remote control that you might use to control your television. The DRIVER STATION allows a team to communicate remotely (using a secure, wireless connection) to the Robot Controller and to issue commands to the Robot Controller. The DRIVER STATION consists of an Android device running an Driver Station app. Most teams use a REV Robotics Driver Hub, but select Android smartphones are also supported.

17.4 REV Robotics Control Hub and Expansion Hub

The REV Robotics Control Hub or Expansion Hub is the electronic input/output (or "I/O") module that lets the Robot Controller talk to the robot's motors, servos, and sensors. The Robot Controller is integrated into the Control Hub, and communicates with the Expansion Hub through a serial connection. For the situation where an Android smartphone is used as the Robot Controller, a USB cable is used to establish the serial connection.

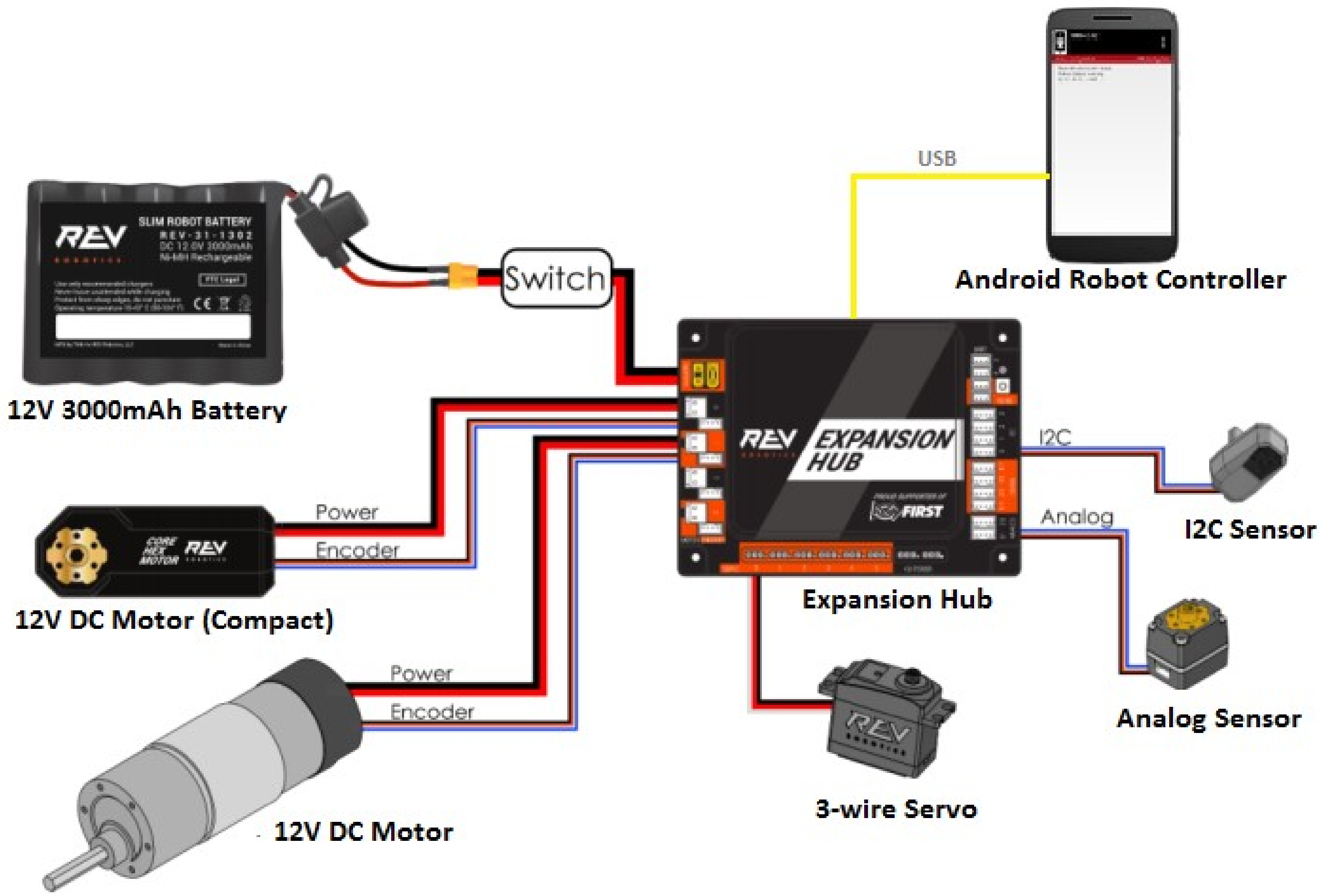
The Control Hub and Expansion Hub are also connected to a 12V battery which is used to power the Control Hub, the Expansion Hub, the motors, the servos and sensors. If an Android smartphone is used as the Robot Controller, then the smartphone will have its own independent battery.



17.5 Android Smartphones

Teams may opt to use an Android smartphone as their DRIVER STATION, ROBOT CONTROLLER or both. DRIVER STATION phones should have the FTC Driver Station app installed, and will need an OTG adaptor USB hub to connect gamepads.

Teams who use an Android smartphone as their ROBOT CONTROLLER will want an additional REV Robotics Expansion Hub to connect motors, servos and sensors. The smartphone is connected to the Expansion Hub via a USB-A to USB-Mini cable and an OTG adaptor.



17.6 What's an OpMode?

During a typical *FIRST* Tech Challenge match, a team's robot has to perform a variety of tasks in an effort to score points. For example, a team might want their robot to follow a white line on the competition floor and then score a game element (such as a ball) into a goal autonomously during a match. Teams write "OpModes" (which stand for "operational modes") to specify the behavior for their robot.

An *OpMode* is a computer program that is used to customize the behavior of a competition robot. The Robot Controller can *execute* a selected OpMode to perform certain tasks during a match.

Teams who are participating in *FIRST* Tech Challenge have a variety of programming tools that they can use to create their own OpMode. Teams can use a visual ("drag and drop") programming tool called the *Blocks Programming Tool* to create their op modes. Teams can also use a text-based Java tool known as the *OnBot Java Programming Tool* or Google's *Android Studio* integrated development environment (also known as an "IDE") to create their OpModes.

Chapter 18

Hardware Component Overview

The *FIRST* Tech Challenge Control System is divided into two main components: the Driver Station (DS) and the Robot Controller (RC). This section will give you a brief introduction to the hardware components, their various configurations, and connections.

18.1 Driver Station Overview

These images represent a basic connection diagram for the components that have typically been used to create a Driver Station. These components have typically been purchased from the *FIRST* Storefront (in the "Control and Communication" kit). These configurations show sample connections, and in no way represent the only possible way of connecting these components. These images also do not represent using a Driver Station Carrier, which is recommended for teams to use for component management and transportation. See rule `DS07` in *Game Manual 1* for more information on Driver Station Carriers.

Driver Station

```
Driver Station C1 - Basic Config
```

License

```
License and Attribution
```

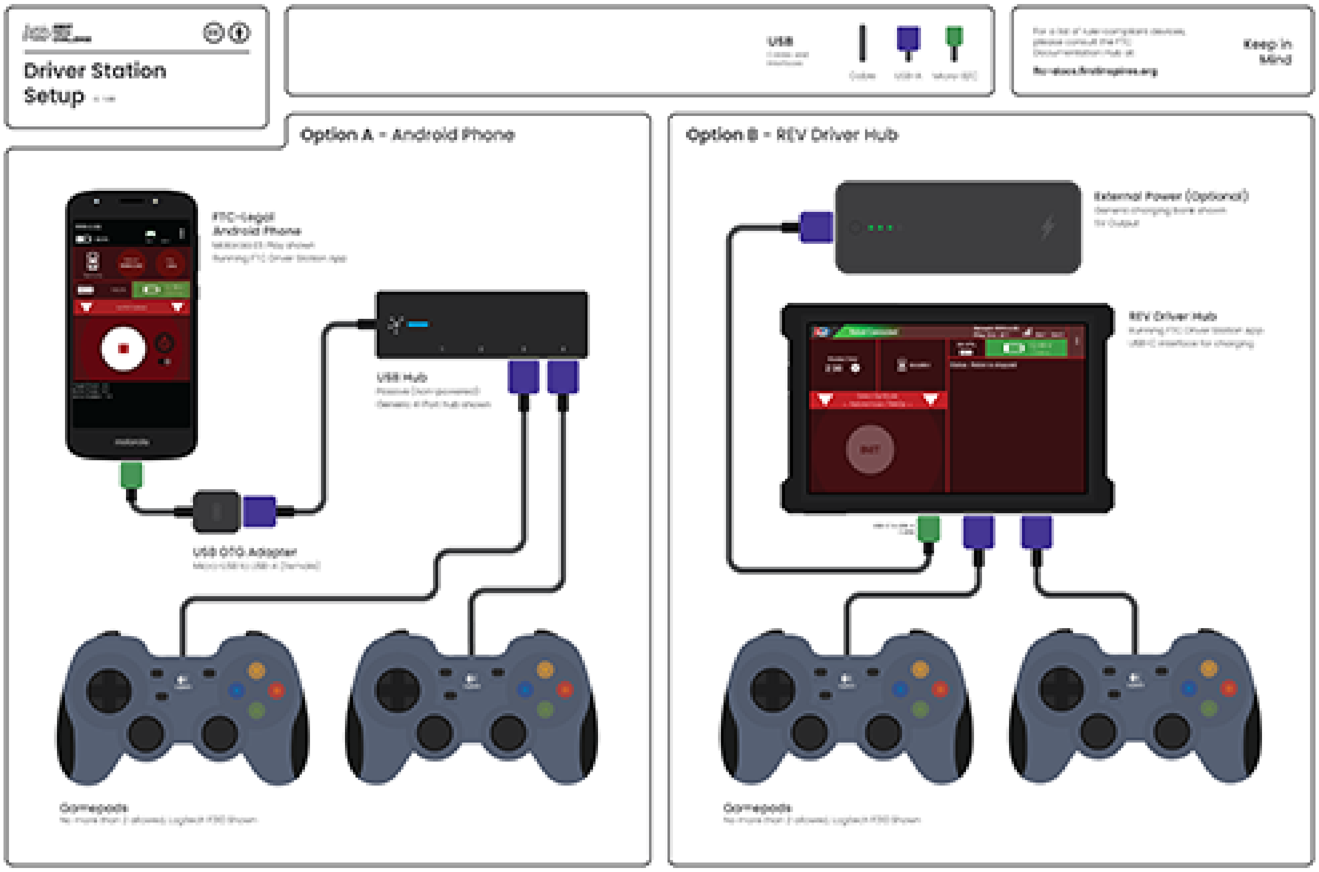


Fig. 1: Diagram courtesy of FRC Team 3161 and Stefen Acepcion

18.1.1 Driver Station Components

Android Device

REV Driver Hub

REV Driver Hub (REV-31-1596)

Android Smartphone

Moto E5

The heart of the Driver Station is the Android Device that runs the Driver Station App. This Android Device requirement can be fulfilled either through the use of a REV Driver Hub or one of the approved Android Smartphones listed in *Game Manual 1*. It is of vital importance that the Driver Station App be updated to a version that meets or exceeds the minimum Driver Station App version as defined in *Game Manual 1*.


NETWORK
USB 2.0
USB C
USB 2.0
USB 2.0
POWER


Google
Duo
Moto
motorola

USB-OTG Adapter / Hubs

USB OTG Cable

USB OTG Adapter Cable

USB Hub

Anker USB Hub

USB OTG Hub

REV UltraUSB (REV-31-1592)

If the Android Device being used is an Android Smartphone, the smartphone only provides a single USB-Micro-B port on the bottom of the phone. In order to use USB devices with the Android Smartphone, like a gamepad, a USB-OTG Adapter Cable

must be used. This cable provides a USB Type A port for the Gamepad or peripherals (like a USB Hub, to allow more than one Gamepad to be used). If available, it is instead recommended to use a USB Hub with OTG cable built in, like the REV UltraUSB (REV-31-1592) - this reduces the number of connections and failure points in the system.

When using a REV Driver Hub, no OTG adapters are necessary - gamepads may connect directly to one of the three USB-A ports on the device.

Comercial USB Battery Pack

USB Battery Pack

Anker Battery Pack

A commercial USB battery pack is an auxiliary power source that can be used in specific situations in accordance with the *Game Manuals*. A USB battery pack is permitted to be used to charge your Android Device. Only the REV Driver Hub can be charged while in use, through its USB-C port.

Gamepads

Logitech F310

Logitech F310 Gamepad

Sony DualSense

Sony DualSense Gamepad

Sony Dualshock 4

Sony DualShock 4 Gamepad

Etpark PS4 Wired

Etpark PS4 Wired Gamepad

Xbox 360

Xbox 360 Gamepad

Quadstick

Quadstick FPS

Game Manual 1 defines the gamepads that are allowed in competition play. Up to two gamepads, in any combination, of the allowed types of gamepads may be used. All gamepads MUST be used in wired mode only, no wireless of any kind is allowed. Special features of some gamepads (Rumble, Lighting) may be programmed and used by teams for notifications and signaling to the drivers of the robot.


SHARE
OPTIONS

18.2 Robot Controller Overview

These images represent a basic connection diagram for the components that have typically been included in a standard Robot Starter Kit plus the components purchased from the *FIRST* Storefront (demonstrating components from the REV and Tetrix starter kits, along with the Electronics kit). These configurations show sample connections, and in no way represent the only possible way of connecting these components. In both diagrams is an extra optional REV Expansion Hub that has NOT been included with standard starter kits nor electronics kits; it is included in these diagrams as a sample of how to connect an additional optional REV Expansion Hub if one is available and desired.

Click on the headers below to switch between the different control system configuration diagrams.

Control Hub

```
Control Hub B1 - Basic Config
```

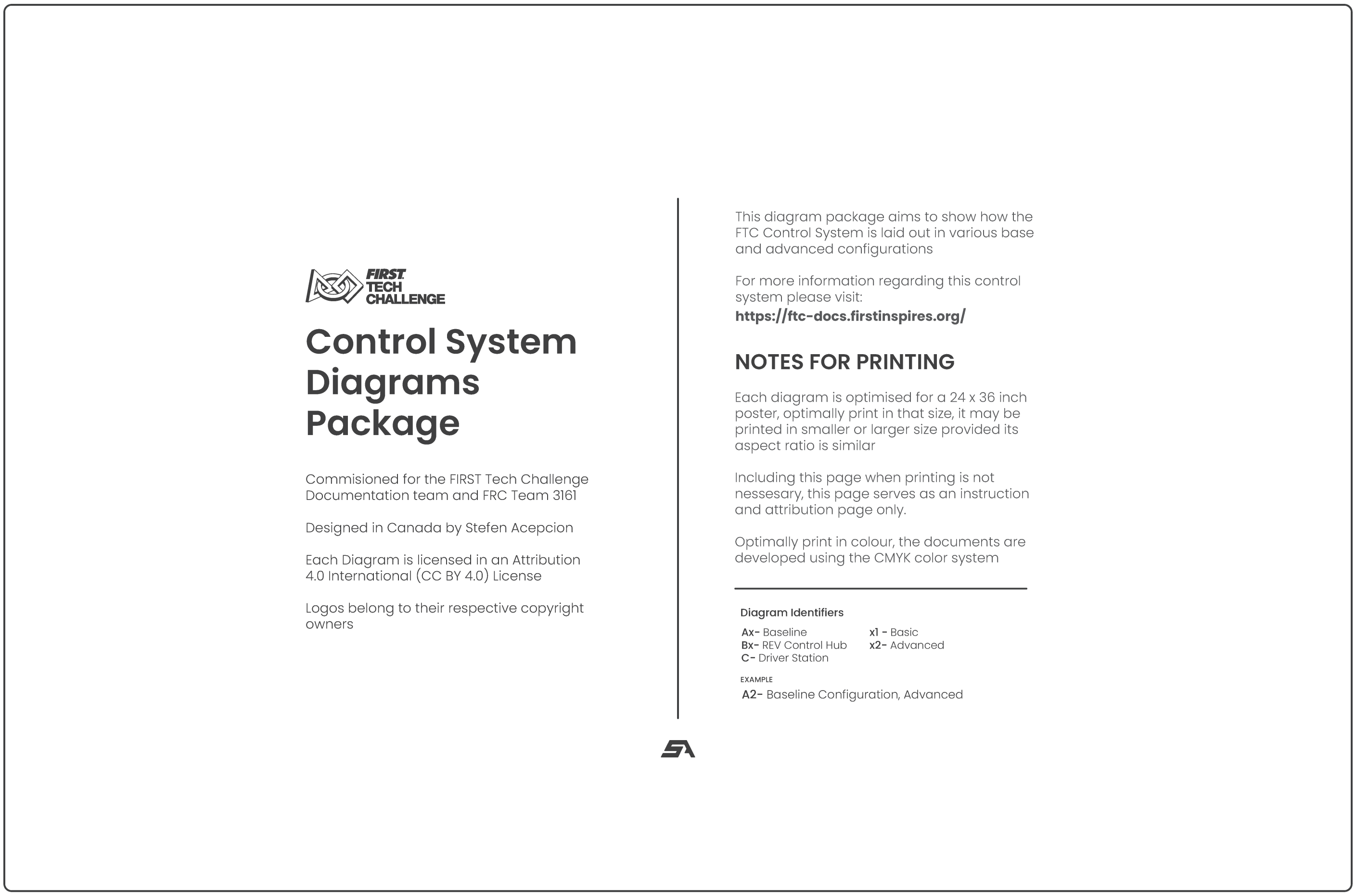
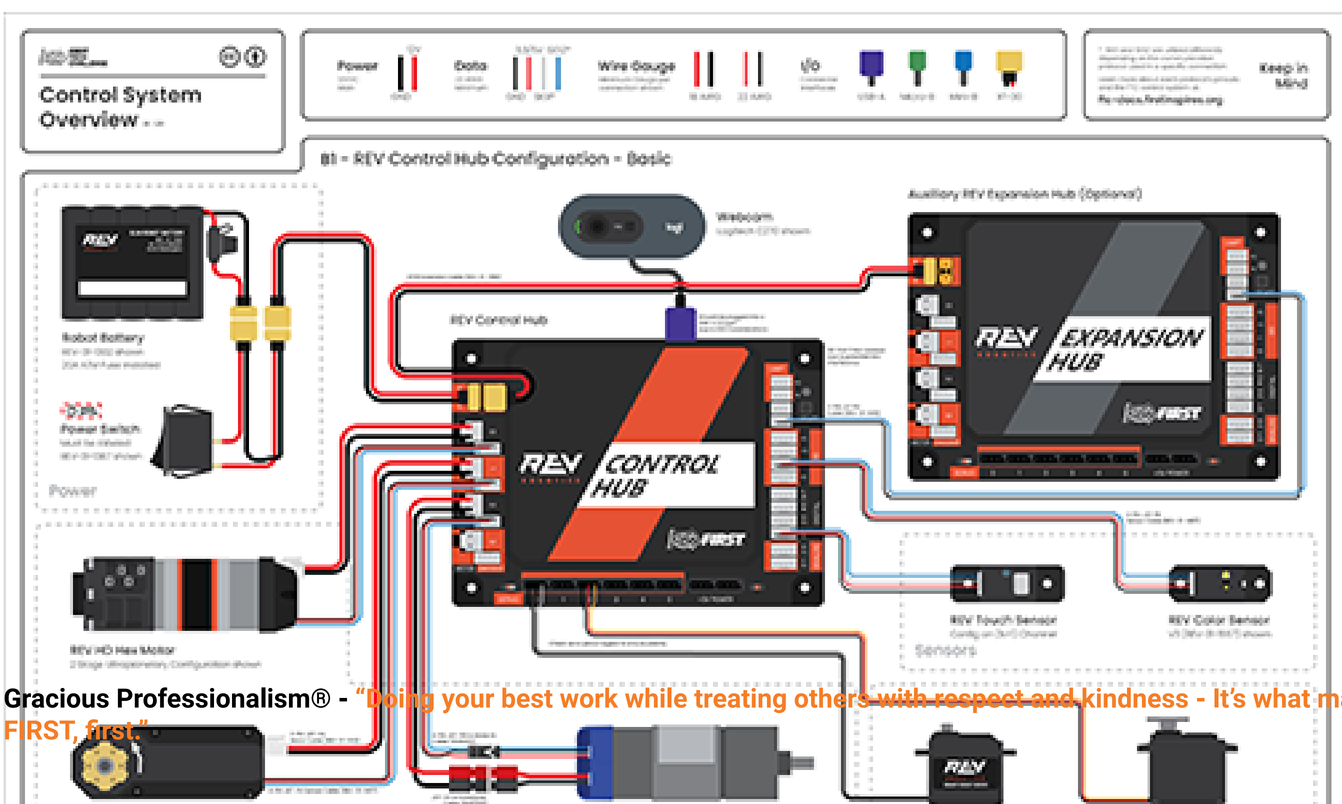


Fig. 2: Diagram courtesy of FRC Team 3161 and Stefen Acepcion



Control Hub B2 - Advanced Config

Android Phone / Hub

`Android Phone A1 - Basic Config`

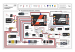


Fig. 4: Diagrams courtesy of FRC Team 3161 and Stefen Acepcion

`Android Phone A1 - Advanced Config`

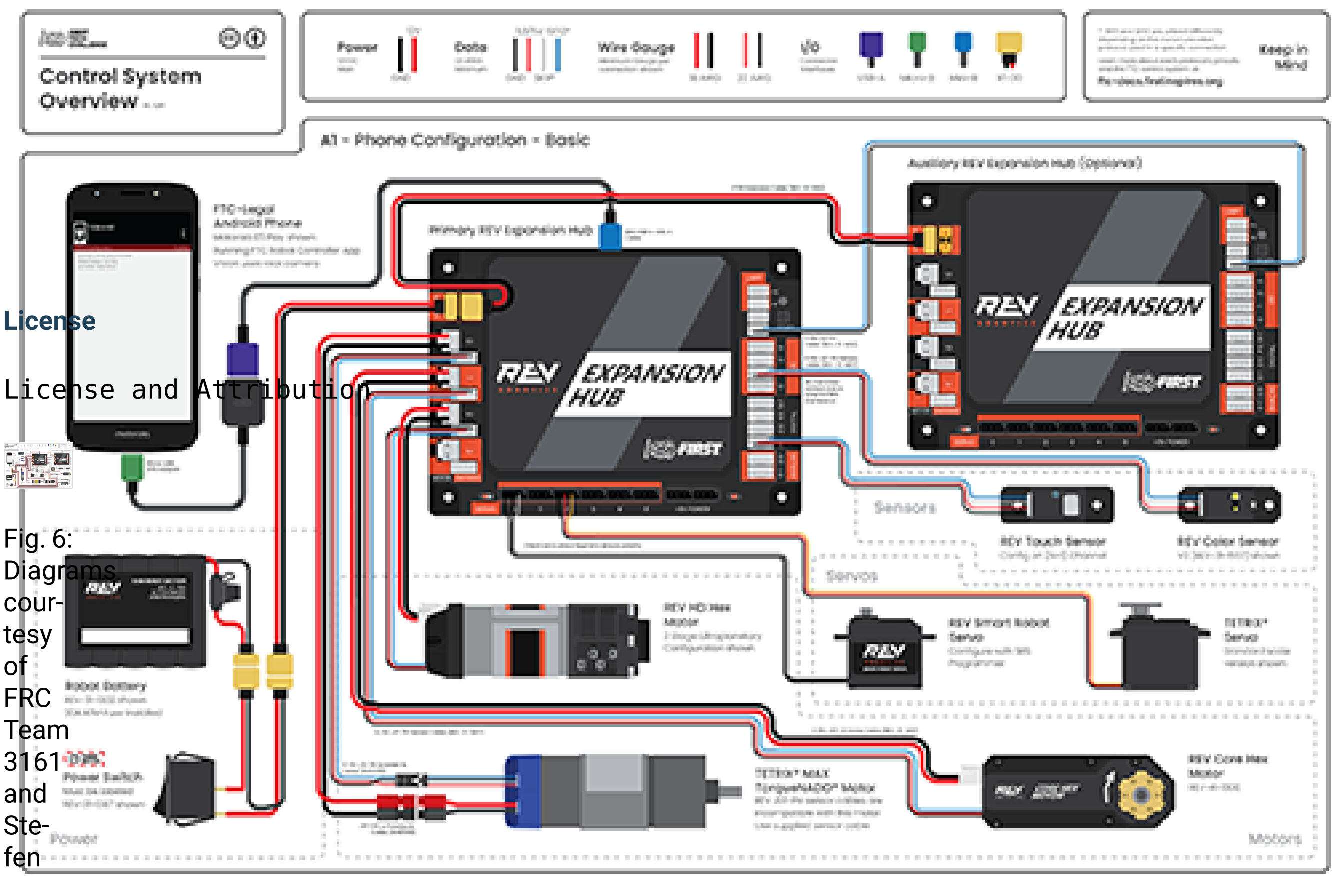


Fig. 5: Diagrams courtesy of FRC Team 3161 and Stefen Acepcion

License

`License and Attribution`

Fig. 6: Diagrams courtesy of FRC Team 3161 and Stefen Acepcion

18.2.1 Power Distribution

Robot Main Battery

TETRIX 12V Battery

TETRIX (W39057, formally 739023)

MATRIX 12V Battery

Modern Robotics/MATRIX (14-0014)

REV 12V Battery

REV Robotics (REV-31-1302)

The main power of a robot comes from one 12v battery. The battery may be one of the batteries shown above. Refer to section <RE03> in the *Game Manual Part 1* for exact information on allowed batteries. Note that it is typically allowed by <RE15> to replace the connector on the batteries, provided the in-line fuse on the battery is preserved.

> **Warning:** Be sure to remove the 20A fuse from the in-line fuse holder prior to cutting any wires/connectors if/when replacing the factory default battery connector.

LEGAL


MODERN ROBOTICS
12V 3000mAh
NiMH Battery


REV
21051T
SLIM ROBOT BATTERY
REV-31-1302
DC 12.0V 3000mAh
Ni-MH Rechargeable
FTC Legal
Made in China

Main Power Switch

REV Power Switch

REV (REV-31-1387)

TETRIX Power Switch

TETRIX (part # W39129)

MATRIX Power Switch

MATRIX (part #50-0030)

AndyMark Power Switch

AndyMark (part #am-4969)

One Main Power Switch must control all power provided by the Main Battery. It along with its label should be placed in accordance to *Game Manual Part 1*. The legal power switches are shown above. <RE01>

`Power Switch Label`

Power Distribution Block

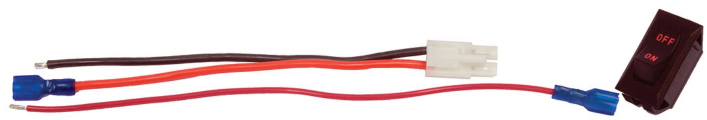
REV XT30 Power Distribution Block

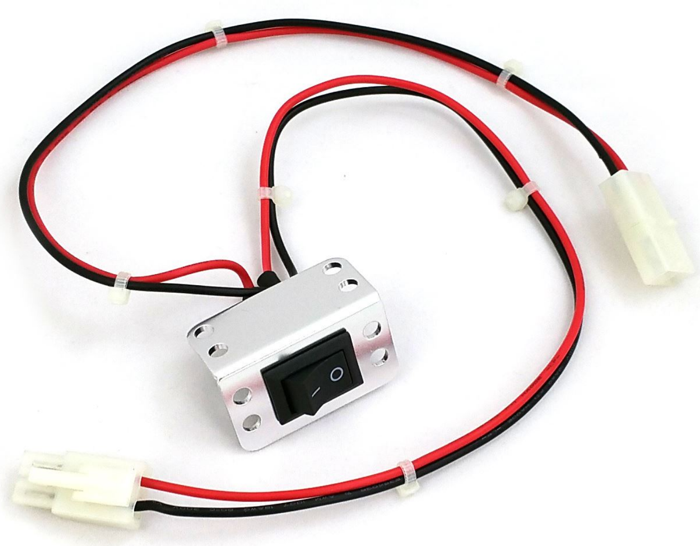
REV (REV-31-1293)

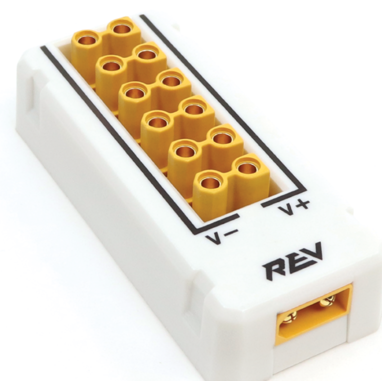
goBILDA XT30 Power Distribution Block

goBILDA (SKU: 3108-2833-0801)

Power Distribution Blocks help to distribute the power to devices such as Control Hubs, SPARKminis, and more. See *Game Manual Part 1* for a description of legal Power Distribution methods. The Power Distribution Blocks shown are not the only legal devices for power distribution.


OFF
ON




V–
V+
REV

REV Servo Power Module

REV Servo Power Module

REV (REV-11-1144)

This is an electronic device that boosts the power supplied to 3-wire servos. A REV Servo Power Module has 6 input servo ports and 6 matching output ports. It draws power from a 12V source and provides 6V power to each output servo port. A REV Servo Power Module can provide up to 15A of current across all output servo ports for a total of 90 Watts of power per module.

COTS USB Battery Pack

USB Battery Pack

Anker Battery Pack

A Commercial Off The Shelf (COTS) USB battery pack is an auxiliary power source that can be used in specific situations in accordance with the *Game Manuals*. In the 2023-2024 season, these batteries were deemed permissible to power LEDs (per <RE12>`f.ii`) and, by extension, COTS light controller sources like the REV Blinkin (per <RE12>e). However, having a COTS USB External Battery on the Robot carries additional considerations. All teams must ensure their COTS USB Battery Pack:

- Is manufactured by a reputable brand.
- Is within allowed Watt-hour capacity limits.
- Includes standard safety features.
- Is secured on the Robot.
- Has unused ports covered.
- Is always charged properly.
- Does not show any signs of distress.
- Is never connected to the Robot power


REV


ANKER
Input

The following sections are intended to help clarify the list above.

Reputable Brands

Far and above, the most important factor regarding the safety of COTS USB Battery Packs is ensuring that the battery pack was manufactured by a reputable brand. International testing of COTS USB Battery Packs has concluded that unbranded batteries, or batteries manufactured by little-known companies, tend to fail more often than batteries from reputable brands. How do you know what brands are reputable and which ones are not? That's not always an easy thing to determine, however brands such as Anker, Belkin, Otterbox, and BioLite are among the most-used brands in the world. *FIRST* Tech Challenge recommends choosing an internationally reputable brand, even if the brand is more expensive than a lesser-known brand, as these batteries will be more apt to follow safety and performance guidelines. NEVER choose a COTS USB Battery Pack based on its (low) price alone!

Capacity Limits

The recurring theme in most discussions of COTS USB Battery Packs is safety. The United States Transportation Safety Administration (TSA) has strict limitations on COTS USB Battery Packs aboard aircraft, and *FIRST* Tech Challenge has adopted the capacity limit restriction. **Batteries are limited to 100 Watt-Hours (Wh) or less**.

How do you calculate Watt-hours? To calculate Watt-hours of a battery, multiply the Voltage (V) of the battery by its capacity measured in Amp-Hours (Ah). For example, a 12V battery with 3,000mAh capacity has a 36Wh capacity - when capacity is measured in milli-Amp Hours (mAh), divide the capacity by 1000 to get Ah and them multiply by Voltage. However, for COTS USB Battery Packs, the Voltage cells predominantly used in the packs is **3.7V**, regardless of the ultimate Voltage provided by the USB ports. Therefore to calculate Wh for a COTS USB Battery Pack, multiply **3.7V** by the **Ah rating** of the pack. A 25,000mAh COTS USB Battery Pack has a rating of 92.5Wh. Using this formula, the maximum capacity COTS USB Battery Pack that is allowed is a **27,000mAh** pack.

Standard Safety Features

The major benefit of using a reputable COTS USB Battery Pack brand is the guarantee that the battery pack includes standard safety features, including but not limited to:

- Reverse Polarity Protection
- Short-Circuit Protection
- Over-Charge Protection
- Over-Temperature or Over-Heat Protection
- Over-Current Protection

You should perform a good-faith effort to determine if your Battery Pack contains these safety features. Often within the documentation provided with your pack it will list the protections offered by the pack. Remember that the Battery Pack likely contains Lithium-Ion or Lithium Polymer batteries that will often explode or catch fire when they fail, and these protections are vital to ensuring that the batteries do not fail prematurely. It is not recommended to use COTS USB Battery Packs without these protections.

Securing the Battery Pack to the Robot

The leading cause of battery failure is physical damage to the battery. For COTS USB Battery Packs this is usually attributed to dropping the battery pack, applying excessive force on the Battery Pack, and subjecting the pack to excessive shock (which might also damage internal components). In order to prevent damage, the Battery Pack should be properly secured within the robot. Tips for securing the battery are:

- Use Hook and Loop or 3M DualLock fasteners to secure the battery, **OR**
- Store the battery in a tight-fitting or custom-fit enclosure within the robot that allows the battery to be exposed to air (for cooling), **AND**
- Protect the battery from contact from other robots, game pieces, or field elements that might breach the perimeter of the robot.

If utilizing a COTS USB Battery Pack, it is of utmost importance to ensure that the battery is secured, protected, and ventilated. All batteries (both main batteries and COTS USB Battery Packs) should be easily accessible and be able to be quickly removed from the robot in case of an emergency.

Cover Unused Ports

Some COTS USB Battery Packs contain multiple ports, and it is often that not all ports are in-use while securely mounted to the robot. For example, the COTS USB Battery Pack may have multiple USB ports, a dedicated charging port, and other ports as necessary. Any ports that are not in-use (meaning don't have a USB connector inserted in them) are at great risk of short-circuiting. The most common reason for short-circuiting is metal fragments that may make their way into the ports, especially swarf due to metal rubbing together, gears wearing, or robot maintenance performed while electronics are present. Any unused ports should be covered using electrical tape, Gaffers Tape, or any other means of preventing debris from entering the ports. Short circuits may present risks of excessive heat, fire, or explosion and all reasonable efforts should be taken to prevent them.

Warning: Never get a COTS USB Battery Pack wet. If it gets wet, follow the manufacturer's recommended procedure to clean and dry the battery before continuing use.

Charge COTS USB Battery Packs properly

The owners/instruction manual provided with the COTS USB Battery Pack often contains instructions for the proper care and maintenance of the battery pack, including proper charging. Always follow the manufacturer's recommendations. In addition, these are common best practices for charging your Batteries:

- Avoid charging the power bank on places that build up heat, such as on your bed or within a bag.
- Unless it's a solar power bank, NEVER leave your battery in the sun!
- Follow the manufacturer's guidelines on the time required to fully charge your COTS USB Battery Pack.
- Avoid leaving your COTS USB Battery Pack on prolonged charge as this may cause it to overheat.
- If the COTS USB Battery Pack becomes excessively hot during charging or discharging, unplug it from the power source or powered device immediately and allow it to cool before doing anything else with the battery.

Checking for signs of distress

Most COTS USB Battery Packs are contained within a hard plastic shell in order to protect and package the battery cell(s) within. Therefore it can be difficult to determine if the battery is showing signs of failure and distress. Here are several tips for identifying a failing battery:

- Check for Leaking Power Cells. Similar to an acid leak in an alkaline battery, check to see if there are any signs of corrosion or acid leak from the battery pack. This might be difficult to determine, so stay vigilent. If signs of acid or corrosion are present, dispose of the battery per the manufacturer's recommendations immediately with extreme prejudice.
- Look for bulging within the battery casing. When Lithium batteries fail, often they will begin to bulge like a balloon. If the case of the battery shows any signs of pressure from within, dispose of the battery per the manufacturer's recommendations immediately with extreme prejudice.
- Test the battery pack for any non-functional ports. Sometimes non-functional ports can be an early sign of internal damage. DO NOT use batteries that are suspected of being damaged - dispose of the battery per the manufacturer's recommendations immediately.

Isolate COTS USB Battery Packs from the Robot Power

Great care must be take to NEVER allow the COTS USB Battery Pack to be connected to the main (or any) power system in use by the robot. The COTS USB Battery Pack and connected devices must be completely isolated from the robot electrical system, with the exception of controlling signals provided by the Game Manual (per rule <RE12>`.d`). When using a COTS USB Battery Pack, controlling signals for LEDs powered by the Pack should ONLY connect to compatible devices listed in rule <RE12>`.e`.

18.2.2 REV Hubs

The REV Hubs are the core control units of a *FIRST* Tech Challenge robot.

Control Hub

Control Hub Ports

Battery Ports

> **Danger: Never** connect a battery charger directly to the battery port. This will void your warranty and fry your hub.

These XT-30 connectors are used to power your REV Hub as well as all the devices connected to it. As the connector is known for its fragility it is highly recommended you be careful when using it. It is also recommended that you expand your connector prongs periodically. For more information on this process please watch this video. While this video features an XT60, a larger version of the XT-30, and a drone the advice is much the same. This port may also be used to connect a grounding strap. For more information on legal grounding straps see <RE15>, *Game Manual Part 1*. For more information on this port please see REV Documentation.

Fig. 8: REV Control Hub (REV-31-1595)

Motor Ports

These JST-VH style connectors are used to power your motors. There are 4 of these ports per hub and they are numbered from 0-3. As you are able to use 8 motors per robot you may want to control more than these hubs allow. In that case it is possible for you to use an *additional hub* or to use a REV SPARKmini Motor Controller (REV-31-1230) to power more motors. For more information on this port please see REV Motor Port Documentation.

Encoder Ports

These 4-pin JST-PH style connectors are used for your quadrature encoders. There are 4 of these ports on each hub and they can be used in tandem with the motor they are adjacent to. However, it is also possible to use this port to connect to a standalone incremental encoder. To connect to more than 4 encoders it is currently necessary to connect an additional hub. For more information on this port please see REV Encoder Port Documentation.

Servo Ports

These 0.1" Header pins are used to power and control your servos. There are 6 ports on each hub and they are numbered from 0-5. Be mindful of matching the polarity of the device attached to this port as it is possible to flip the connector. For increasing the power supplied to these servos it is possible to use a Servo Power Module that is in compliance with <RE05>, *Game Manual Part 1*. For more information on this port please see REV Servo Port Documentation.

+5V Power Ports

These 0.1" Header pins are used to power and control various appliances. There are two ports on each hub. These connectors can be used for a limited range of applications in FIRST Tech Challenge, such as powering powered USB hubs. For more information on this port please see REV +5V Power Port Documentation and *Game Manual Part 1*.

Analog Ports

These 4-pin JST-PH style connectors are used for your analog inputs. There are 2 of these ports on each hub. These ports have 4 channels labeled from 0-4. This port can be used to connect to a standalone analog sensor. A common example of an analog sensor is a potentiometer. An analog sensor is one that outputs a range of values rather than digital which alternates between one of two states. For more information on this port please see REV Analog Port Documentation.

Digital Ports

These 4-pin JST-PH style connectors are used for your digital inputs. There are 4 of these ports on each hub with a total of 8 channels labeled from *0-7*. A device attached to a digital port alternates between one of two states (e.g., on and off). One such device would be a button. It is important to note that each port has two channels and devices such as the REV Touch Sensor will only operate on one channel (N+1).

I2C Ports

These 4-pin JST-PH style connectors are used for connecting I2C sensors. Each port is a single I2C bus where multiple sensors can be attached. Using sensors with identical addresses on the same bus can cause problems. The range of I2C sensors that can be connected is limited by *Game Manual Part 1*. While it is possible to use a large range of sensors, the vast majority of I2C sensors do not have drivers built into the SDK. It is possible to use community drivers or create your own. For more information on this port please see REV I2C Port Documentation.

RS485

These 3-pin JST-PH style connectors are used for serial communication between REV Hubs. You would use this port if you wished to use a second REV Hub as described *in this tutorial*. Both RS485 ports can be used to add redundancy by using two cables connecting both ports between the REV Hubs.

UART

This connector is used only for **Developer** (non end user) debugging. Its use is not supported by FIRST.

USB-A Ports

A Control Hub has both a USB 2.0 and 3.0 Type-A female port. This is primarily used for connecting UVC Cameras in acccordance with <RE14>.

Fig. 9: Control Hub Front Ports

USB-B Port

On a Control Hub, the USB-mini-B port is used only to communicate directly to the I/O system. In this case, it is only for the purpose of uploading firmware to the device.

USB-C Ports

A Control Hub has a USB-C port. This is primarily used for connecting to a laptop for loading the SDK but can also be used with a UVC Camera in acccordance with <RE14>.

HDMI

The Control Hub lacks a display of its own even though it is a fully-fledged Android device. The Control Hub has an HDMI port that provides video output for the device; this HDMI port can be used to connect to an external display.

A REV Control Hub combines a REV Expansion Hub with an embedded Android daughterboard connected to it. This means it is able to control all of the hardware components of your robot and also run your actual robot software. This is in contrast to the REV Expansion Hub which was only able to control hardware devices but is not able to to interpret and run the SDK.

Expansion Hub

Expansion Hub Ports

Battery Ports

Danger: Never connect a battery charger directly to the battery port. This will void your warranty and fry your hub.

These XT-30 connectors are used to power your REV Hub as well as all the devices connected to it. As the connector is known for its fragility it is highly recommended you be careful when using it. It is also recommended that you expand your connector prongs periodically. For more information on this process please watch this video. While this video features an XT60, a larger version of the XT-30, and a drone the advice is much the same. This port may also be used to connect a grounding strap. For more information on legal grounding straps see <RE15>, *Game Manual Part 1*. For more information on this port please see REV Documentation.

Fig. 10: REV Control Hub (REV-31-1595)

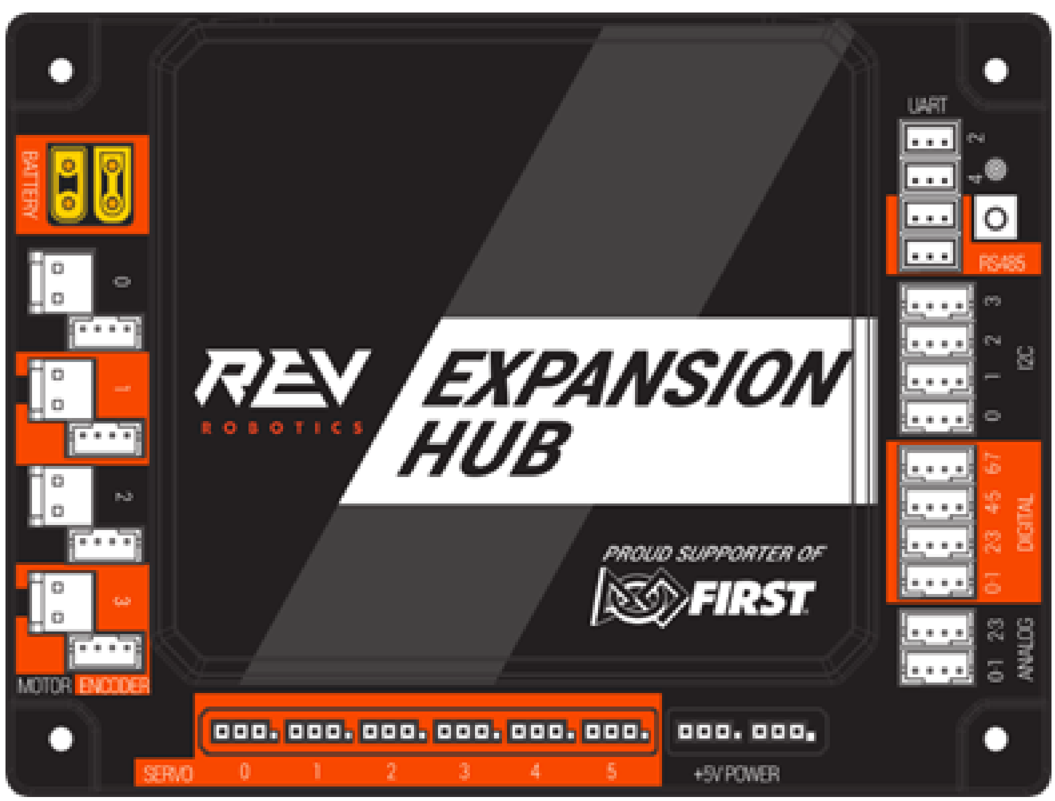


Fig. 11: REV Expansion Hub (REV-31-1153)

Motor Ports

These JST-VH style connectors are used to power your motors. There are 4 of these ports per hub and they are numbered from 0-3. As you are able to use 8 motors per robot you may want to control more than these hubs allow. In that case it is possible for you to use an *additional hub* or to use a REV SPARKmini Motor Controller (REV-31-1230) to power more motors. For more information on this port please see REV Motor Port Documentation.

Encoder Ports

These 4-pin JST-PH style connectors are used for your quadrature encoders. There are 4 of these ports on each hub and they can be used in tandem with the motor they are adjacent to. However, it is also possible to use this port to connect to a standalone incremental encoder. To connect to more than 4 encoders it is currently necessary to connect an additional hub. For more information on this port please see REV Encoder Port Documentation.

Servo Ports

These 0.1" Header pins are used to power and control your servos. There are 6 ports on each hub and they are numbered from 0-5. Be mindful of matching the polarity of the device attached to this port as it is possible to flip the connector. For increasing the power supplied to these servos it is possible to use a Servo Power Module that is in compliance with <RE05>, *Game Manual Part 1*. For more information on this port please see REV Servo Port Documentation.

+5V Power Ports

These 0.1" Header pins are used to power and control various appliances. There are two ports on each hub. These connectors can be used for a limited range of applications in FIRST Tech Challenge, such as powering powered USB hubs. For more information on this port please see REV +5V Power Port Documentation and *Game Manual Part 1*.

Analog Ports

These 4-pin JST-PH style connectors are used for your analog inputs. There are 2 of these ports on each hub. These ports have 4 channels labeled from 0-4. This port can be used to connect to a standalone analog sensor. A common example of an analog sensor is a potentiometer. An analog sensor is one that outputs a range of values rather than digital which alternates between one of two states. For more information on this port please see REV Analog Port Documentation.

Digital Ports

These 4-pin JST-PH style connectors are used for your digital inputs. There are 4 of these ports on each hub with a total of 8 channels labeled from *0-7*. A device attached to a digital port alternates between one of two states (e.g., on and off). One such device would be a button. It is important to note that each port has two channels and devices such as the REV Touch Sensor will only operate on one channel (N+1).

I2C Ports

These 4-pin JST-PH style connectors are used for connecting I2C sensors. Each port is a single I2C bus where multiple sensors can be attached. Using sensors with identical addresses on the same bus can cause problems. The range of I2C sensors that can be connected is limited by *Game Manual Part 1*. While it is possible to use a large range of sensors, the vast majority of I2C sensors do not have drivers built into the SDK. It is possible to use community drivers or create your own. For more information on this port please see REV I2C Port Documentation.

RS485

These 3-pin JST-PH style connectors are used for serial communication between REV Hubs. You would use this port if you wished to use a second REV Hub as described *in this tutorial*. Both RS485 ports can be used to add redundancy by using two cables connecting both ports between the REV Hubs.

UART

This connector is used only for **Developer** (non end user) debugging. Its use is not supported by FIRST.

USB-B Port

An Android RC phone controls an Expansion Hub through this USB-mini-B port, connected with USB OTG cable. This port also allows firmware updates.

Fig. 12: REV Expansion Hub (REV-31-1153)

A REV Expansion Hub is a hub that is used to control all of the hardware components of your robot. It takes the commands your Android Device sends and actually makes it happen. If you want to move a motor, an Expansion Hub is what takes the instruction of moving the motor and actually sends power to the motor in the correct manner. It however does not know when to do this which is where the Android Device comes into play. This device can either be a traditional Android

Phone connected via USB or just the embedded device in a Control Hub. When using more than one hub, these hubs can be connected via RS485 or USB. More information can be found *here*.

18.2.3 Motors

TETRIX 12V DC Motor

TETRIX 12V TorqueNADO DC Motor

AndyMark 12V DC Motor

AndyMark NeveRest series 12V DC Motors

MATRIX 12V DC Motors

Modern Robotics/MATRIX 12V DC Motors

REV 12V DC Motor

REV Robotics HD Hex 12V DC Motor

REV 12V DC Motor

REV Robotics Core Hex 12V DC Motor

Motors are the primary drivers of a robot. All motors are 12v brushed DC motors and are enumerated in *Game Manual Part 1*. They may only be controlled via a REV Expansion Hub, REV Control Hub, REV SPARKmini Motor Controller, or VEX Motor Controller 29. <RE09>


AndyMark
am-3103
NeveRest 60 Motor


FIRST
TECH
CHALLENGE
LEGAL
goBILDA
5202 SERIES
YELLOW JACKET
435 RPM @ 12VDC
PLANETARY GEAR MOTOR


HD HEX
MOTOR


REV
ROBOTICS
CORE HEX
MOTOR

Additional Resources

- *Port Overview*
- *Connecting Motors*
- *Configuring Motors*

18.2.4 Encoders (Rotation Counters)

REV HD Hex Motor

Built-in Encoder in the REV HD Hex Motor

REV Through Bore Encoder

REV Through Bore Encoder, in incremental mode.

An encoder is a device that measures the rotational displacement around an axis. Most legal *FIRST* Tech Challenge motors contain a built in quadrature encoder that is compatible with a REV Hub. It is also possible to use a standalone incremental encoder like a REV Through Bore Encoder (shown above). An incremental encoder works by sending out a “tick” per partial rotation of the shaft. More information on how many ticks are output per rotation can be found on the manufacturer’s website. An absolute encoder is able to indicate the displacement of the shaft from its starting position and the the exact angle the shaft is currently at relative to a “zero” position.

Additional Resources

- *Encoder Port Overview*

18.2.5 Servos

REV Smart Robot Servo (SRS)

REV-41-1097

goBILDA Dual Mode Servo

GB 2000-0025-0504

Hitec Conventional Servo

W39197

A servo is a type of device that takes a Pulse-Width Modulated (PWM) signal as an input and, with the help of an embedded controller, produces linear or rotational movement based upon the input signal. Servos may take an input signal generated by a REV Hub (either by a Control Hub or Expansion Hub) which itself provides 5V of power and a limited amount of current (see REV Documentation for more information). A REV Servo Power Module (SPM) may be utilized to boost the power provided to servos to a maximum of 90W at 6V for up to 6 servos per device. Robots in *FIRST* Tech Challenge may employ up to 12 total servos.


REV
robotics
REV-41-1097


BILDA
DUAL MODE SERVO
TURN
Super Speed
BILDA
DREAM • MAKE • REPEAT
2000-0025-0504
DUAL MODE
TURN
Super Speed


DELUXE
HiTEC
HS-485HB

Additional Resources

- *Port Overview*
- *Connecting Servos*
- *Configuring Servos*
- *REV Servo Power Module*

18.2.6 Sensors

Note: As per *Game Manual Part 1* a UVC Webcam is not considered a sensor.

Listed below are some examples of common robot sensors. This is not intended to limit or extend in any way the scope of sensors as established in <RE12>. While the *FIRST* Tech Challenge SDK supports many sensors not all are natively supported.

Examples

Distance Sensor (Ultrasonic)

MaxBotix I2C Ultrasonic Sensor

MB1242

An Ultrasonic Distance Sensor is a device that is able to measure the distance between an object and the sensor. It does this by sending out a sound wave and measuring the time it takes for the wave to travel to the object and back. Using this and the speed of sound the distance can be calculated.

Distance Sensor (Optical)

REV 2m Distance Sensor

REV-31-1505

An Optical Time of Flight (ToF) Sensor is a device that is able to measure the distance between an object and the sensor. It does this by sending out a light beam and measuring the time it takes for the beam to travel to the object and back. Using this time and the known speed of light the distance can be calculated. Be aware that the way the object in question interacts with light can change the accuracy of the distance measurement. A transparent object like field panels will often provide inaccurate measurements.

Color Sensor

REV Color Sensor

REV-31-1557

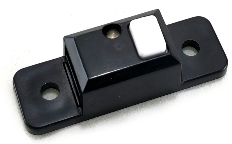
Modern Robotics Color Sensor

MR 45-2018

A color sensor is usually a digital output device that is able to measure the color of an object. Most color sensors require the object in question to be relatively close to the sensor.

Touch Sensor

REV Touch Sensor



REV-31-1425

A touch sensor is a digital output device that detects the activation of a button. This can be used as a limit switch, a way to limit the range of motion of a mechanism. Such a device would typically use the digital port.

Magnetic Limit Switch

REV Magnetic Limit Switch

REV-31-1462

A Magnetic Limit Switch is used to detect the presence of a magnet in near proximity. This is commonly used to limit the range of movement of a mechanism that would cause damage if it went beyond said limit. This is done by placing a magnet on said mechanism which would cause the Limit Switch to activate. It is important to note that as a digital device this will only send out a boolean output and not a range. For measuring the strength of a magnetic field take a look at a magnetometer.

IMU

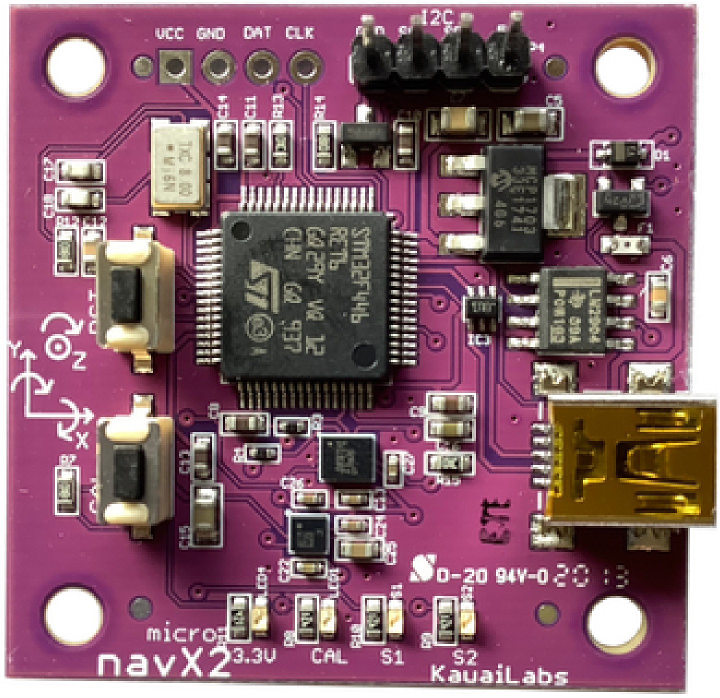
Navigation Sensor

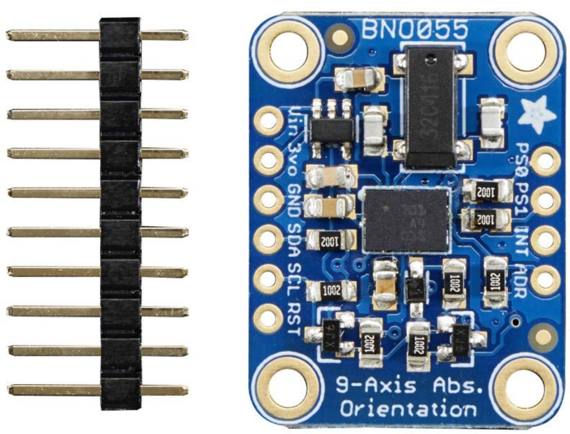
navX2-Micro

BNO055

BNO055

An Interial Measurement Unit (IMU) is a sensor that is a combination of a Gyroscope, Accelerometer, and Magnetometer. A Gyroscope is a device that reports the angular orientation of an object in 3 dimensions. An Accelerometer is a device that


VCC GND DAT CLK
I2C
micro
navX2
3.3V
CAL
S1
S2
KauaiLabs


BNO055
Vin
3vo
GND
SDA
SCL
RST
PS0
PS1
INT
ADR
9-Axis Abs.
Orientation

reports the acceleration of an object in 3 dimensions. Acceleration can be thought of as the rate of change of speed at any given instant. A Magnetometer is a device that measures the strength of magnetic fields in 3 axes. This can be used as a compass to gain the orientation of a robot relative to the poles of the Earth, an absolute measurement.

Potentiometer

REV Potentiometer

REV-31-1155

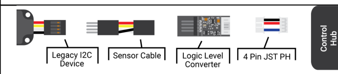
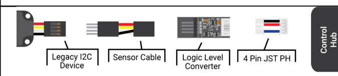
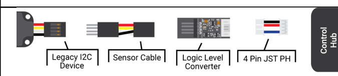
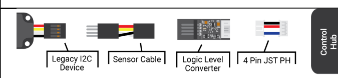
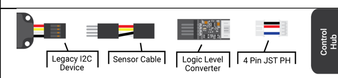
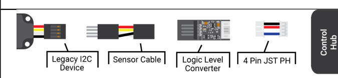
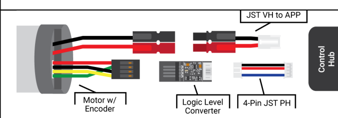
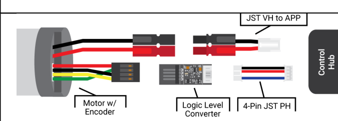
50k Ohm Potentiometer

50k Ohm Potentiometer

A Potentiometer is a device that changes the output voltage based upon the degree to which the adjuster is turned. It is often used as a form of measuring the absolute orientation of an axle. The manner in which the output voltage changes is based on the Potentiometer that is used. Such a device is typically attatched via the analog port of the REV Hub.

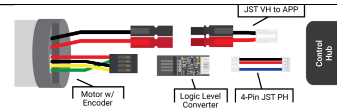
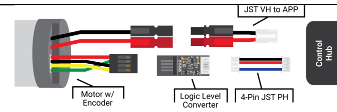
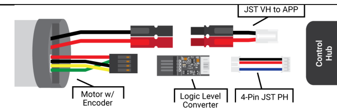
Sensor Compatibility Chart

Thanks to the folks at REV Robotics for providing this handy chart of sensor compatibility.

Sensor	Type	Compatible	Adapters Needed
Absolute Orientation IMU Fusion Breakout - BNO0552472Adafruit	I2C	Yes	3.3V Compatible Custom Wiring Harness Needed
RGB Color Sensor with IR filter and White LED - TCS347251334AdaFruit	I2C	Yes	3.3V Compatible Custom Wiring Harness Needed
ColorSensor45-2018Modern Robotics	I2C	Yes	
Compass45-2003Modern Robotics	I2C	Yes	
Integrating Gyro45-2005Modern Robotics	I2C	Yes	
IR Locator 36045-2009Modern Robotics	I2C	Yes	
IR Seeker V345-2017Modern Robotics	I2C	Yes	
Ranger Sensor45-2008Modern Robotics	I2C	Yes	
NeveRest MotorAM-3461, AM-3102, AM-2964a, AM-3103, AM-3104AndyMark	Quad Encoder	Yes	
HD Hex MotorREV-41-1301REV Robotics	Quad Encoder	Yes	Directly Compatible No Custom Adapters Needed
Core Hex MotorREV-41-1301REV Robotics	Quad Encoder	Yes	Directly Compatible No Custom Adapters Needed
12v 4mm Motor Kit50-0119MATRIX	Quad Encoder	Yes	

continues on next page

Table 1 – continued from previous page

Sensor	Type	Compatible	Adapters Needed
12v 6mm Motor Kit50-0120MATRIX	Quad Encoder	Yes	
Standard Motor Kit50-0001MATRIX	Quad Encoder	Yes	
Max Motor Shaft Encoder KitW38000Tetrix	Quad Encoder	Yes	
Limit Switch45-2401Modern Robotics	Digital	Yes	No Adapter Needed Custom Wiring Harness Required.
Rate Gyro45-2004Modern Robotics	Analog	No	Not Officially Supported
Optical Distance Sensor45-2006Modern Robotics	Analog	No	Not Officially Supported
Touch Sensor45-2007Modern Robotics	Analog	Yes	No Adapter Needed Custom Wiring Harness Required
Light Sensor45-2015Modern Robotics	Analog	No	Not Officially Supported
Magnetic Sensor45-2020Modern Robotics	Analog	No	Not Officially Supported

Additional Resources

- *Analog Port Overview*
- *Digital Port Overview*
- *I2C Port Overview*

18.2.7 UVC Webcam

Logitech C270

Logitech C270

Logitech C920

Logitech C920

A webcam is a device that provides visual images of the surrounding environment. For use as part of *FIRST* Tech Challenge teams must use a COTS UVC (USB Video Class) Compatible Camera. This device can be connected directly to the REV Control Hub or to the Robot Control system via a powered USB hub <RE14>. This device is intended to be used in vision related tasks. An example use case of a webcam is reading the state of the barcode after randomization, a vision task in *Freight Frenzy*, 2021-2022.


720p
logitech


HD 1080p
logi

Additional Resources

- *Connecting UVC Camera via Powered USB Hub*
- *Connecting UVC Camera directly to REV Control Hub*
- *USB Port Overview*
- *Vision in FIRST Tech Challenge*

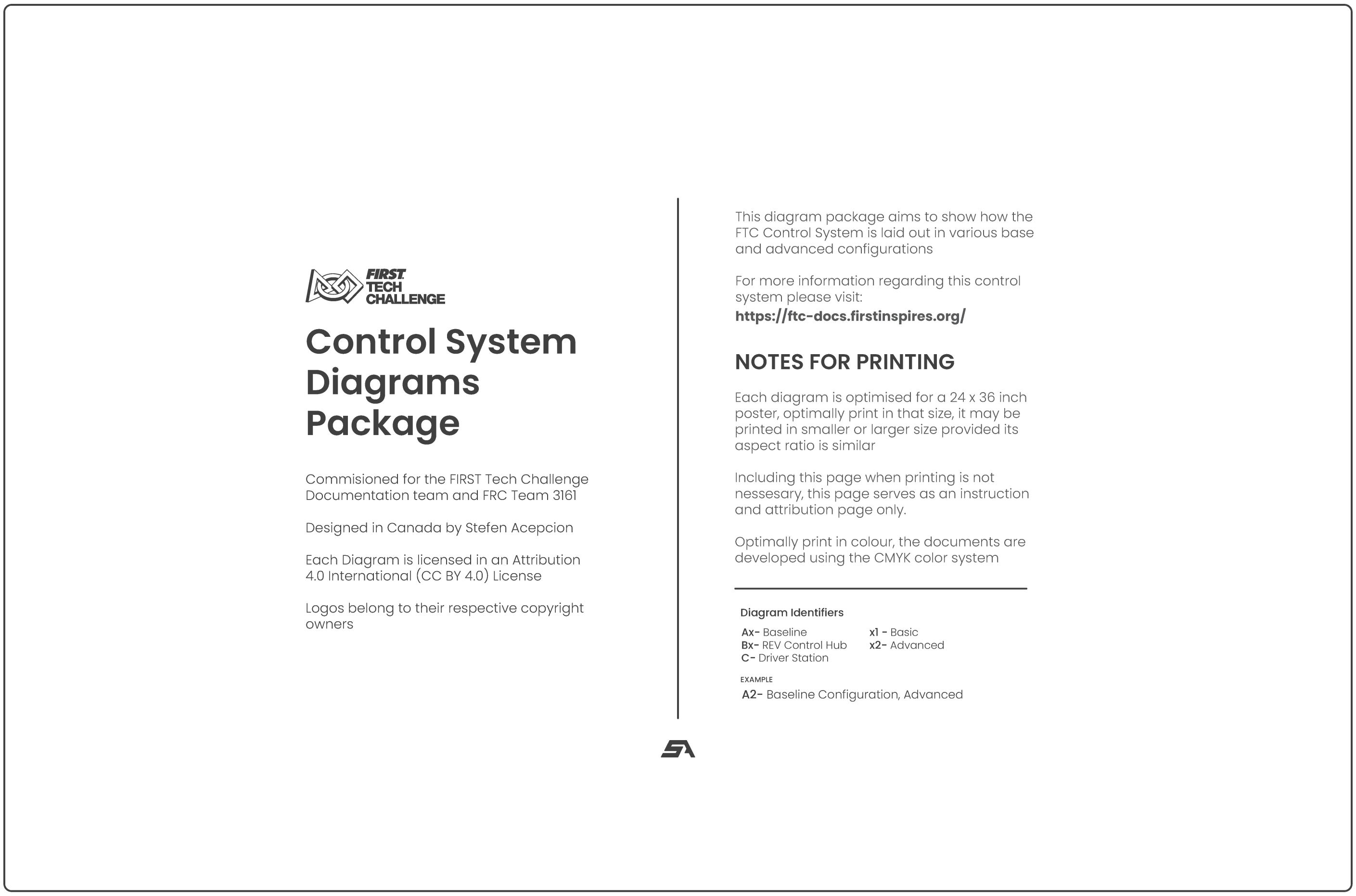


Fig. 7: Diagram courtesy of FRC Team 3161 and Stefen Acepcion

Chapter 19

Hardware and Software Configuration

Hardware and Software Configuration for the Control System

19.1 Connecting Devices To a Control or Expansion Hub

This section explains how to connect a motor, a servo, and some sensors to your REV Robotics Control Hub or REV Robotics Expansion Hub. While the Control Hub differs from the Expansion Hub because of its built in Android device, the layout of the external motor, servo, and sensor ports are identical for the Control Hub and Expansion Hub.

The images in this section use an Expansion Hub to demonstrate how to connect the devices. The process, however, is identical for a Control Hub.

When the instructions in this section use the word "Hub", they are referring to a Control Hub or Expansion Hub.

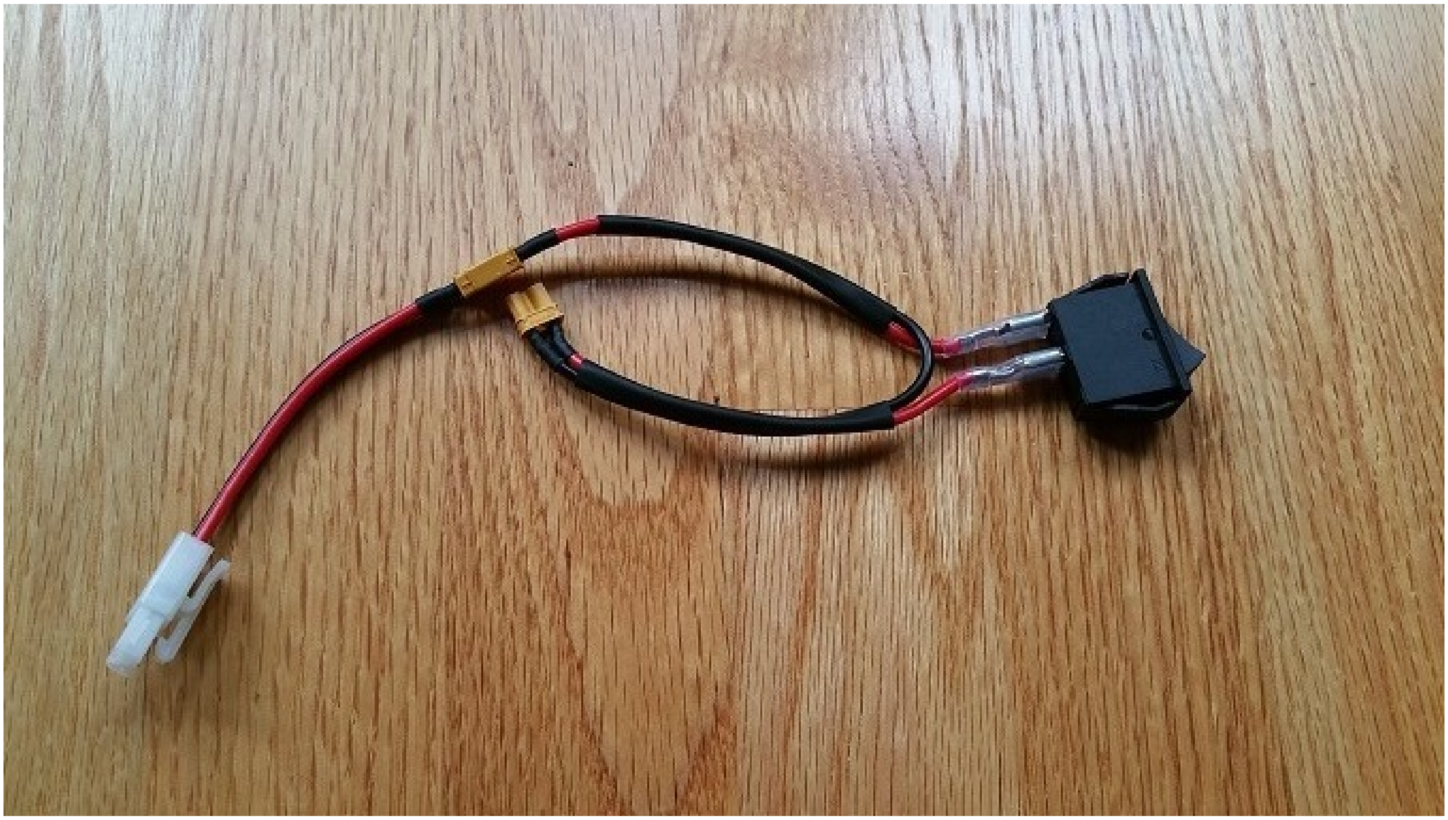
19.1.1 Connecting 12V Power to the Hub

The Hub draws power from a 12V rechargeable battery. For safety reasons, the battery has a 20A fuse built in. A mechanical switch is used to turn on/turn off the power.

Note that it will take an estimated 5 minutes to complete this task.

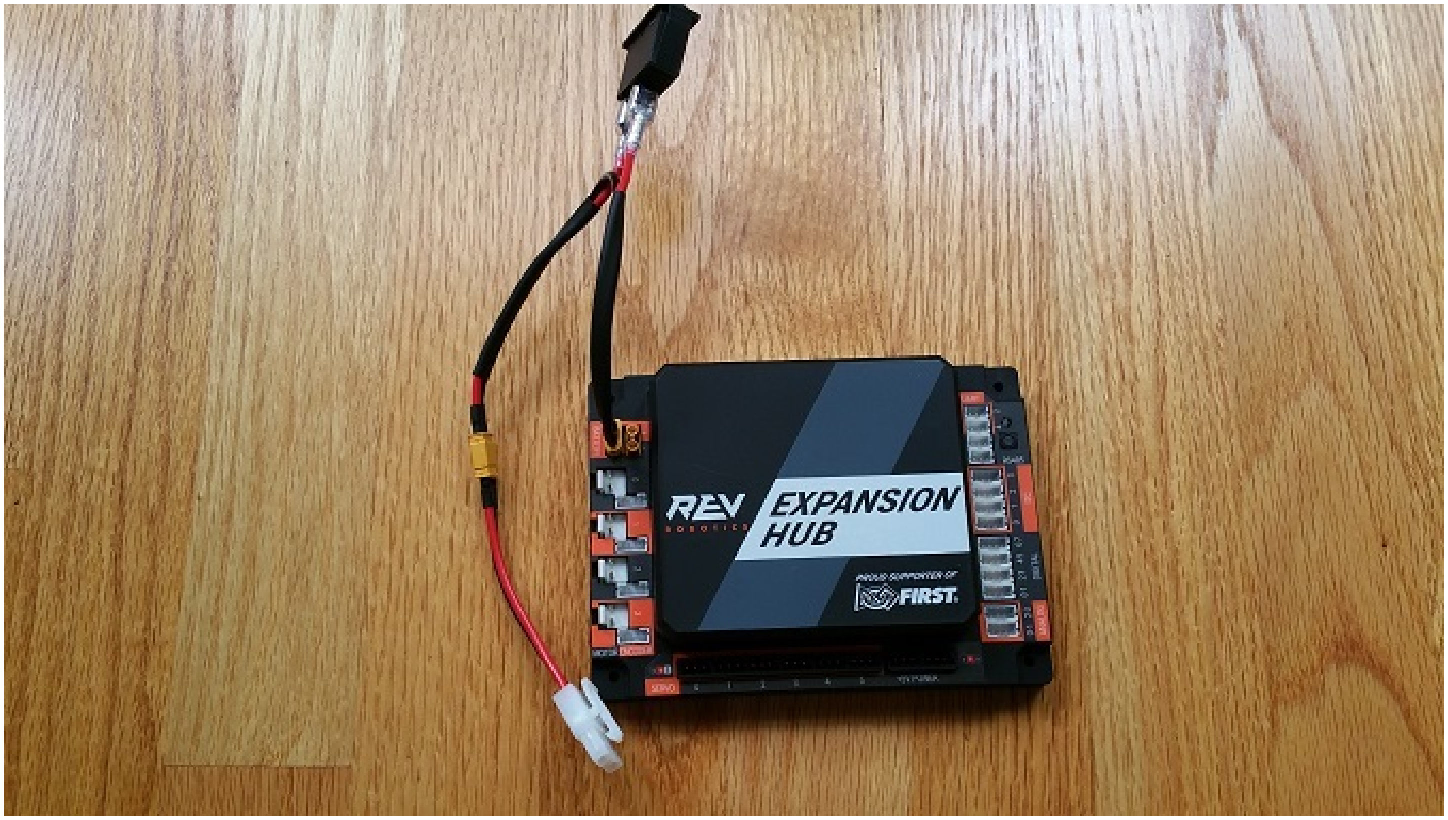
Connecting 12V Power to the Hub Instructions

1. If your 12V battery has a Tamiya style connector, connect the Tamiya to XT30 adapter cable to the matching end of the switch cable.



Note: Do not connect the 12V battery to the Tamiya adapter yet. We will connect the battery during a later step.

2. Connect the other end of the switch cable to a matching XT30 port on the Hub.



3. Verify that the switch is in the OFF position.

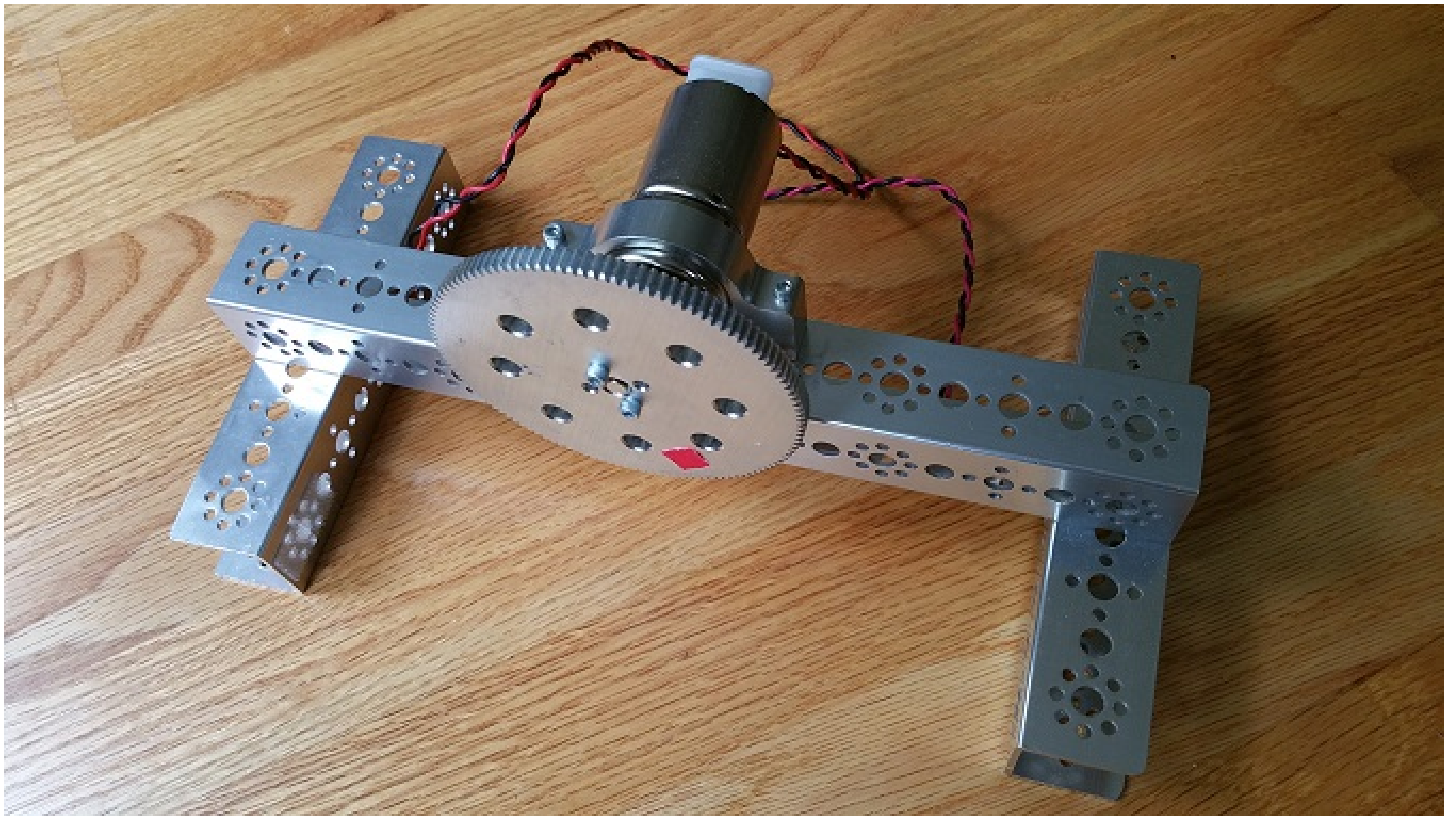
4. Connect the 12V battery to the Tamiya to XT30 cable.

5. Turn on the switch and verify that the Hub is drawing power from the battery. Note that the Hub's LED should be illuminated (notice the blue LED in upper right-hand corner of the Hub in the image below).

6. Turn off the switch and verify that the Hub is off. Note that the Hub’s LED should not be illuminated.

19.1.2 Connecting a Motor to the Hub

The Hub can drive up to four (4) 12V DC motors per Hub. The Hub uses a type of electrical connector known as a 2-pin JST VH connector. Many of the FIRST-approved 12V DC motors are equipped with Anderson Powerpole connectors. An adapter cable can be used to connect the Anderson Powerpole connectors to the Hub motor port (see FIRST Tech Challenge Robot Wiring Guide for more information).

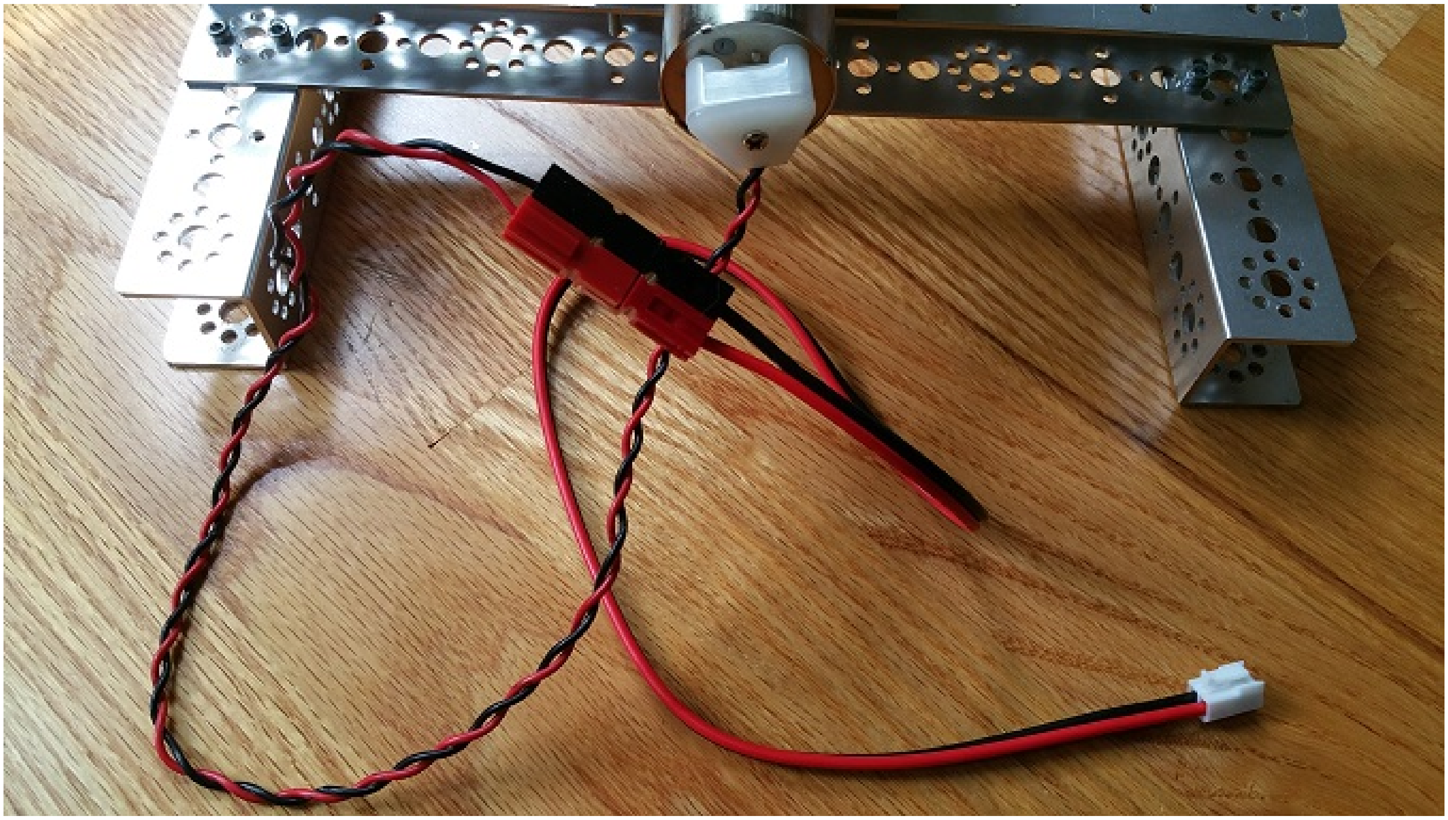


For the examples in this tutorial, *FIRST* recommends that the user build a simple rig to secure the motor in place and prevent it from moving about during the test runs. The image above shows a Tetrix motor installed in a rig built with a Tetrix motor mount and some Tetrix C-channels. A gear was mounted on the motor shaft to make it easier for the user to see the rotation of the shaft.

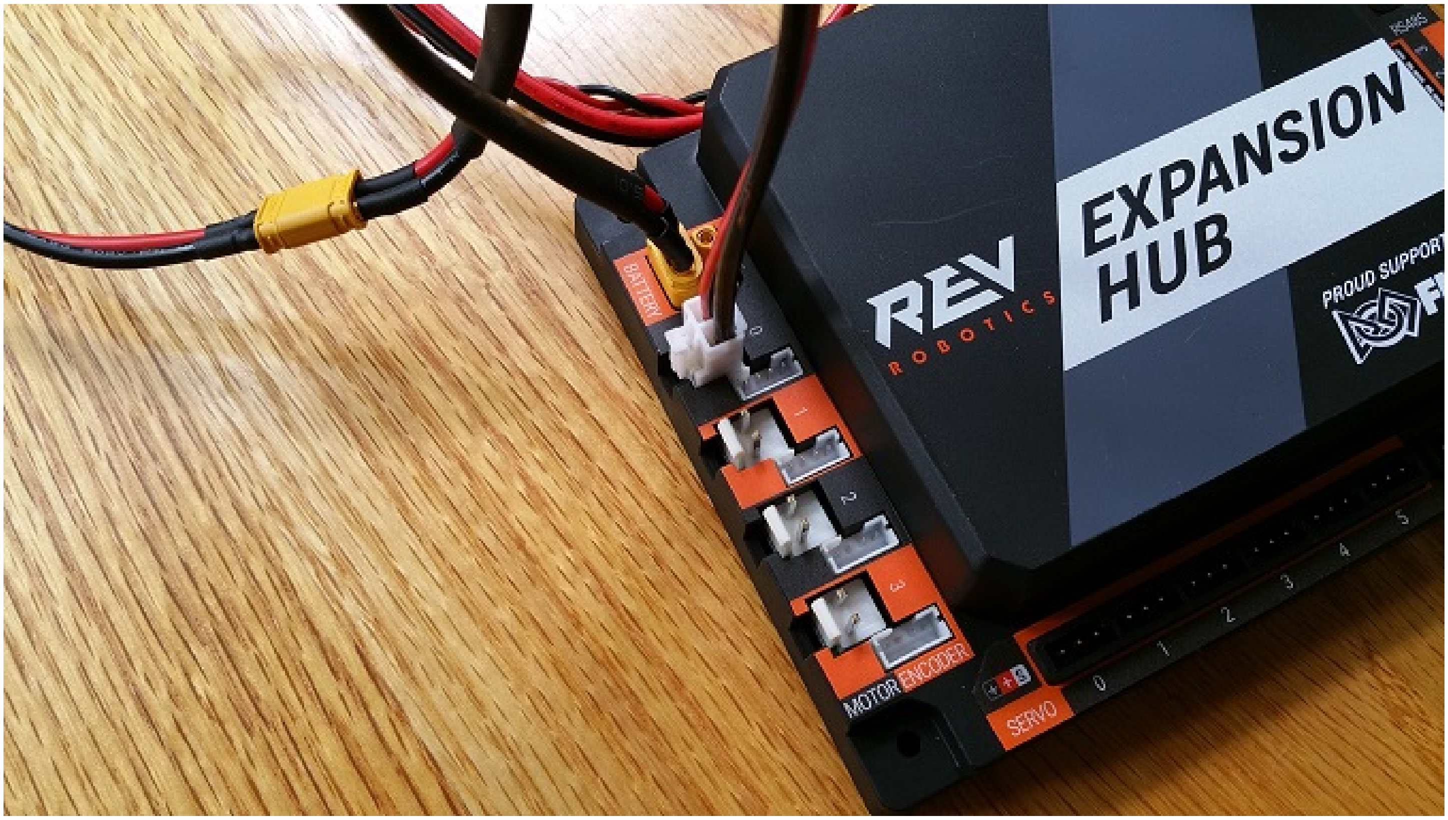
Note that it will take an estimated 2.5 minutes to complete this task.

Connecting a 12V Motor to the Hub Instructions

1. Connect the Anderson Powerpole end of the motor's power cable to the Powerpole end of the Anderson to JST VH adapter cable.



2. Connect the other end of the Anderson to JST VH adapter cable into the motor port labeled “0” on the Hub.



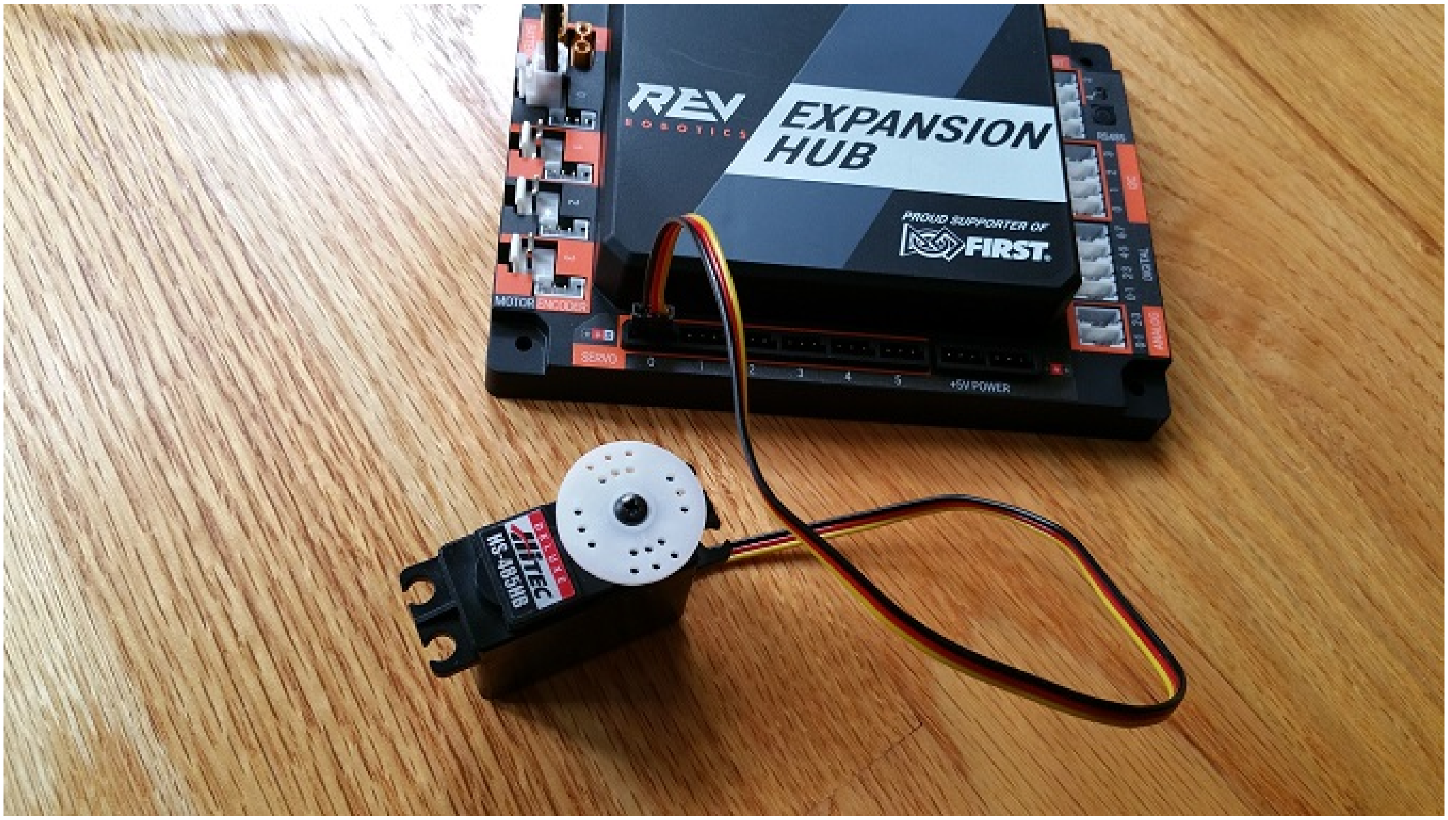
19.1.3 Connecting a Servo to the Hub

The Hub has 6 built-in servo ports. The servo ports accept the standard 3-wire header style connectors commonly found on servos. Note that ground pin is on the left side of the servo port.

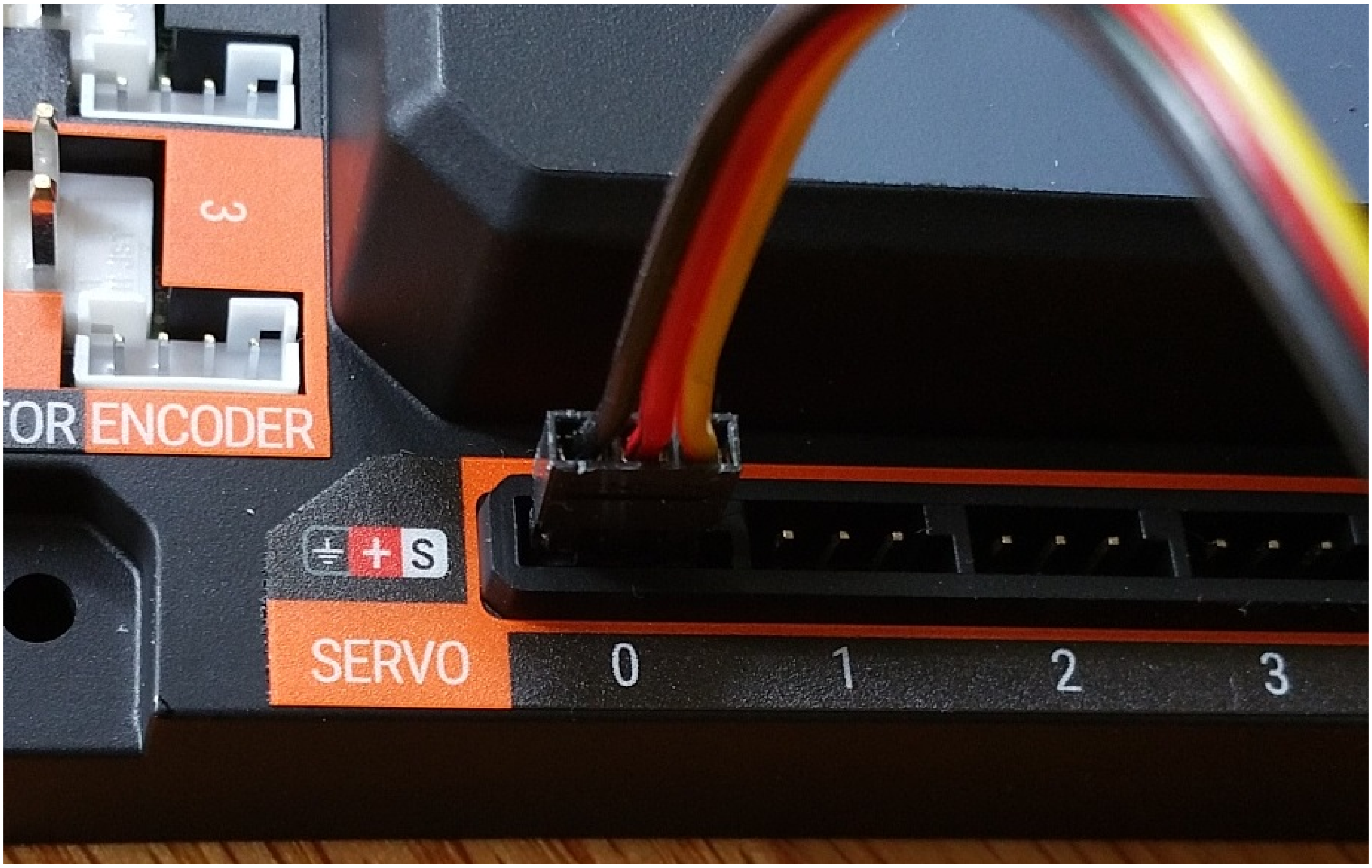
Note that it will take an estimated 2.5 minutes to complete this task.

Connecting a Servo to the Hub Instructions

1. Connect the servo cable to the servo port labeled "0" on the Hub. Note that the ground pin is on the left side of the servo port.



2. Verify that the black ground wire of the servo cable matches the ground pin of the servo port (which is aligned on the left side of the port).



19.1.4 Connecting a Color-Distance Sensor to the Hub

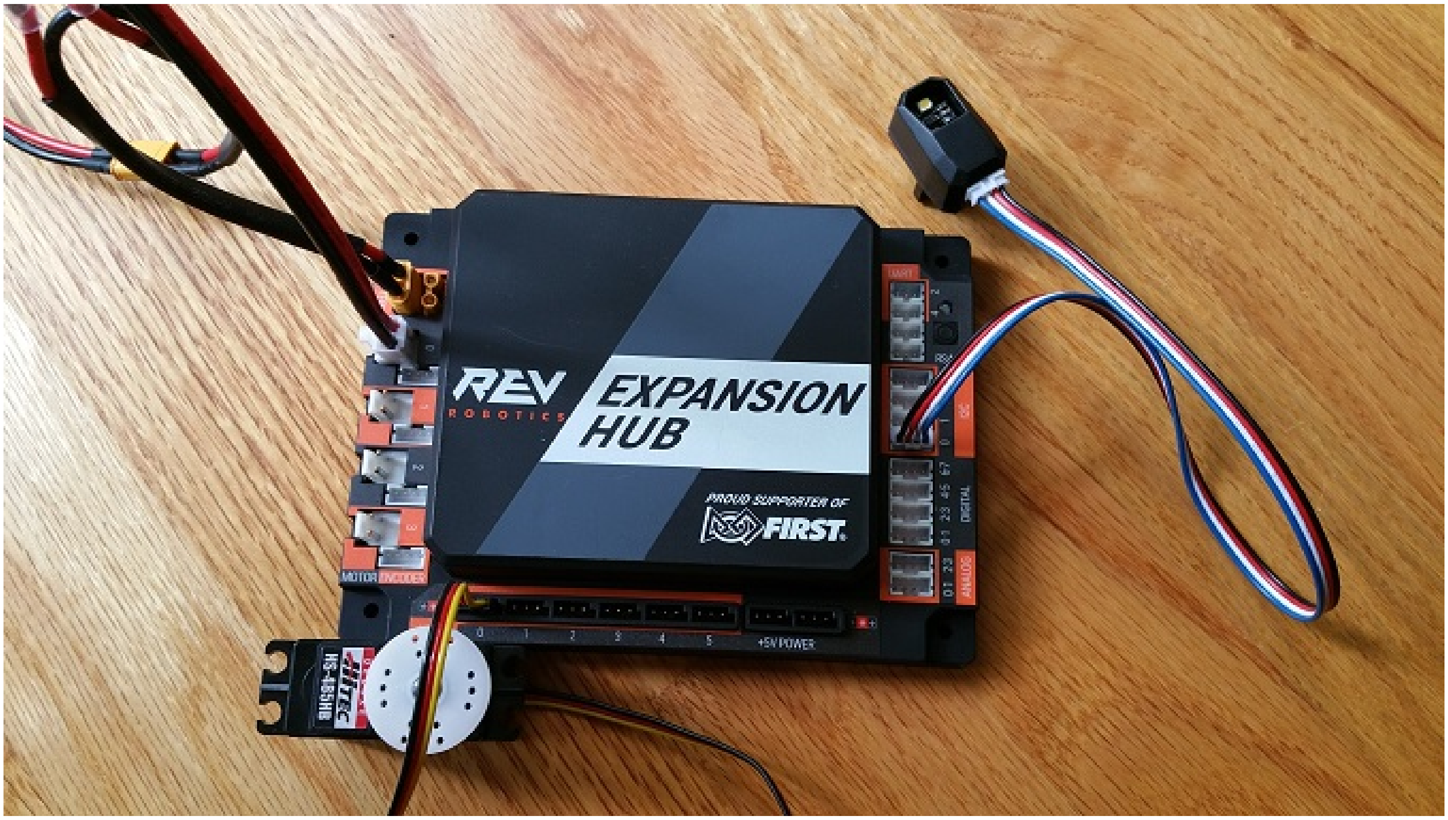
The Hub has 4 independent I2C buses. Each bus has its own port on the Hub. We will connect a REV Robotics Color-Distance sensor to the I2C bus #0 on the Hub.

Note that it will take an estimated 2.5 minutes to complete this task.

Connecting a Color-Distance Sensor to the Hub Instructions

1. Connect one end of the 4-pin JST PH cable to the REV Robotics Color-Distance sensor.

2. Plug the other end of the 4-pin JST PH cable to the I2C port labeled “0” on the Hub.



19.1.5 Connecting a Touch Sensor to the Hub

The Hub has 4 independent digital input/output (I/O) ports. Each port has two digital I/O pins for a total of 8 digital I/O pins on a Hub. You will connect a REV Robotics Touch sensor to one of the digital I/O ports.

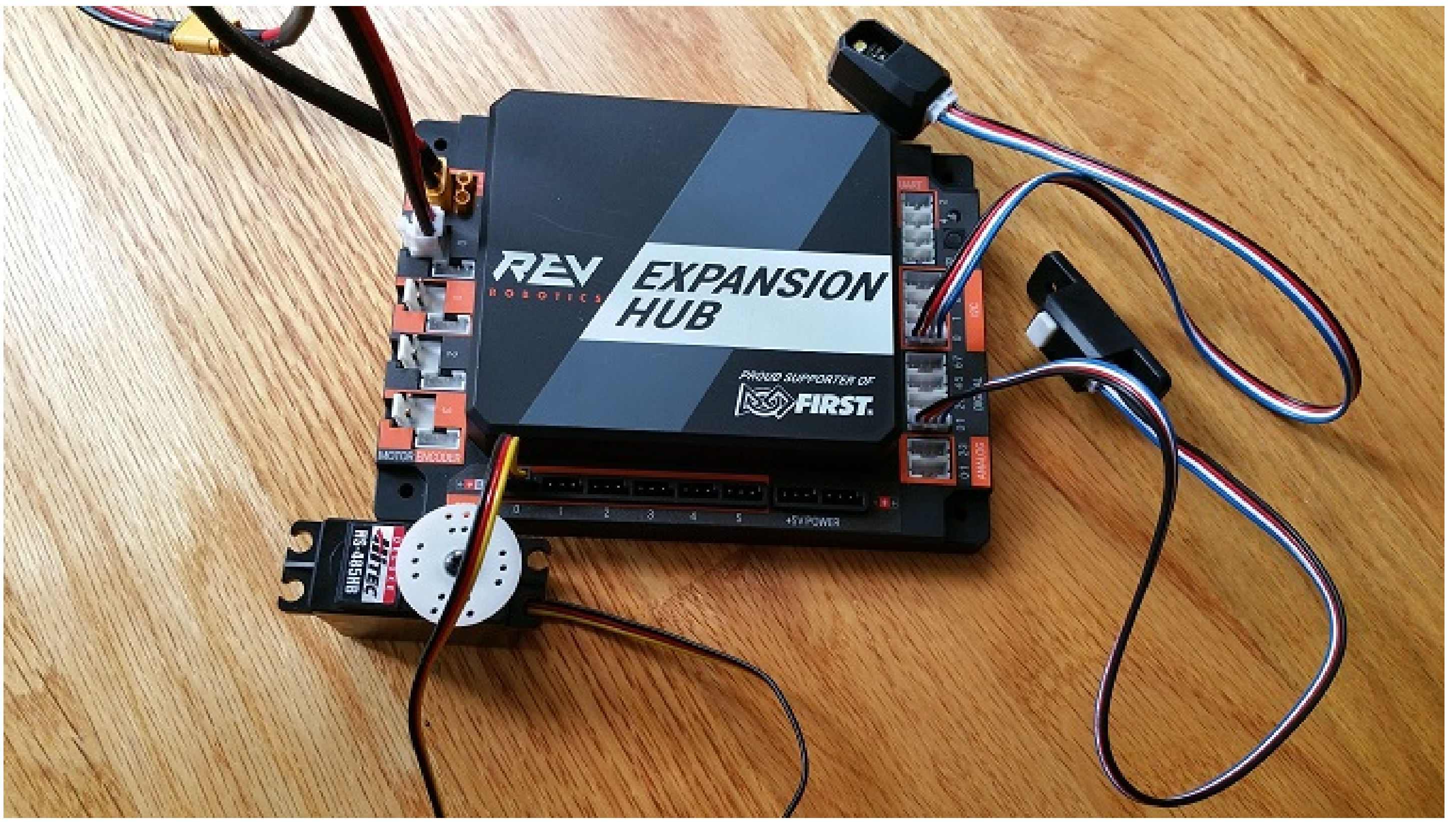
Note that in the case of the REV Robotics Touch Sensor, the device has a connector port for a 4-pin sensor cable. However, the device only needs to connect to one of the two available digital I/O pins. For the REV Robotics Touch Sensor, the second digital I/O pin in the port is the one that gets connected when a standard REV Robotics 4-pin JST PH cable is used. For the "0-1" port, it is the pin labeled "1" that gets connected through the 4-pin cable. Similarly, for the "2-3" port, it is the pin labeled "3" that gets connected through the 4-pin cable.

Note that it will take an estimated 2.5 minutes to complete this task.

Connecting a Touch Sensor to the Hub Instructions

1. Connect one end of the 4-pin JST PH cable to the REV Robotics Touch sensor.

2. Plug the other end of the 4-pin JST PH cable to digital I/O port labeled “0” on the Hub.



19.2 Configuring Your Hardware

This page contains information on configuring your control system hardware such that you may use them in your own projects.

19.2.1 Getting Started

Creating a Configuration

Before you can communicate with the motor, servo and sensors that are connected to the Control Hub or Expansion Hub, you first must create a configuration file on your Robot Controller, so that the Robot Controller will know what hardware is available on the Control Hub's or Expansion Hub's external ports.

Getting the Control Hub Ready

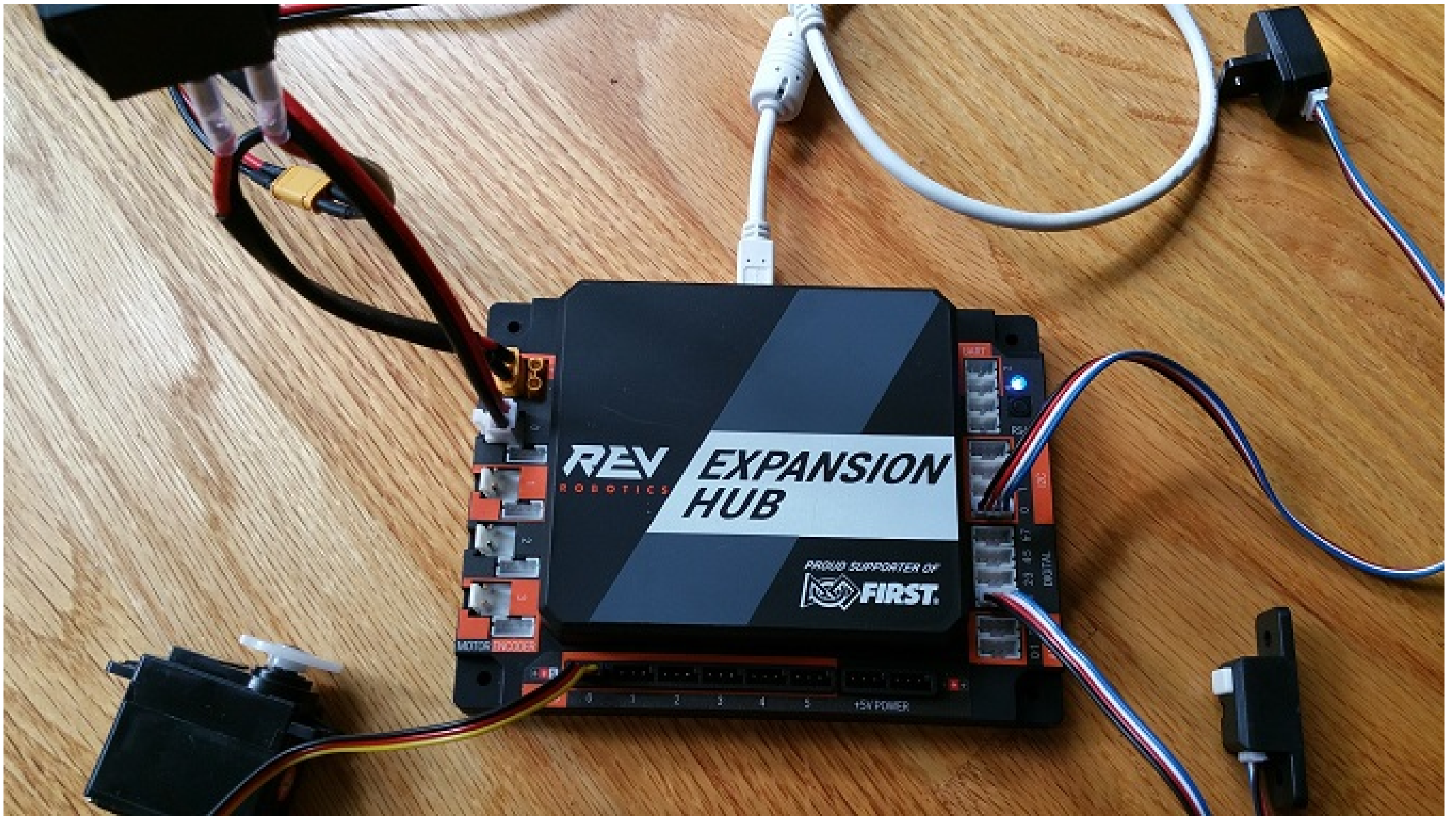
If you are using a Control Hub, you do not need to make any additional connections. You simply need to make sure that the Control Hub is powered on and paired to the DRIVER STATION.

Connecting an Android Smartphone to an Expansion Hub

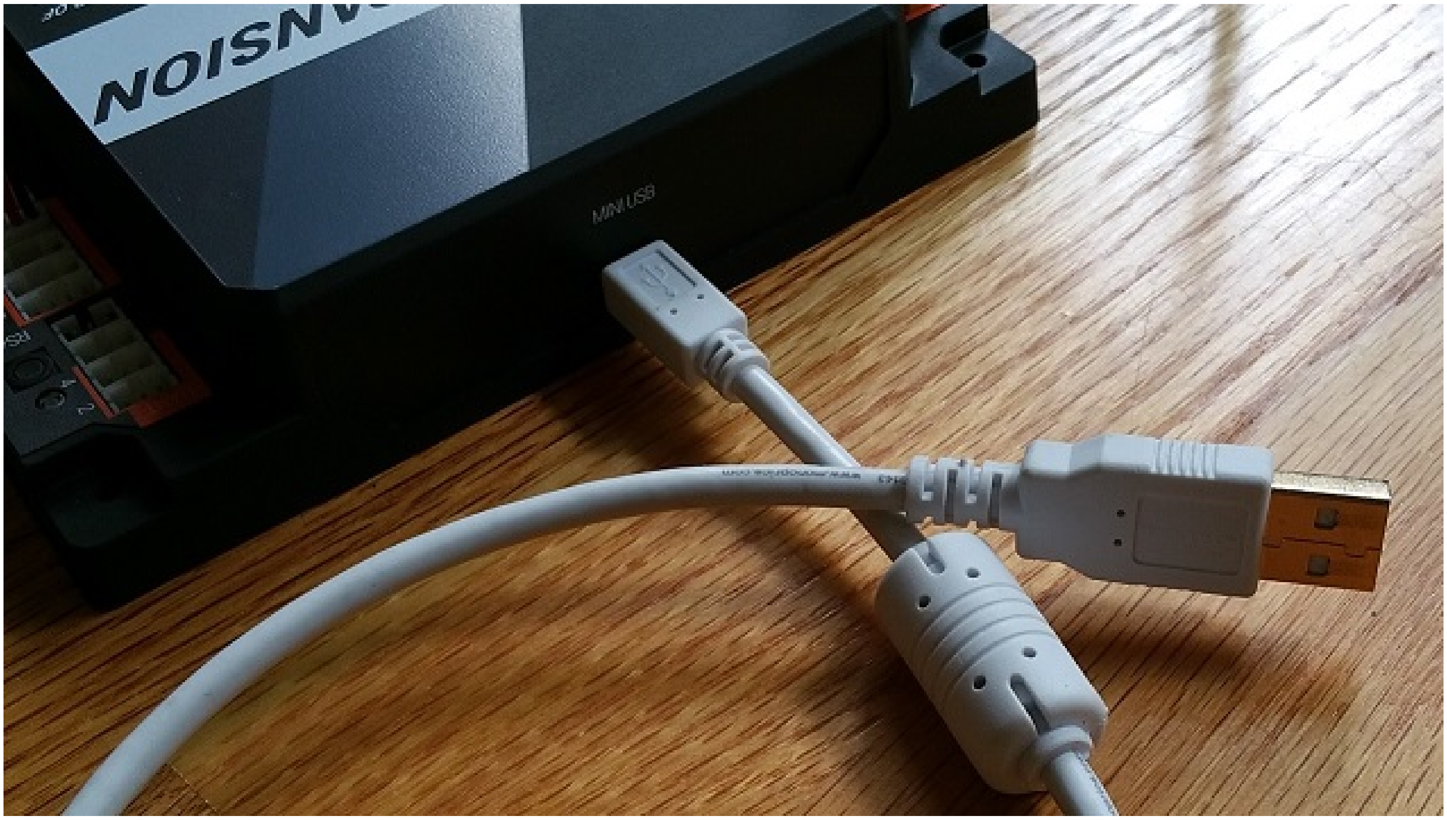
If you are using an Android smartphone as a Robot Controller, you must physically connect the Robot Controller smartphone to the Expansion Hub using a USB cable and an On-The-Go (OTG) adapter. Also, you should verify that the DRIVER STATION is currently paired to the Robot Controller.

Connecting an Android Smartphone to an Expansion Hub Instructions

1. Power on the Expansion Hub by turning on the power switch.

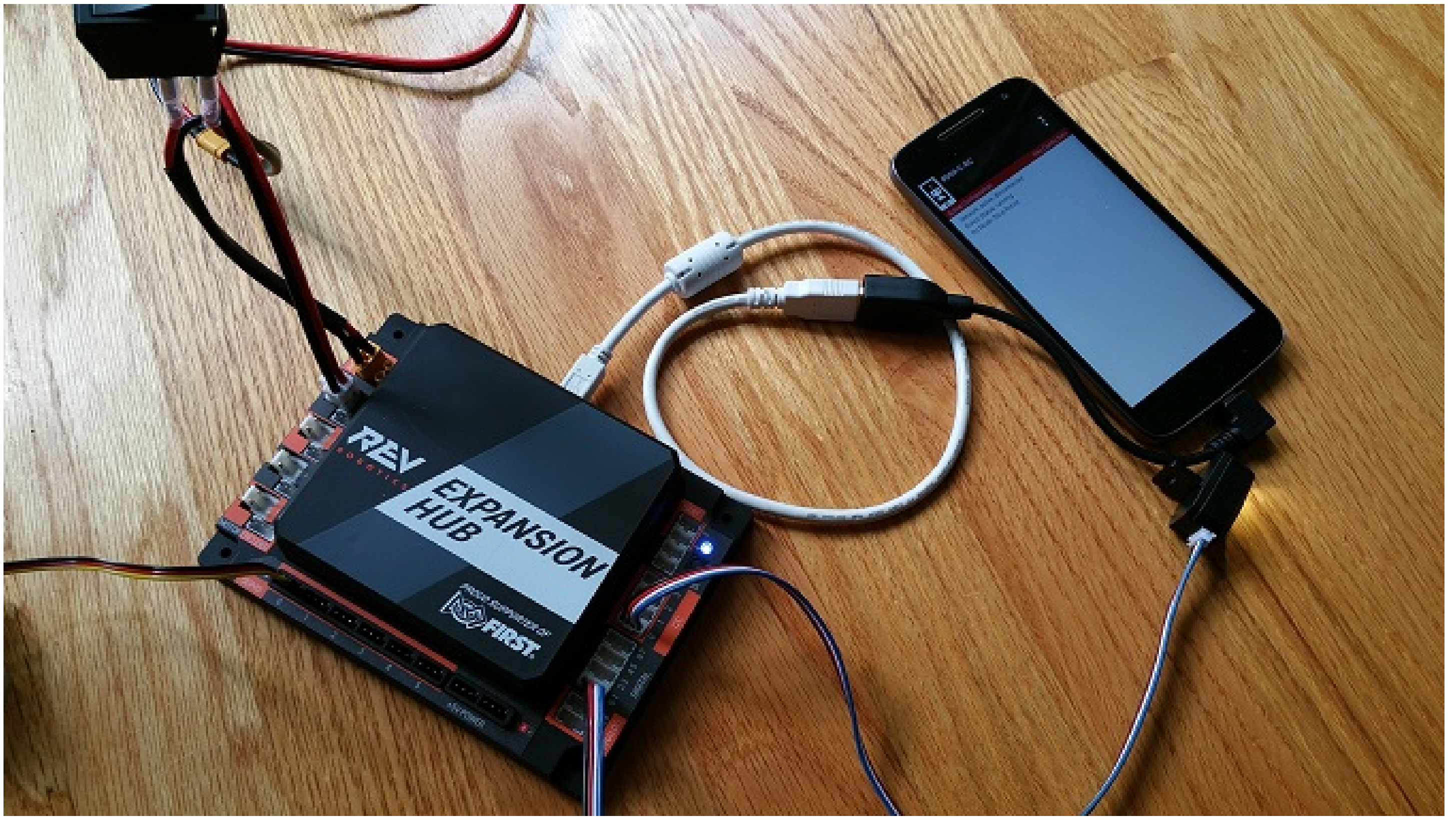


2. Plug the Type B Mini end of the USB cable into the USB mini port on the Expansion Hub.



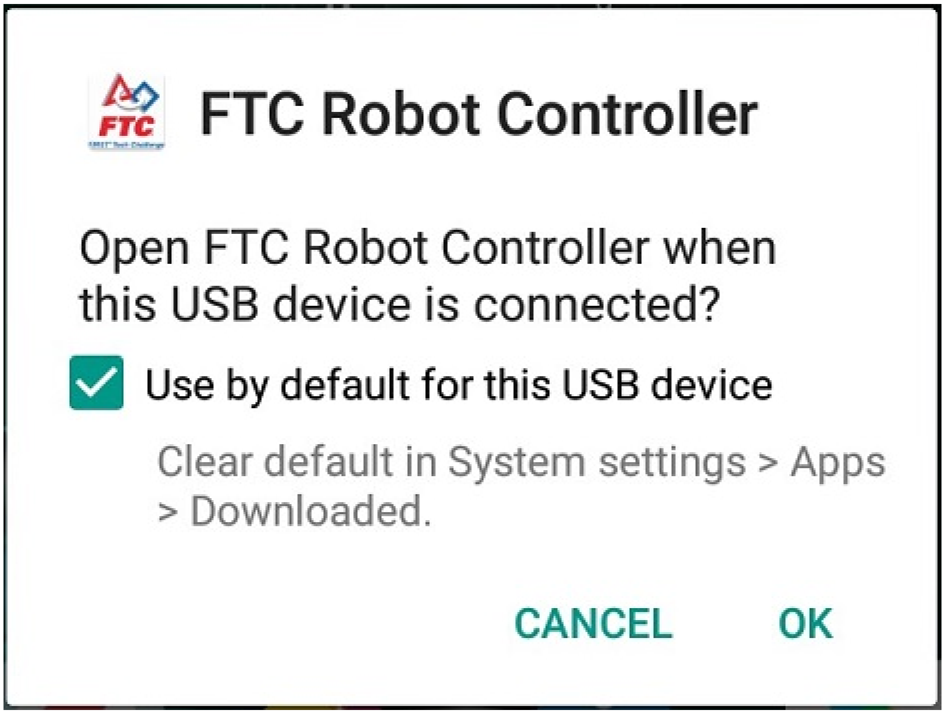
3. Plug the Type A end of the USB cable into the OTG adapter.

4. Verify that your Robot Controller smartphone is powered on and unlocked. Plug in the USB Micro OTG adapter into the OTG port of the Robot Controller phone.



Note that when the OTG adapter is plugged into the smartphone, the phone will detect the presence of the Expansion Hub and launch the Robot Controller app.

5. The first time you connect the Robot Controller smartphone to the Expansion Hub, the Android operating system should prompt you to ask if it is OK to associate the newly detected USB device (which is the Expansion Hub) with the Robot Controller app.



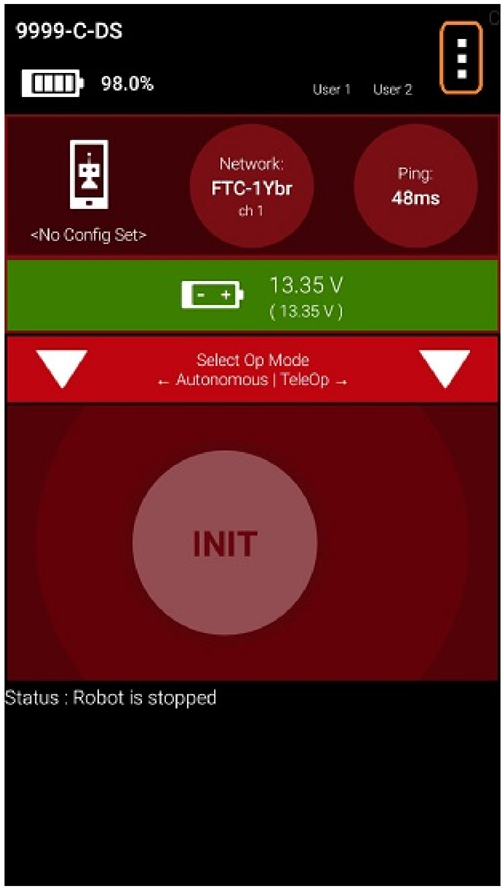
Important: You might be prompted multiple times to associate the USB hardware with the Robot Controller. Whenever you are prompted by your phone with this message, you should always select the "Use by default for this USB device" option and hit the "OK" button to associate the USB device with the Robot Controller app. If you fail to make this association, then the Robot Controller app might not reliably connect to this Expansion Hub the next time you turn your system on.

Creating a Configuration File Using the DRIVER STATION

Although the configuration file needs to reside on the Robot Controller, for this tutorial we will use the DRIVER STATION app to create the configuration file remotely. The DRIVER STATION can be used to create a configuration file for a Control Hub or for an Android smartphone Robot Controller.

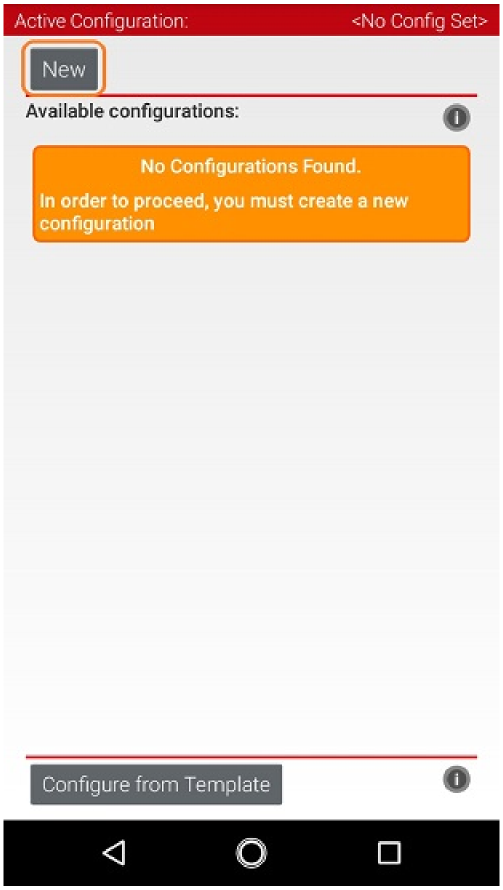
Creating a Configuration File on the Robot Controller using the DRIVER STATION Instructions

1. Touch the three vertical dots in the upper right hand corner of the Driver Station app. This will launch a pop-up menu.



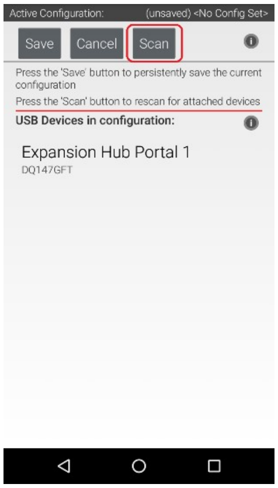
2. Select **Configure Robot** from the pop up menu to display the **Configuration** screen.

3. If your Robot Controller does not have any existing configuration files, the screen will display a message indicating that you need to create a file before proceeding.



Hit the **New** button to create a new configuration file for your Robot Controller.

4. When the new configuration screen appears, the Robot Controller app will do a scan of the serial bus to see what devices are connected to the Robot Controller.

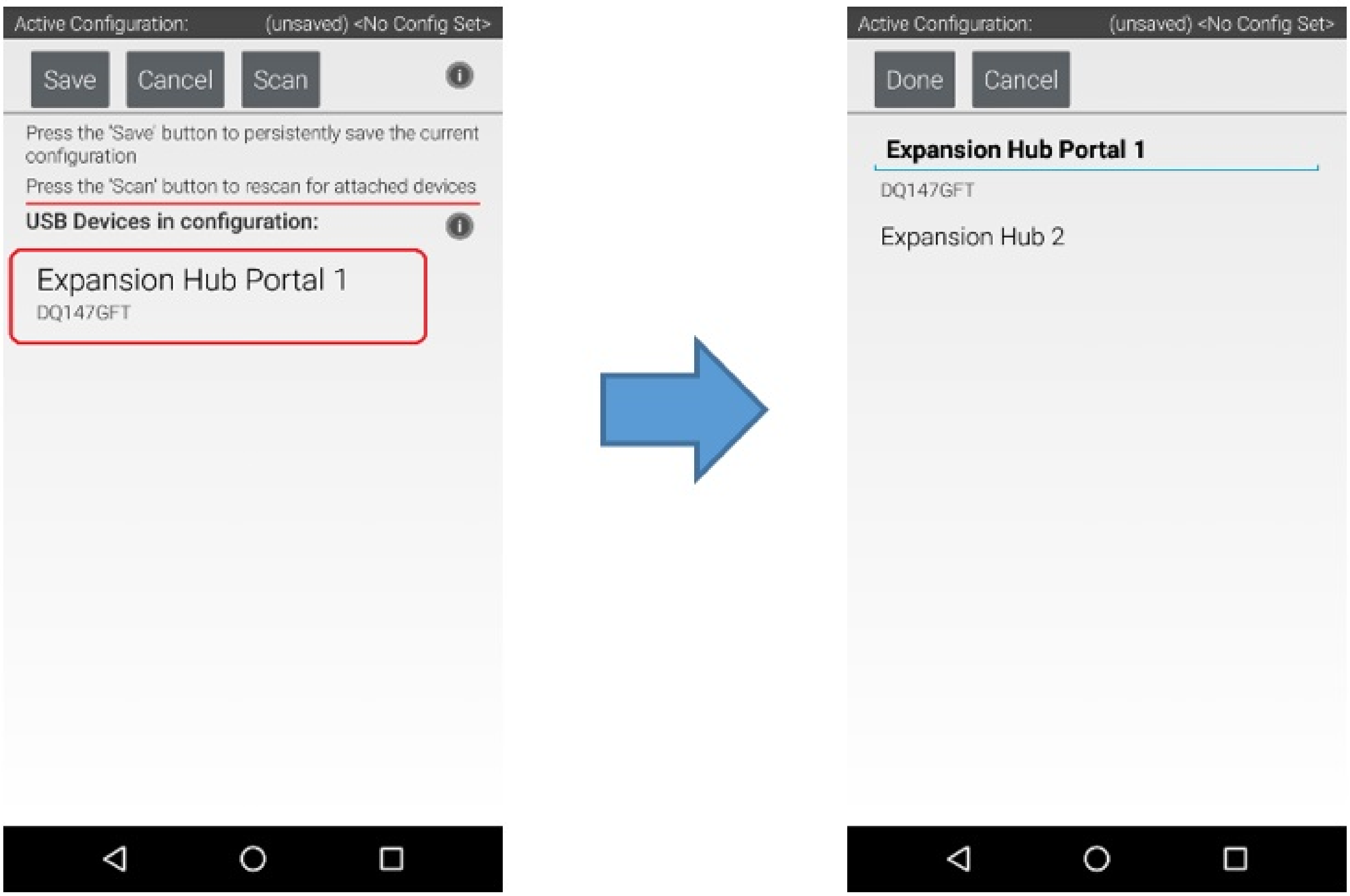


It will display the devices that it found in a list underneath the words "USB Devices in configuration." You should see an entry that says something like "Expansion Hub Portal 1" in the list.

Your Expansion Hub is listed as a Portal because it is directly connected to the Robot Controller phone through the USB cable or in the case of the Control Hub through the internal serial bus.

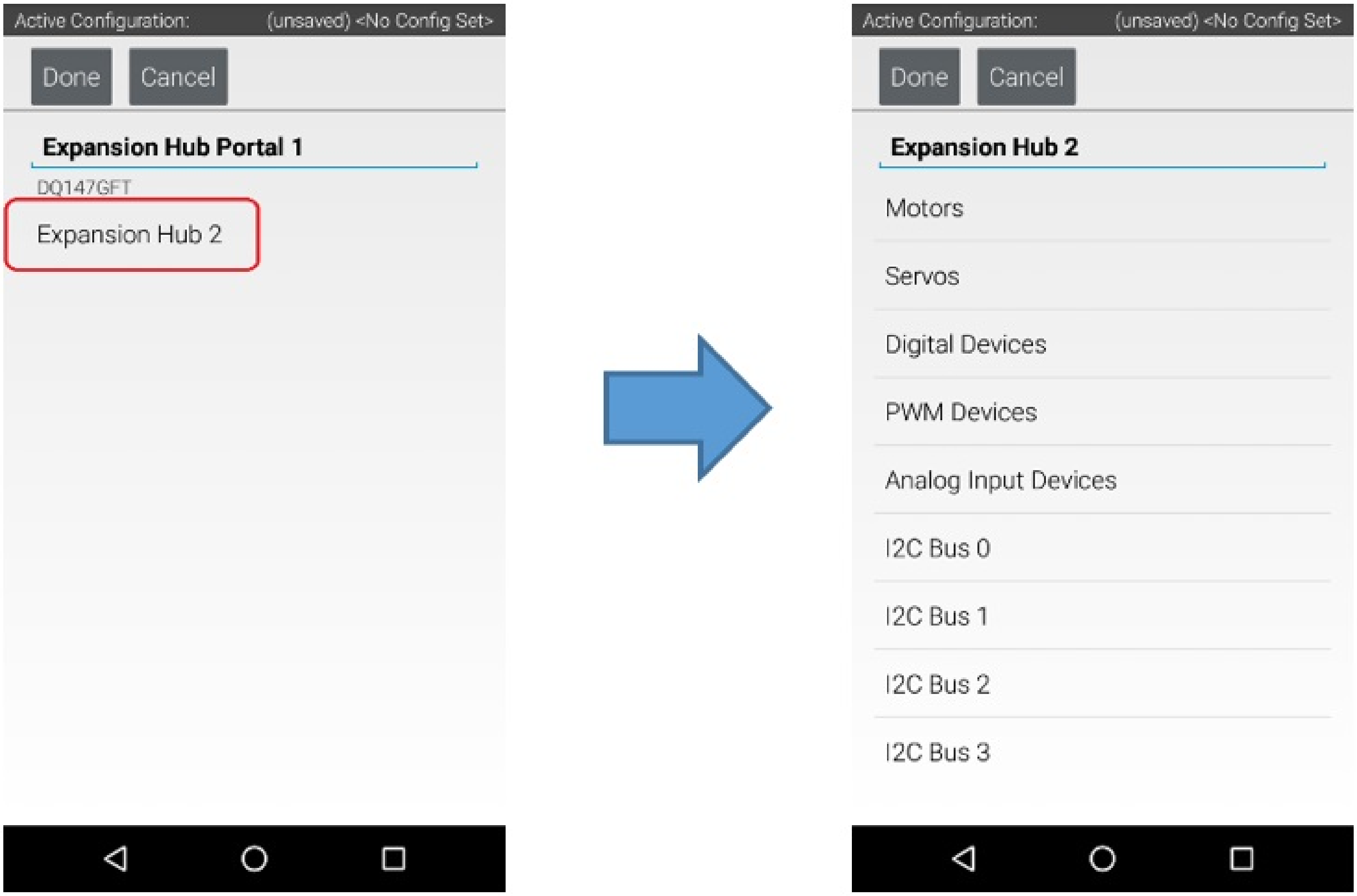
If you do not see your Expansion Hub Portal listed and you are using a smartphone as a Robot Controller, check the wired connections to make sure they are secure and then press the Scan button one or two times more to see if the smartphone detects the device on a re-scan of the USB bus.

5. Touch the Portal listing ("Expansion Hub Portal 1" in this example) to display what Expansion Hubs are connected through this Portal.



Since we only have a single Expansion Hub connected, we should only see a single Expansion Hub configured ("Expansion Hub 2" in this example).

6. Touch the Expansion Hub listing ("Expansion Hub 2" in this example) to display the Input/Output ports for that device.



The screen should change and list all the motor, servo and sensor ports that are available on the selected Expansion Hub.

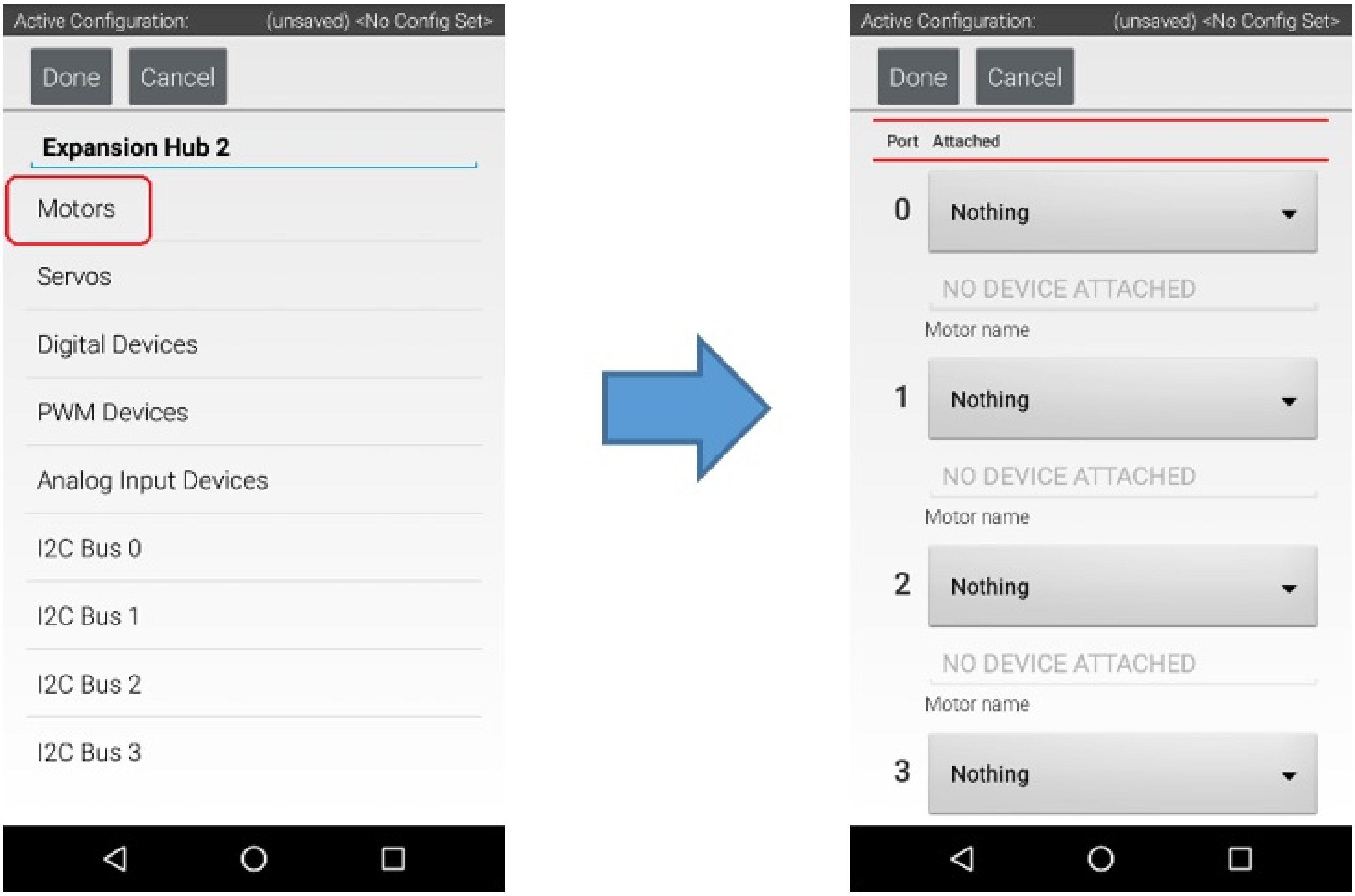
19.2.2 Configuring a DC Motor

Now that you've created a file, you will need to add a DC Motor to the configuration file.

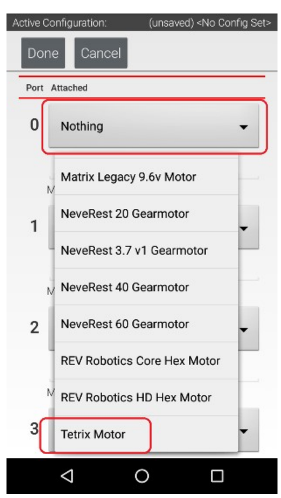
Important: At this point, although you have created your configuration file, you have not yet saved its contents to the Robot Controller. You will save the configuration file later in the *Saving the Configuration Information* step.

Configuring a DC Motor Instructions

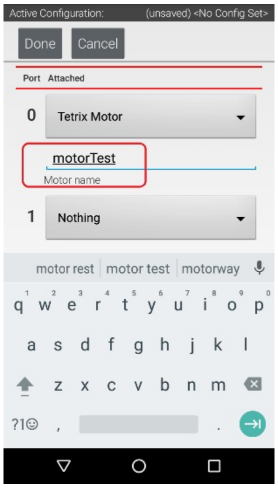
1. Touch the word **Motors** on the screen to display the Motor Configuration screen.



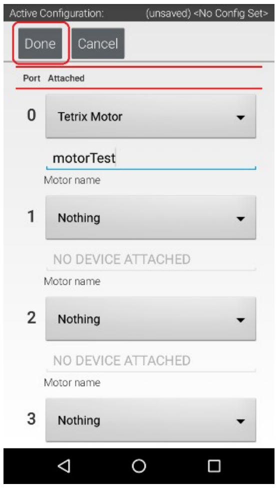
2. Since we installed our motor onto port #0 of the Expansion Hub, use the dropdown control for port 0 to select the motor type (Tetrix Motor for this example).



3. Use the touch screen keypad to specify a name for your motor ("motorTest" in this example).



4. Press the **Done** button to complete the motor configuration. The app should return to the previous screen.

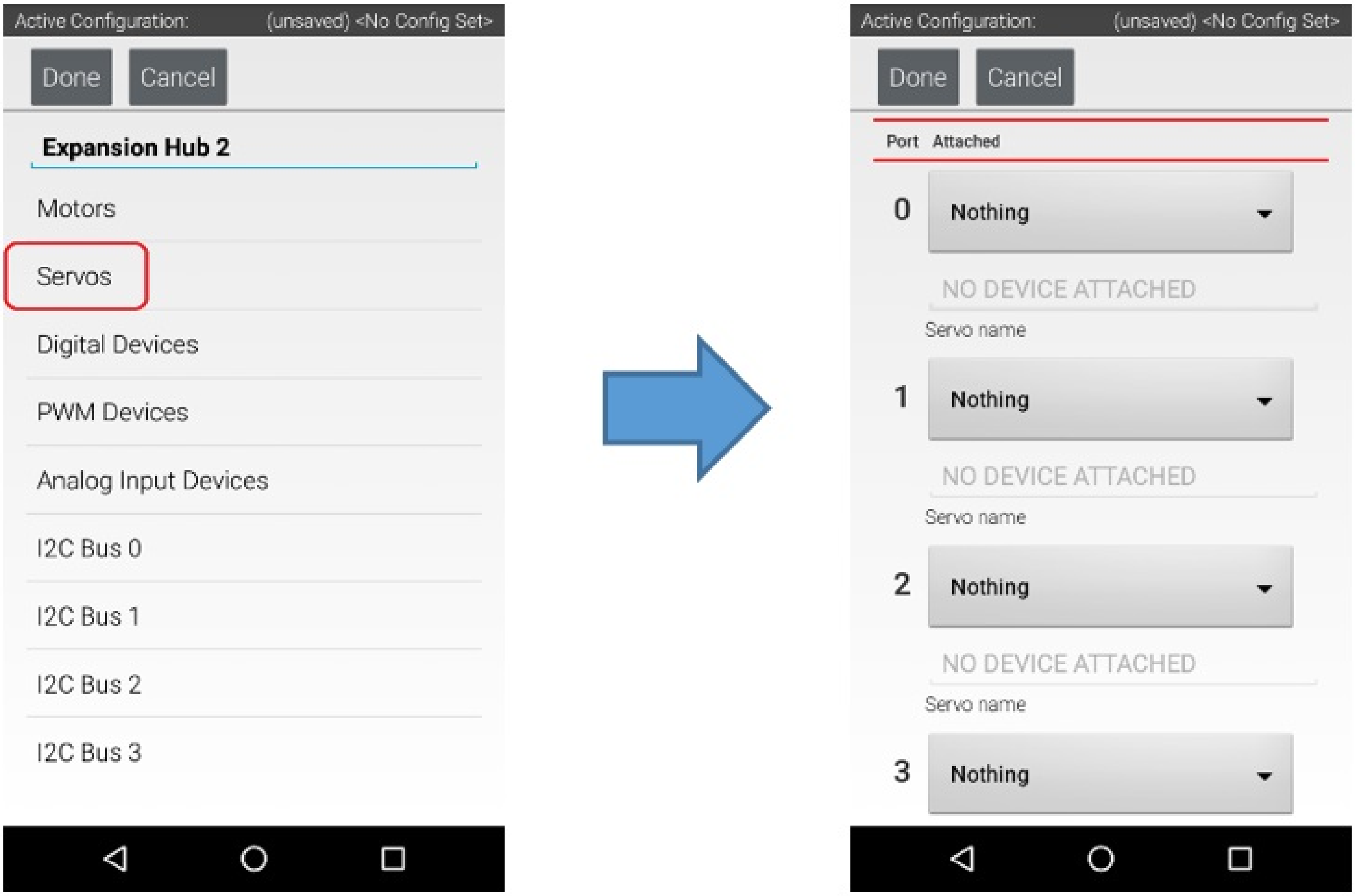


19.2.3 Configuring a Servo

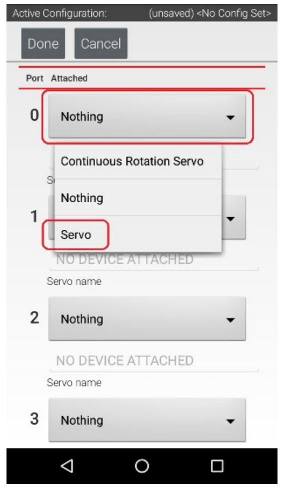
You will also want to add a servo to the configuration file. In this example, you are using a standard 180-degree servo.

Configuring a Servo Instructions

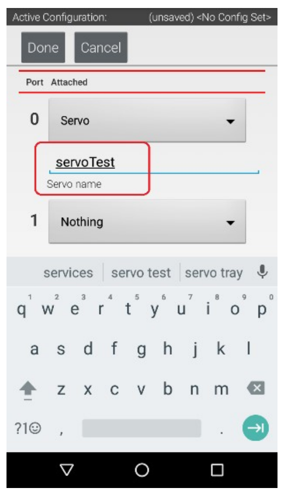
1. Touch on the word **Servos** on the screen to display the **Servo Configuration** screen.



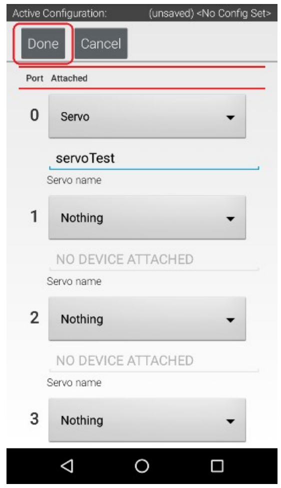
2. Use the dropdown control to select “Servo” as the servo type for port #0.



3. Use the touch pad to specify the name of the servo ("servoTest" for this example) for port #0.



4. Press the **Done** button to complete the servo configuration. The app should return to the previous screen.

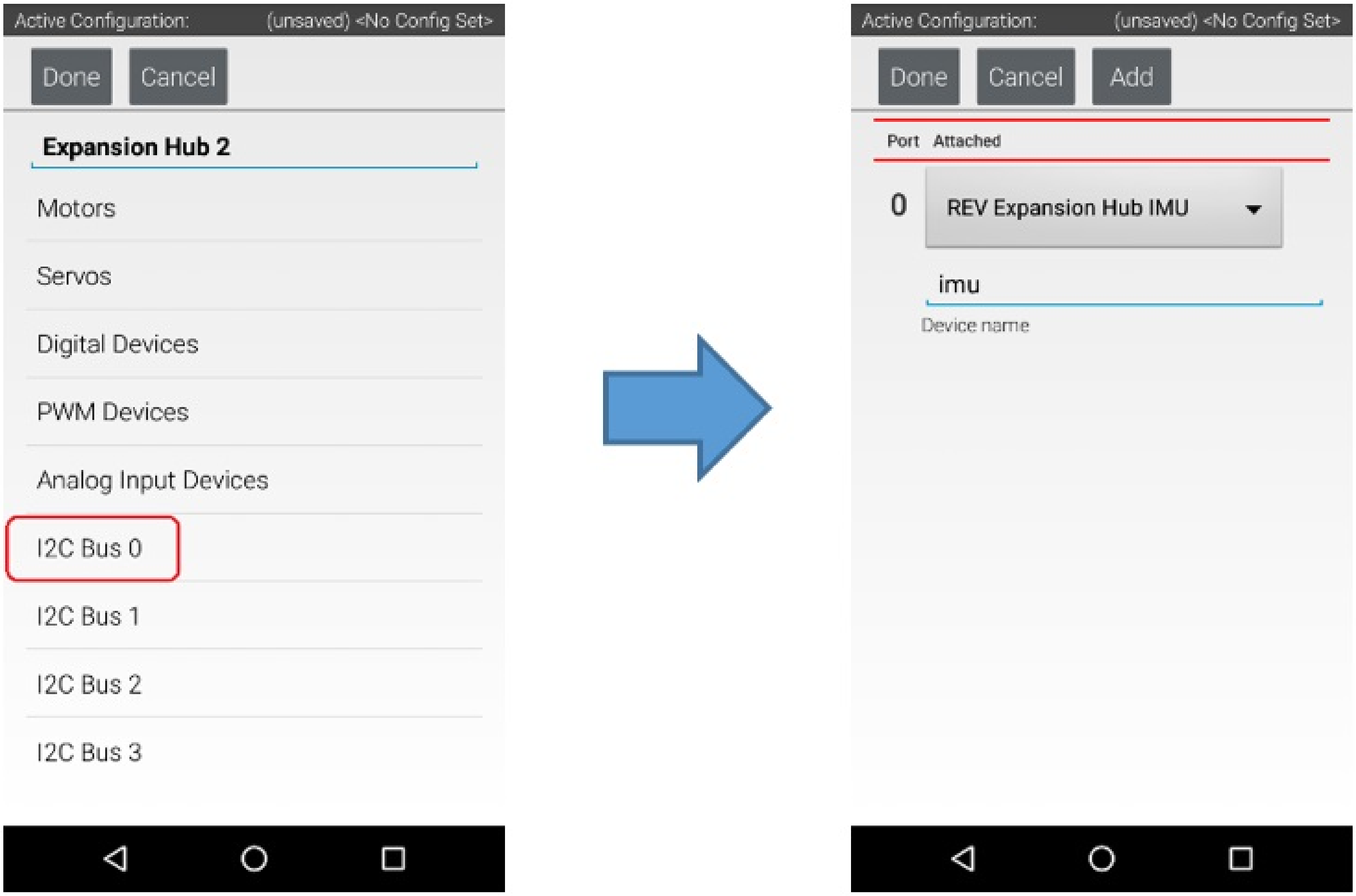


19.2.4 Configuring a Color Distance Sensor

The REV Robotics Color Distance Sensor is an I2C sensor. It actually combines two sensor functions into a single device. It is a color sensor, that can determine the color of an object. It is also a distance or range sensor, that can be used to measure short range distances. Note that in this tutorial, the word “distance” is used interchangeably with the word “range”.

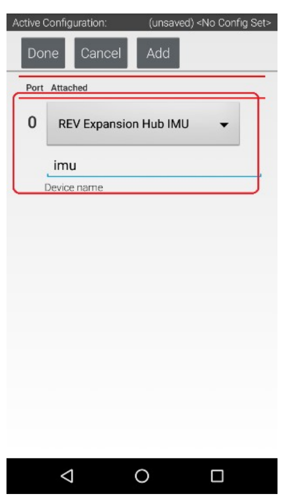
Configuring a Color Distance Sensor Instructions

1. Touch the words **I2C Bus 0** on the screen to launch the I2C configuration screen for this I2C bus.



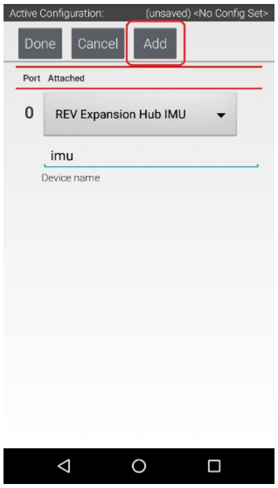
The Control Hub or Expansion Hub has four independent I2C buses, labeled “0” through “3”. In this example, since you connected the Color Sensor to the port labeled “0”, it resides on I2C Bus 0.

2. Look at the **I2C Bus 0** screen. There should already be a sensor configured for this bus. The Control Hub or Expansion Hub has its own built-in inertial measurement unit (IMU) sensor. This sensor can be used to determine the orientation of a robot, as well as measure the accelerations on a robot.

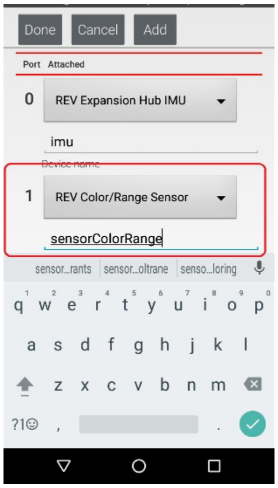


The built-in IMU is internally connected to I2C Bus 0 on each Control Hub or Expansion Hub. Whenever you configure a Control Hub or Expansion Hub using the Robot Controller, the app automatically configures the IMU for I2C Bus 0. You will need to add another I2C device for this bus to be able to configure the color sensor.

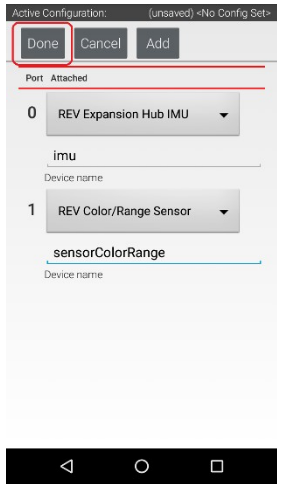
3. Press the **Add** button to add another I2C device to this bus.



4. Select "REV Color/Range Sensor" from the dropdown selector for this new device. Use the touchscreen keyboard to name this device "sensorColorRange".



5. Press the **Done** button to complete the I2C sensor configuration. The app should return to the previous screen.

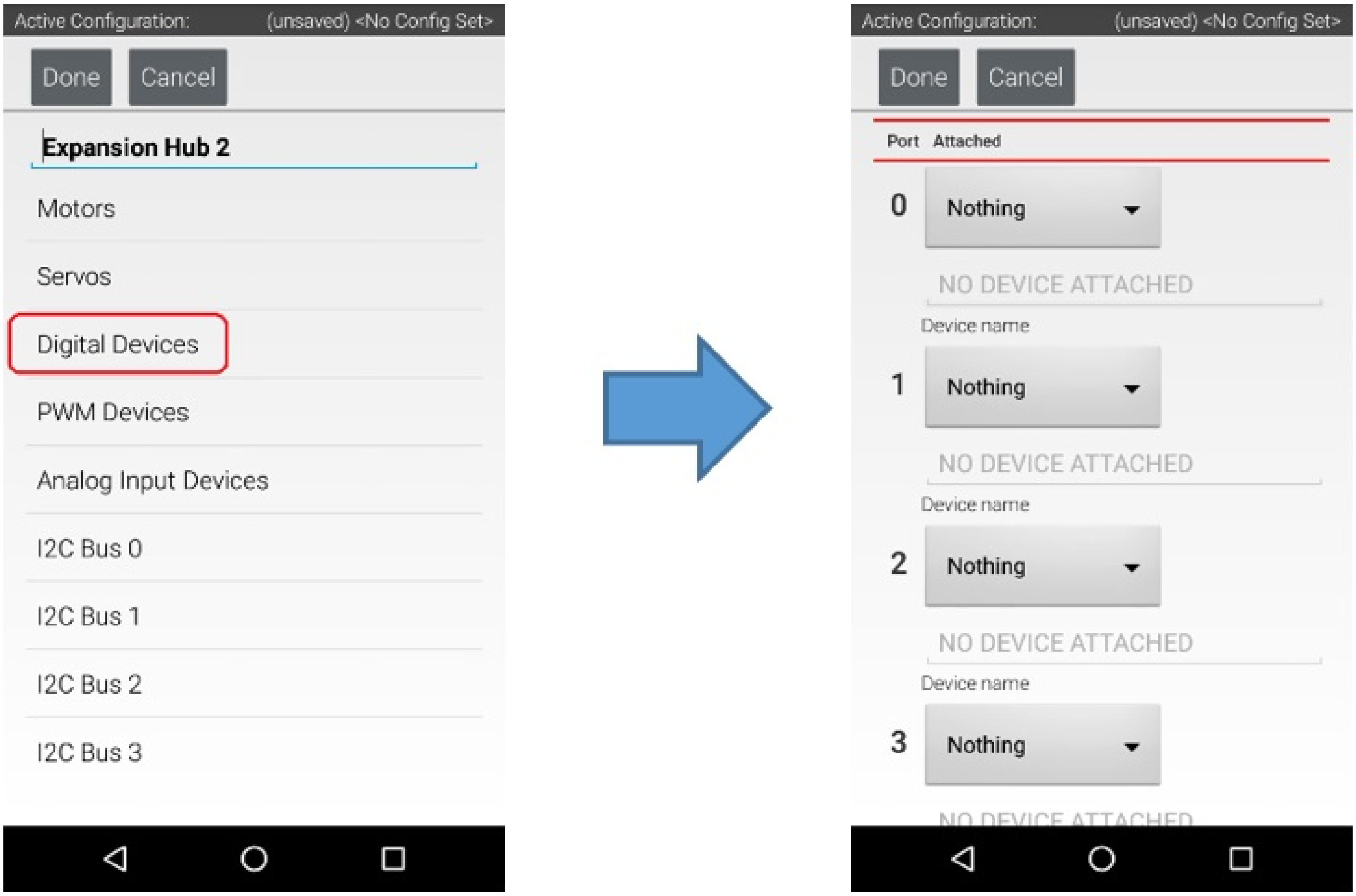


19.2.5 Configuring a Digital Touch Sensor

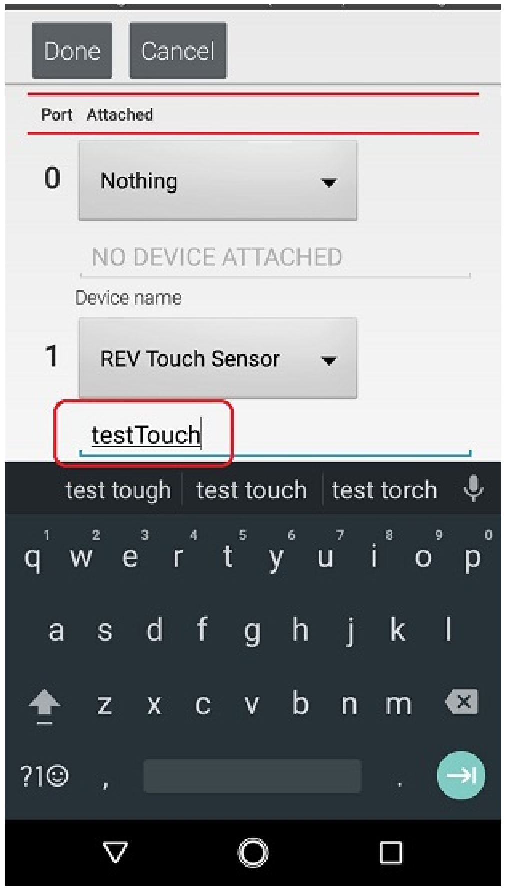
The REV Robotics Touch Sensor is a digital sensor. An Op Mode can query the Touch Sensor to see if its button is being pressed or not.

Configuring a Digital Touch Sensor Instructions

1. Touch the words **Digital Devices** on the screen to launch the Digital I/O configuration screen.

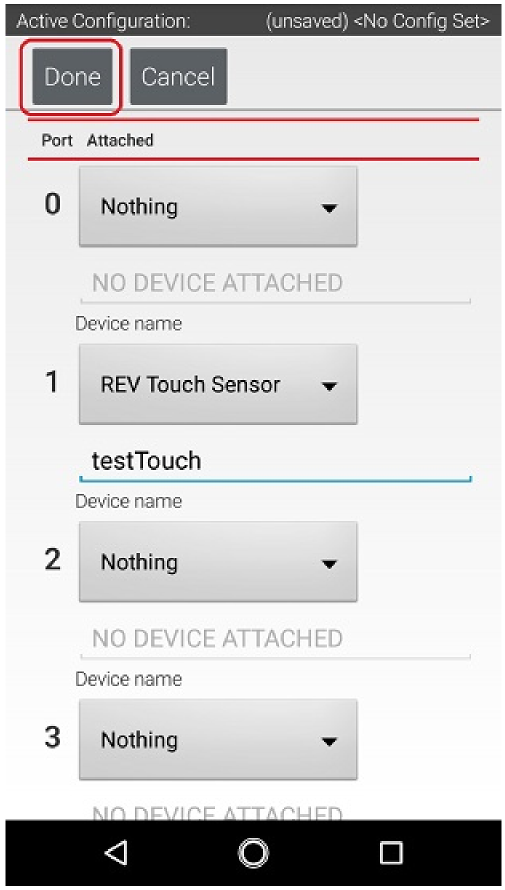


2. Use the touch screen to add a “REV Touch Sensor” for port #1 and name the device “testTouch”.



Notice that we are configuring the Touch Sensor on port #1 instead of port #0. This is because when the REV Robotics Touch Sensor is connected to a digital port using a standard 4-wire JST sensor cable, it is the second digital pin that is connected. The first pin remains disconnected.

3. Press the **Done** button to return to the previous screen.



19.2.6 Configuring an External Webcam with a Control Hub

Introduction

The Competition Manual allows the use of USB Video Class (UVC) compatible cameras for computer vision-related tasks. See rule R715 for the full details. If you are using a REV Robotics Control Hub, then you will need to use an external webcam, since the Control Hub does not include a built-in camera. This document describes how to connect, configure and use an external webcam with a Control Hub.

Special thanks to Chris Johannesen of Westside Robotics (Los Angeles) for putting together this documentation.

Type of External Camera

Theoretically, any USB Video Class (UVC) camera should work with the system. However, FIRST recommends using UVC web cameras from Logitech. The following cameras have been tested and calibrated to work accurately with SDK software:

- Logitech C270 HD Webcam
- Logitech C310 HD Webcam
- Logitech C920 HD Webcam

Calibrating a UVC camera is an optional, advanced task. Instructions for creating a calibration file are in the comments of the teamwebcamcalibrations.xml file in the ftc_app project folder (visit this link for an online copy of the file).

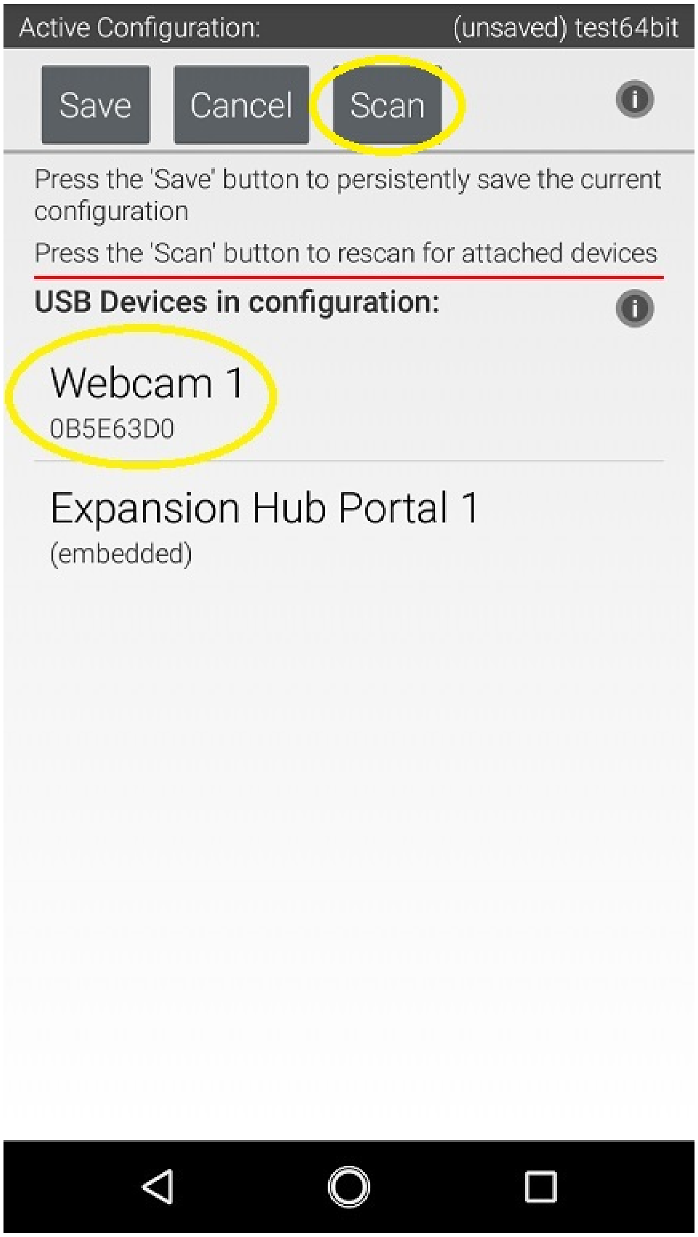
Connecting the Camera

The UVC camera plugs directly into the USB 2.0 port on the REV Control Hub. Unlike the REV Expansion Hub, there is no need for an external powered USB hub.

Camera Configuration

Before using the external camera, it must be added to the active configuration file as a USB-connected device.

Use the Configure Robot menu item on the paired DRIVER STATION device to add the webcam as a USB-connected device to an existing or newly created configuration file. Note that the Scan operation for the Configure Robot activity should detect the webcam and give it a default name of “Webcam 1”.

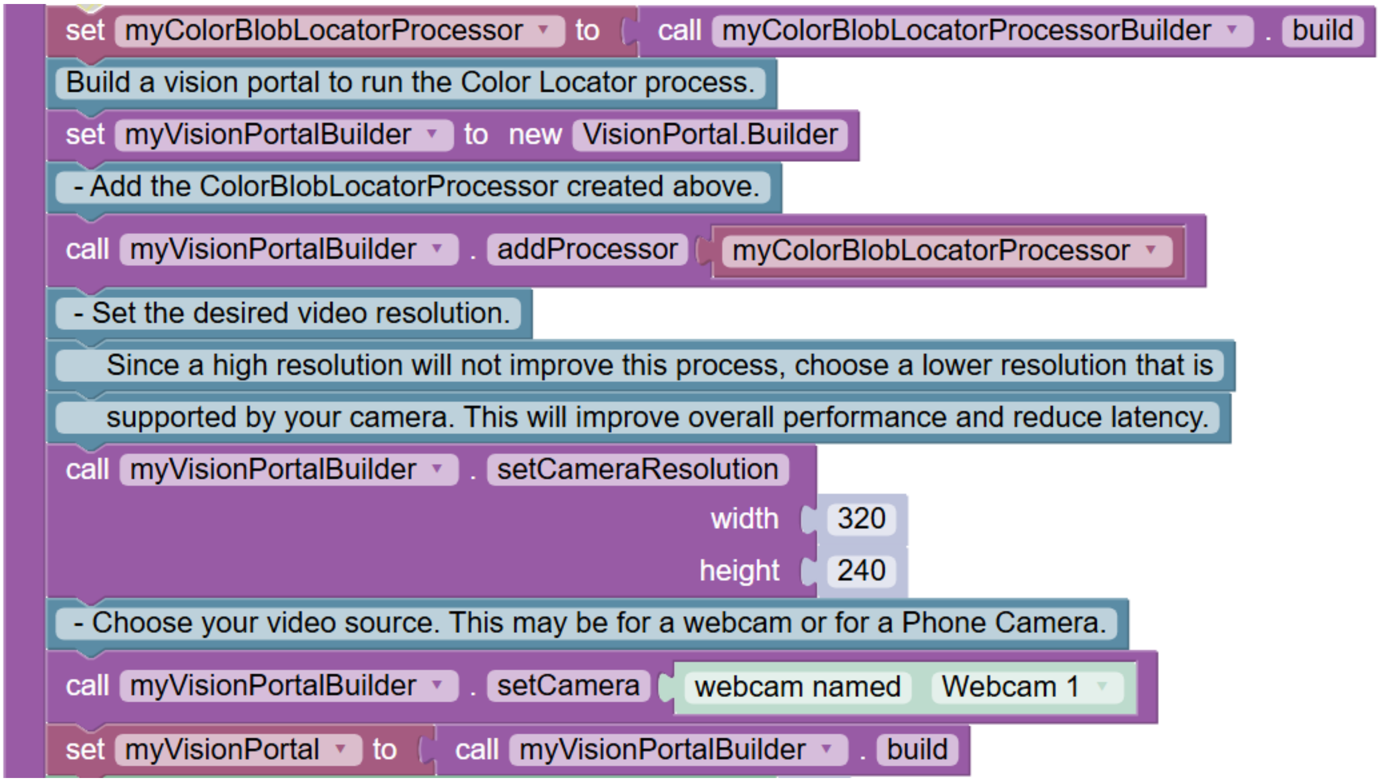


You can keep this default name (the sample Op Modes reference this name) or change it. If you change the webcam name, make sure your Op Modes refer to this new name.

Sample Op Modes

When the configuration has been saved and activated, the external UVC camera can be programmed for robot vision tasks.

The SDK software offers "webcam" versions of its sample Blocks and Java Op Modes, showing how to use the external UVC camera for VisionPortal operations.

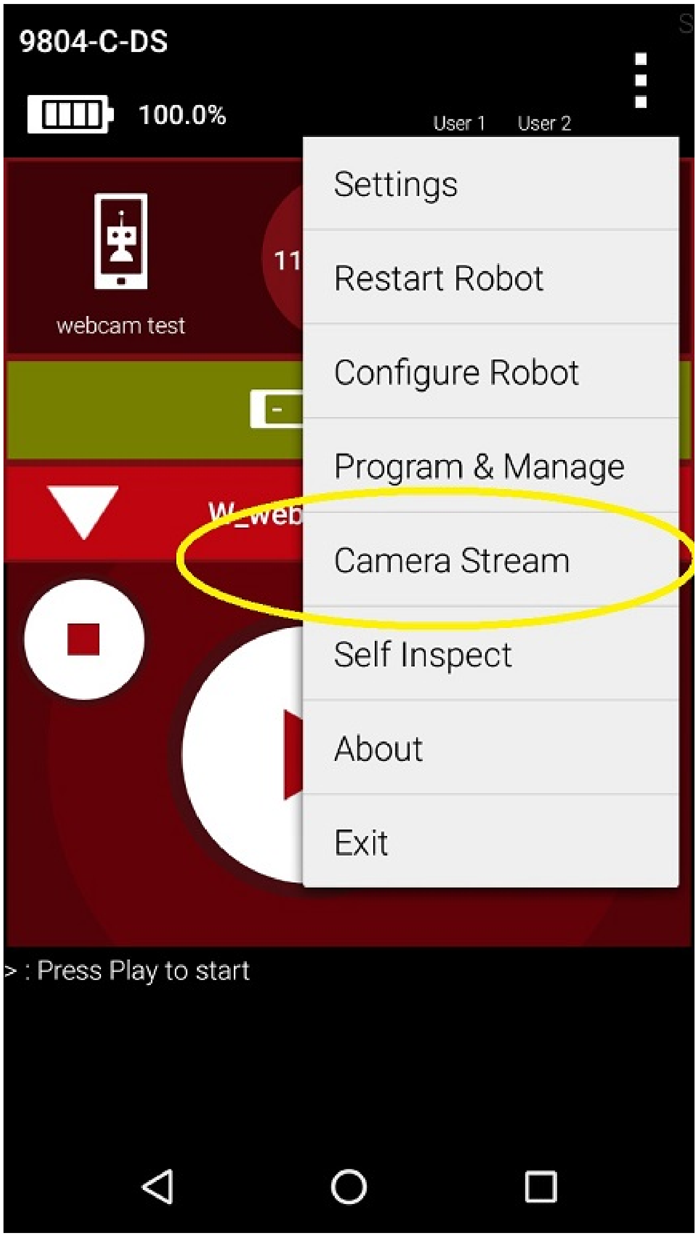


Before opening and editing an Op Mode, verify that the intended configuration (with camera) is active. Also verify that the name referenced in the Op Mode matches the name specified in the configuration file.

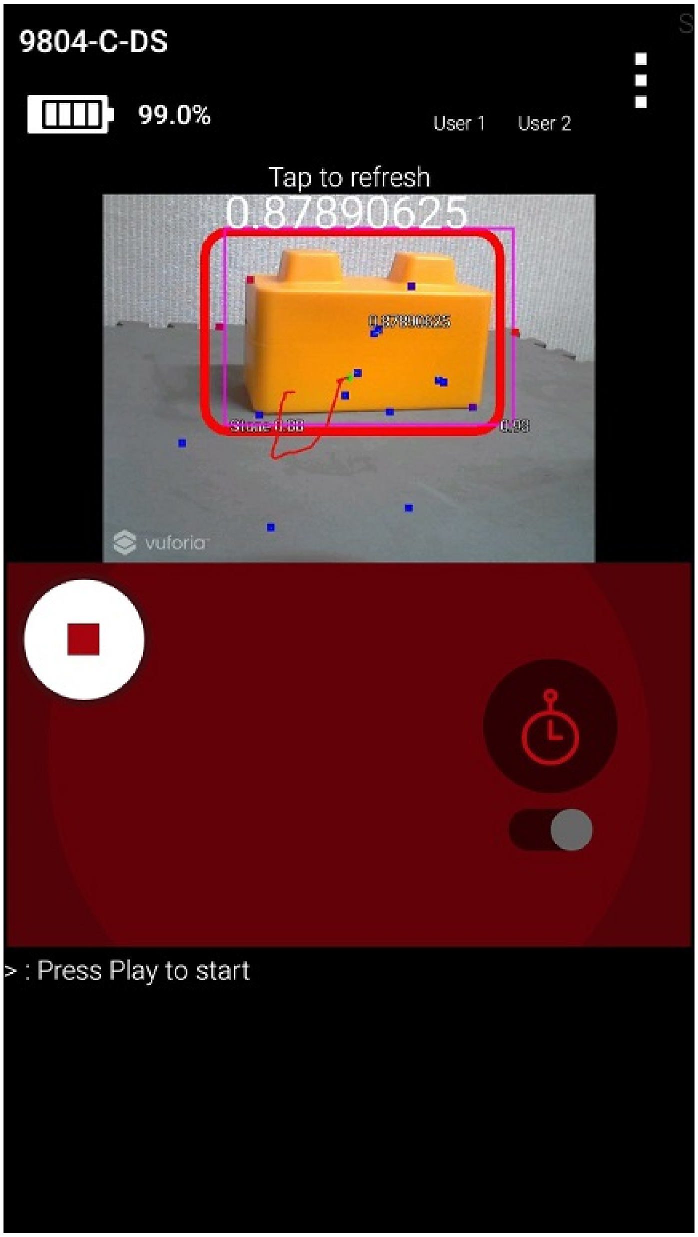
Image Preview

The *FIRST* Tech Challenge apps provide camera preview for 'stream-enabled' Op Modes using VisionPortal.

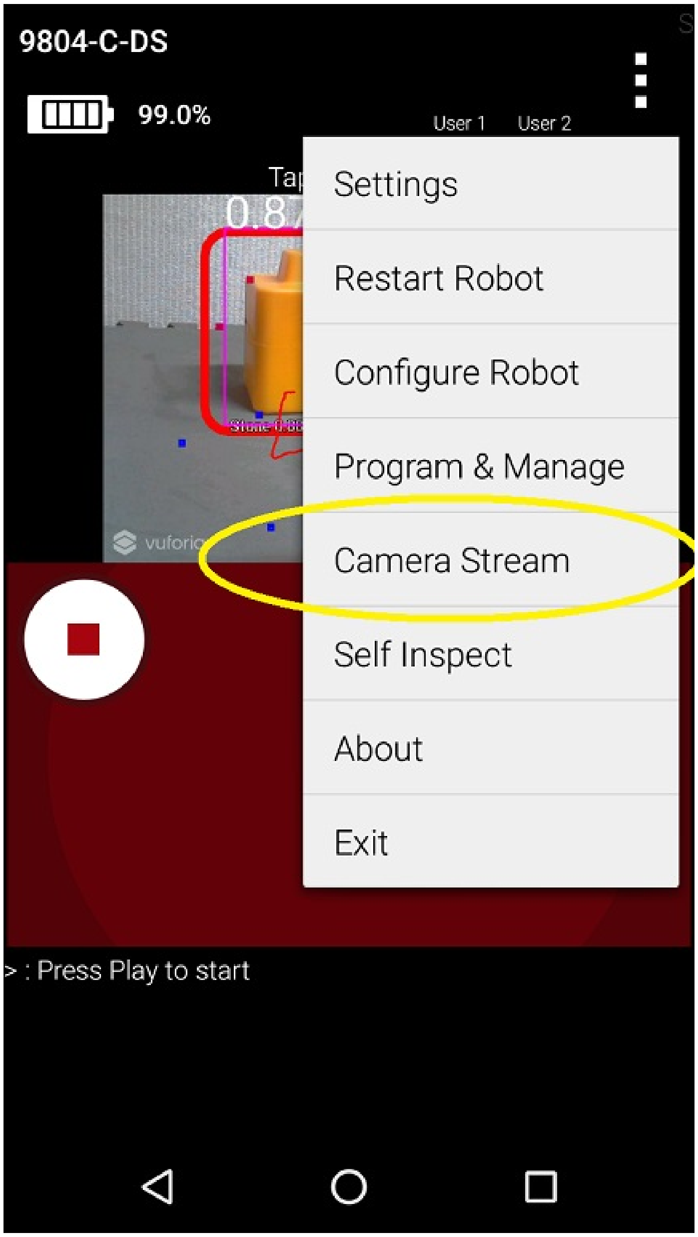
On a paired DRIVER STATION device, with the camera connected and configured, select a stream-enabled Op Mode. Press the INIT button, and wait briefly for streaming software to initialize; do not press the START button. Instead open the main menu (the 3 dots in upper right hand corner of the screen) and select Camera Stream. This option appears only at this time, during which the game pads and START button are disabled for safety.



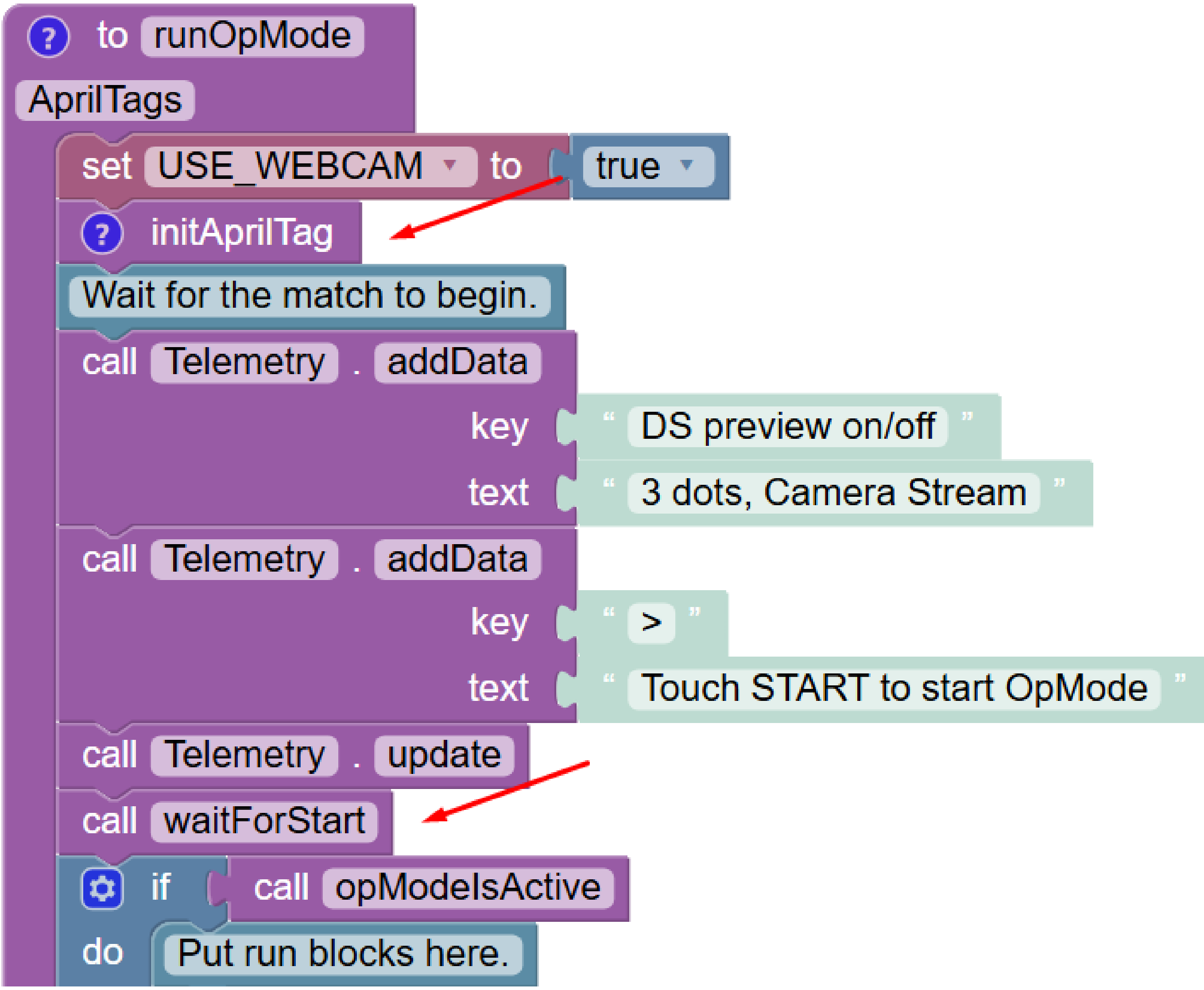
The camera image will appear on the DRIVER STATION screen. Manually touch the image to refresh it. To preserve bandwidth, only one frame is sent at a time.



This option may be used to adjust the camera, with frequent manual image refreshing as needed. When finished, open the main menu and select Camera Stream again to turn off the preview. The preview image will close, the game pads will be enabled, and the START button may be pressed to continue running the Op Mode.



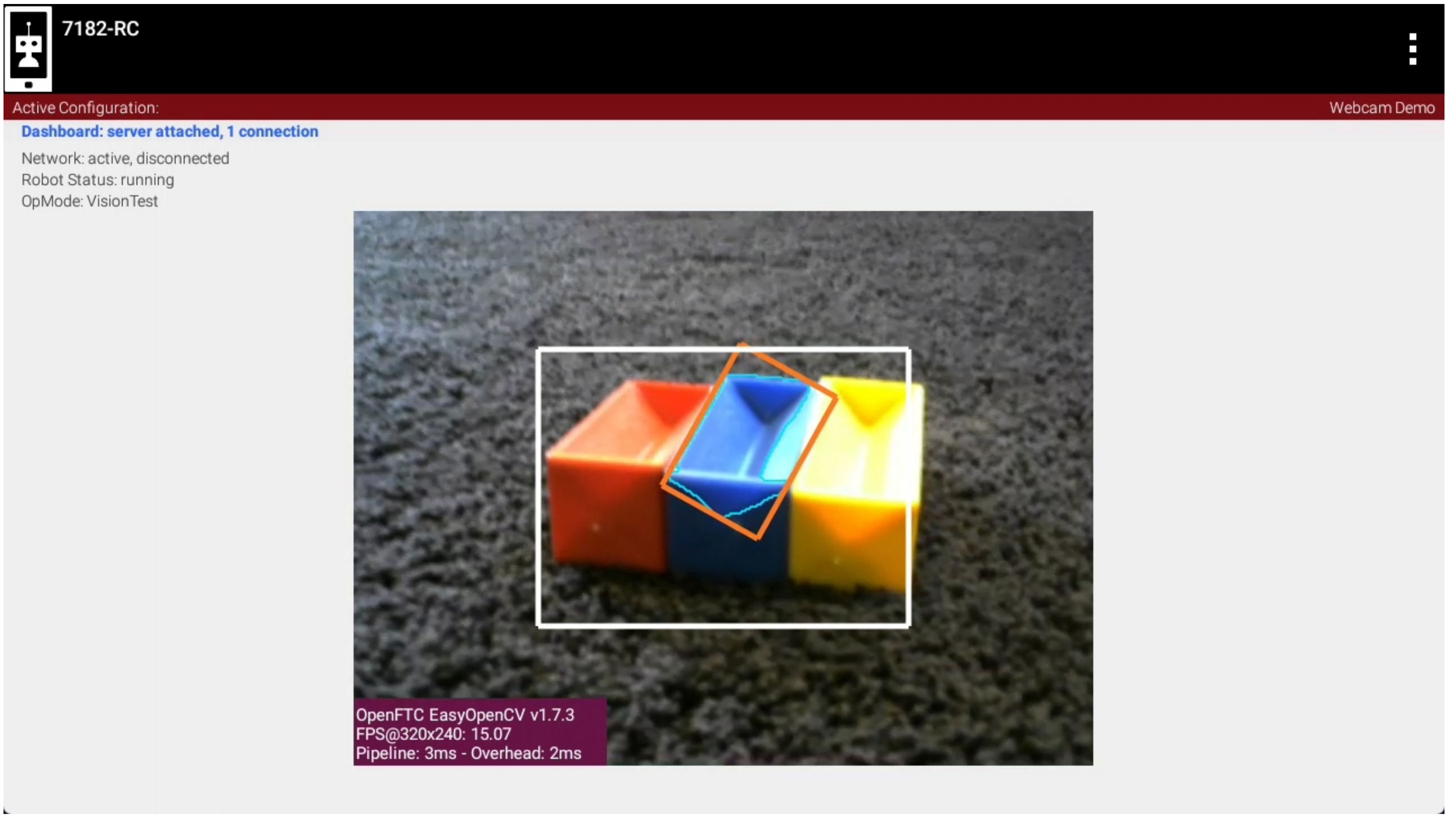
Important Note: Because the Camera Stream feature is only available during the INIT phase of an Op Mode, you must ensure that the VisionPortal is activated in your Op Mode **before** the waitForStart command:



If you do not see the Camera Stream option in your main menu on your DRIVER STATION, then verify that the VisionPortal is activated before the waitForStart command in your Op Mode. Also make sure you've given the system enough time to initialize the Vuforia software before you check to see if Camera Stream is available.

Scrcpy

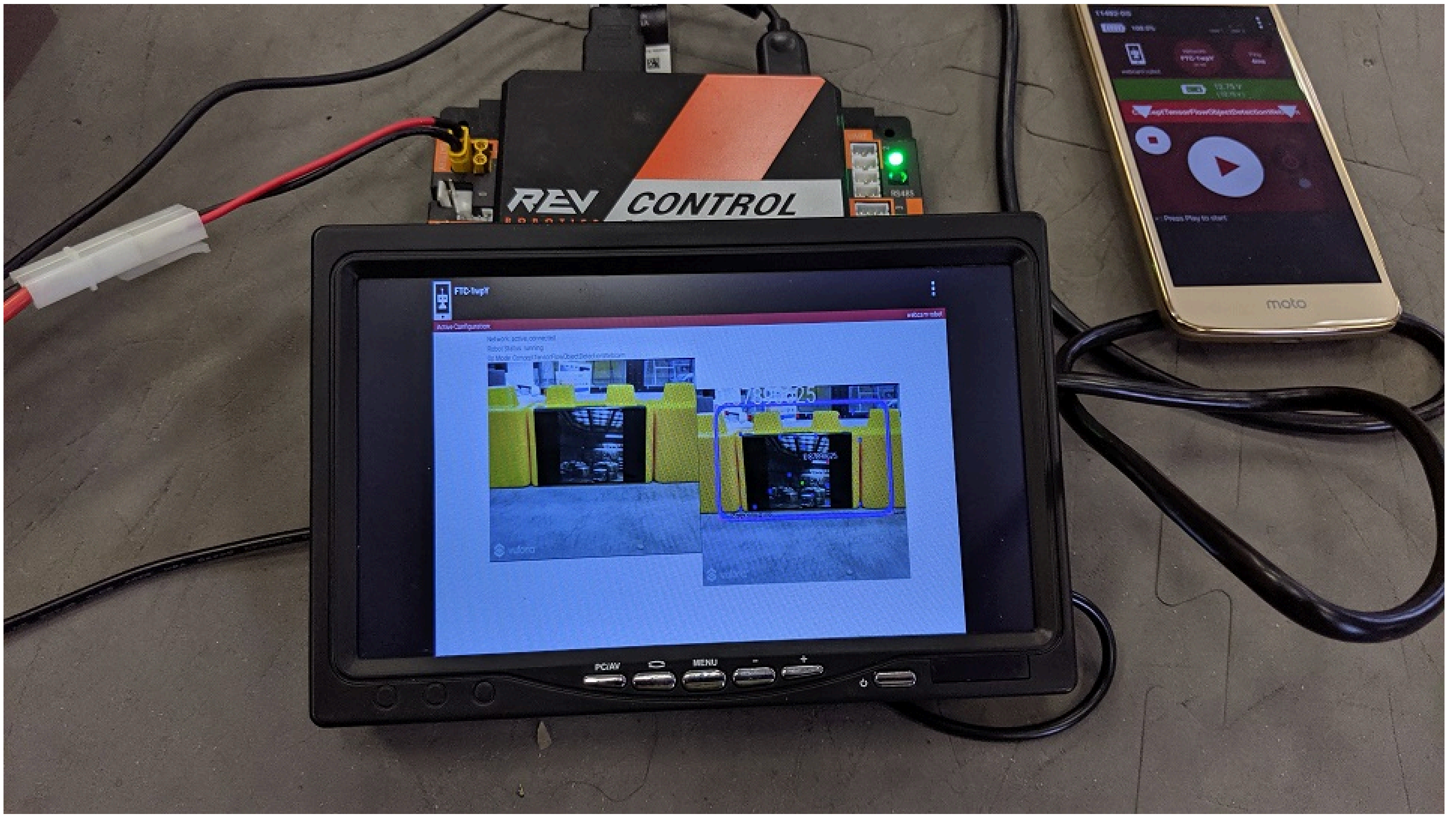
To view the camera output from a computer while an OpMode is running, you can use scrcpy. To do this, you will first need to obtain an ADB connection with your Control Hub. This can be done by connecting a USB-A - USB-C cable to the USB-C port on your Control Hub. If on Windows, you may also connect to your Control Hub WiFi network and open the REV Hardware Client. Once connected, use these instructions to install and run scrcpy on your computer.



Important Note: While scrcpy is a great way to view the camera output outside of competitions, the Competition Manual does not allow teams to have any devices other than the DRIVER STATION connected to your Control Hub during a match. See rule R708 for the full details.

External HDMI Monitor

Alternatively, camera output can be viewed on a display monitor or other device plugged into the HDMI port on the REV Control Hub.



Important Note: While a portable display monitor can be used to view or troubleshoot the camera stream on your Control Hub, teams are not allowed to have a portable display monitor connected to their Control Hub during a match.

Advanced Users

For custom streams, advanced users of Android Studio may consult the API documentation for CameraStreamClient, CameraStreamServer and CameraStreamSource classes.

19.2.7 Configuring an External UVC Camera and a Powered USB Hub

Introduction

The Competition Manual allows the use of USB Video Class (UVC) compatible cameras for computer vision-related tasks. See rule R715 for the full details. Teams who are using an Android smartphone as their ROBOT CONTROLLER have the option of using an externally connected camera instead of the built-in camera for computer vision tasks.

The advantage of using an external camera is that the camera can be mounted in a location that is convenient for vision-related tasks while the Android Robot Controller can be mounted where it is convenient for Robot Controller-related tasks.

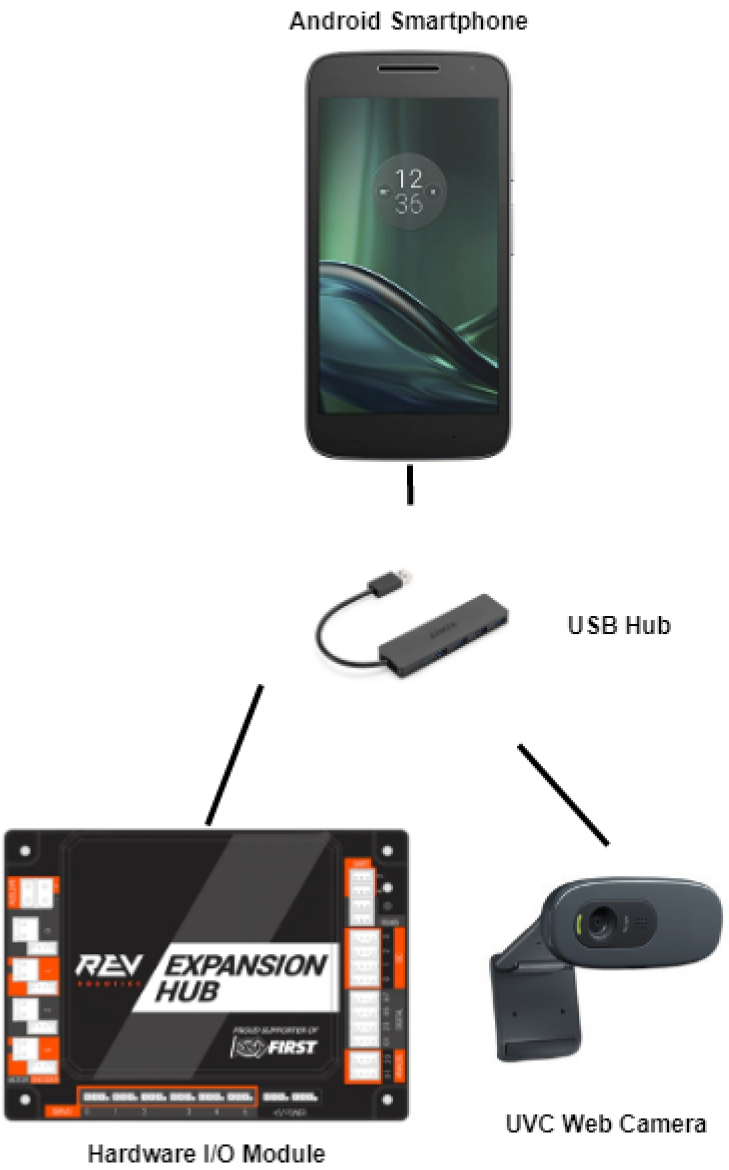
The disadvantage of using an external camera is there is additional complexity introduced with the USB-connected camera. An external camera adds costs and weight to a robot and it needs to be wired correctly to run properly.

What type of External Camera can be Used?

The system supports a "UVC" or USB Video Device Class cameras. Theoretically, if a camera is UVC compliant, then it should work with the system. However, there are a couple of recommended web cameras that have been tested with the FIRST Tech Challenge software and have been calibrated to work accurately with this software:

- Logitech HD Webcam C310
- Logitech HD Pro Webcam C920

Note that calibrating a UVC camera is an advanced task. Details on how to create a calibration file can be found in the comments of the *teamwebcamcalibrations.xml* file that is available as part of the ftc_app project folder (visit this link for an online copy of the file).



USB Hub

Teams who would like to use an external camera will need a USB hub to connect their Android Robot Controller to the external camera and the REV Robotics Expansion Hub. To work properly, the USB hub should meet the following requirements:

1. Compatible with USB 2.0.
2. Supports a data transfer rate of 480Mbps.

Note that the Modern Robotics Core Power Distribution Module cannot be used for this task since its data transfer speed is not fast enough to work with the USB-connected webcam.

Also note that the Competition Manual permits the use of a powered USB hub to make this connection. See rule R617 for the full details. If a team uses a powered USB hub, the power to operate the USB hub can only come from either of the following sources:

1. An externally connected COTS USB Battery Pack in compliance with the Competition Manual. See rule R602 for the full details.
2. The 5V DC Aux power port of a REV Robotics Expansion Hub (note that this requires advanced skills to implement).

FIRST has tested a few USB 2.0 powered hubs and recommends one from Anker. At the time this document was written, this hub was available from Anker.com.

The Anker 4-port powered hub is convenient because it has a Micro USB port that is used to connect the hub to a 5V power source (highlighted with orange circle in figure below).

This port allows a user to plug a standard USB type B Micro Cable into the hub, and then connect the other end of the cable (which has a USB Type A connector) into the output port of an external 5V USB battery pack. In the image below, the Anker 4 port hub is powered by a “limefuel” external 5V battery pack using a standard Type A to Type B USB Micro cable. Note the battery is highlighted by the yellow outline in the figure below.

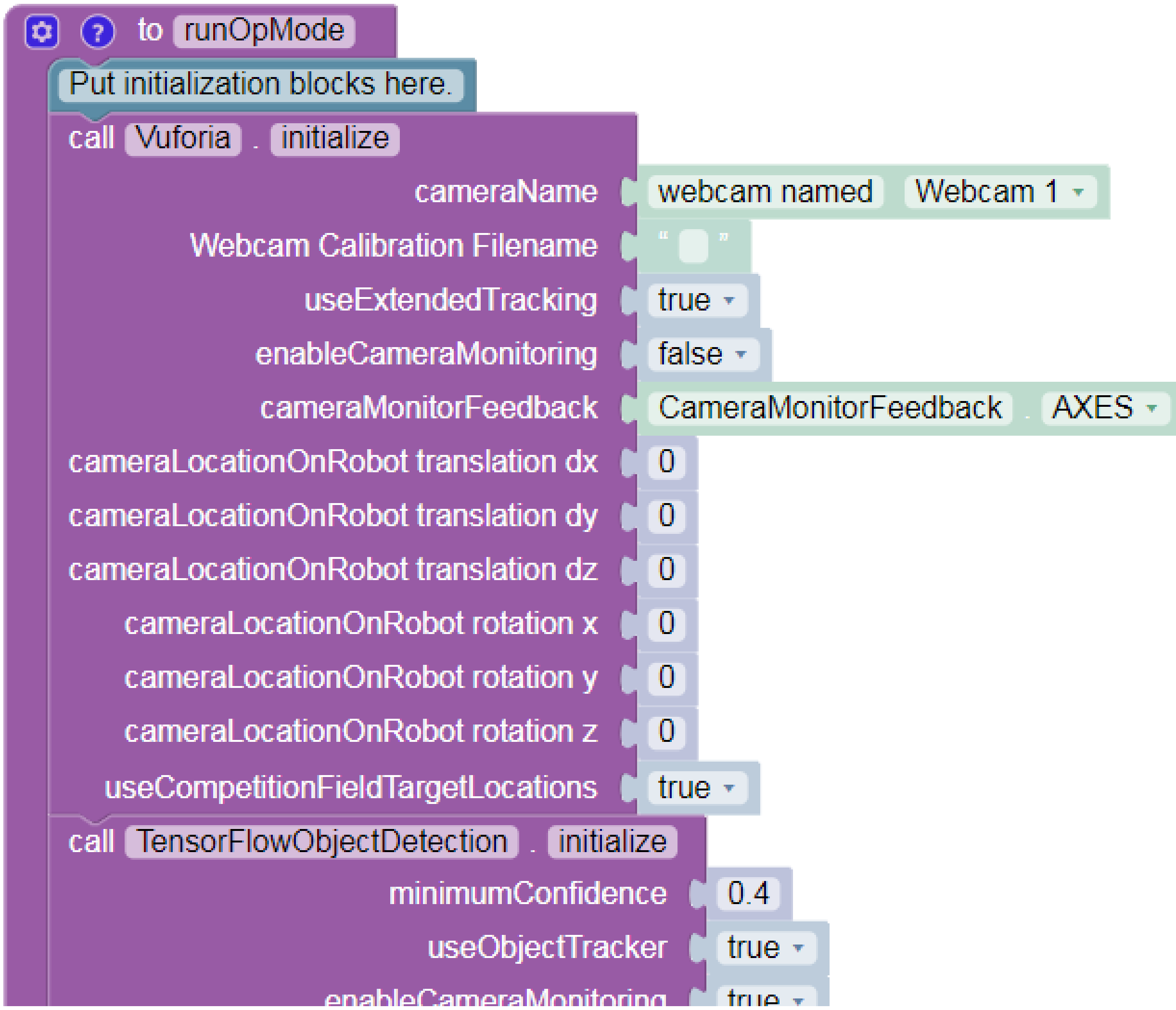
A USB hub can also draw power from the 5V auxiliary ports on the REV Robotics Expansion Hub. This configuration requires that the user have a special cable that on one end can be plugged into the 5V Auxiliary port and on the other end can be plugged into the power port of the USB hub.

Note that teams can create this special cable using one end of a servo extension cable (to plug into the 5V aux port) and one end of a Micro USB cable (to plug into the Anker hub's power port). **Creating this cable is an advanced task and should only be attempted by teams who have guidance from an adult mentor who has expertise in electronics and wiring! It is extremely important that the polarity is correct for this special cable. If the polarity is reversed it could damage your electronic equipment.**

Sample Op Modes

There are sample Blocks and Java Op Modes that demonstrate how to use the external UVC web camera for Vuforia or TensorFlow operations. Before a team can use the external UVC camera, a configuration file must be configured with the external camera defined as one of the USB-connected devices.

Once a valid configuration file has been defined and activated, the programmer can use the external UVC camera, instead of the internal Android cameras, for vision-related tasks.



19.2.8 Adding an Expansion Hub

Introduction

A single REV Robotics Control or Expansion Hub has a limited amount of input/output (I/O) ports available. In some instances, you might want to use more devices than there are ports available. For these instances you might need to connect an Expansion Hub to your first Hub to add more I/O ports.

This document describes how to connect and configure an additional Expansion Hub for use in the FIRST Tech Challenge. Note that the FIRST Tech Challenge Competition Manual limits the maximum number of Control or Expansion Hubs on a single robot to two. See rule R701 for the full details.

Equipment Needed

To follow along with the instructional steps in this document, you will need the following items:

Required Item(s)	Image
REV Robotics Driver Hub (REV-31-1596)	
REV Robotics Switch, Cable, & Bracket (REV-31-1387).	
REV Robotics Tamiya to XT30 Adapter Cable (REV-31-1382).	
FIRST-approved 12V Battery (such as Tetrix W39057). For a list of FIRST-approved 12V batteries, refer to the current Competition Manual, rule R601.	

continues on next page

Table 1 – continued from previous page

Required Item(s)	Image
REV Robotics Control Hub (REV-31-1595).	
REV Robotics Expansion Hub (REV-31-1153).	

continues on next page

Table 1 – continued from previous page

Required Item(s)	Image
REV Robotics (or equivalent) 3-Pin JST PH Cable (REV-35-1414, 3 pack shown but only one needed).	
REV Robotics XT30 Extension Cable (REV-31-1394).	

Connecting the Expansion Hub

1. The first step is to use the 3-pin JST PH cable and the XT30 cable to daisy chain the two Hubs together. Before you do this, ensure that neither Hub is powered on.

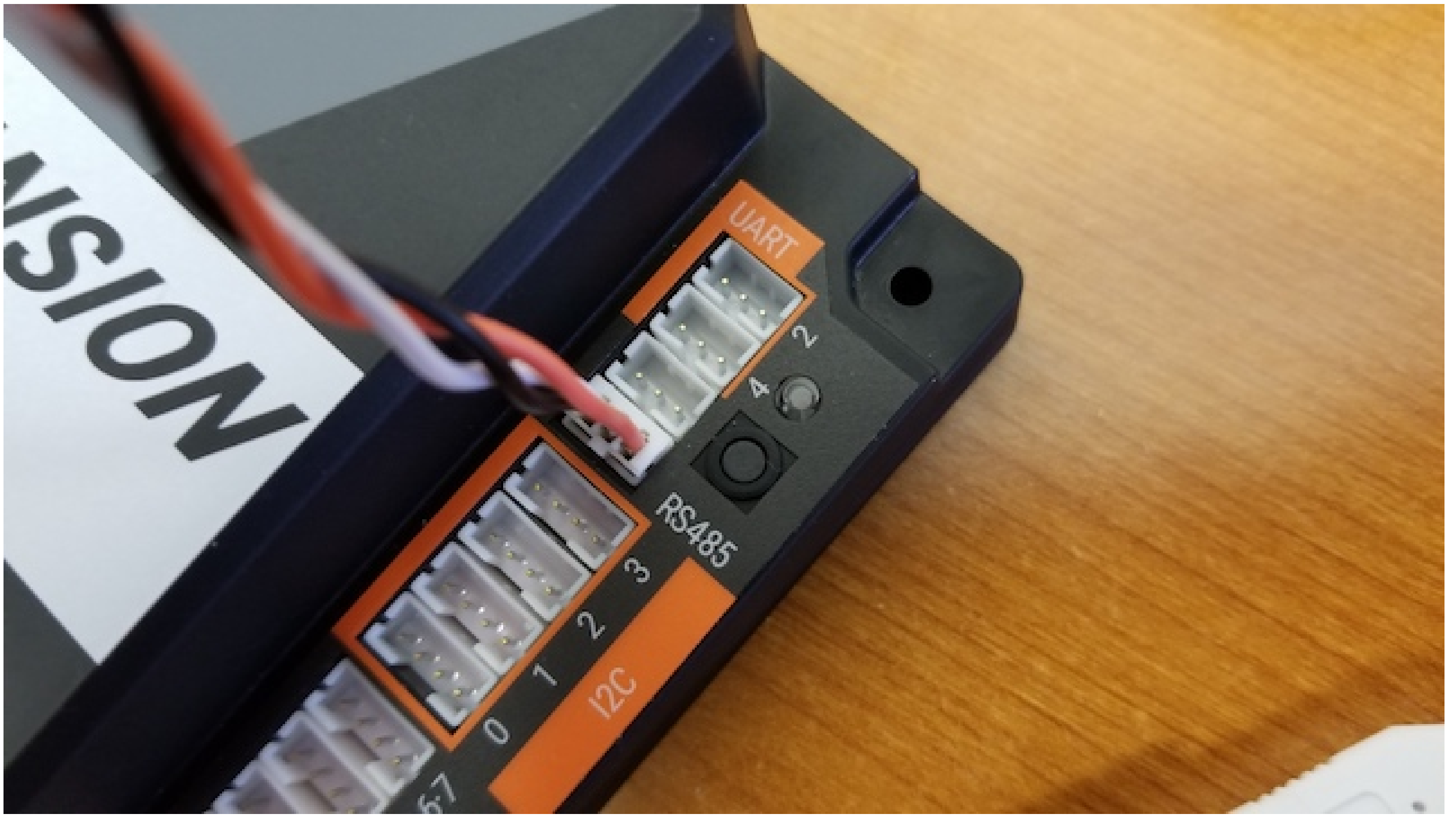
Use the XT30 extension cable to connect an XT30 power port on the Control Hubs to an XT30 power port on the other Expansion Hub.

<INSET IMAGE>

2. The Control Hub and Expansion Hub use the RS-485 serial bus standard to communicate between devices. You can use the 3-pin JST PH cable to connect one of the ports labeled “RS485” on the Control Hub to one of the ports labeled “RS485” on the Expansion Hub.

<INSERT IMAGE>

Note that it is not important which “RS485” port that you select on the Expansion Hub or Control Hub. Either port should work.



3. Once you have the two devices daisy chained together (12V power and RS-485 signal) you can connect the battery and power switch, and power on the devices.

<INSERT IMAGE>

Configuring Both devices

If you successfully daisy chained your Expansion Hub and Control Hub, then you should be able to create a new configuration file that includes both devices.

Note: If you already have a configuration that contains just the Control Hub, you can add the Expansion Hub by editing the configuration and pressing the "Scan" button.

Connect your Driver Hub to the Control Hub's WiFi network and select the Configure Robot option from the Driver Station app. Press the New button to create a new configuration file. When you first scan for hardware, your Robot Controller should detect the embedded Control Hub. The Robot Controller will automatically label this device as an Control Hub "Portal". The Robot Controller will communicate through this portal to the individual Hubs.

<INSERT IMAGE>

If you click on the Portal item in the configuration screen, you should see both the Control Hub and the Expansion Hub listed.

<INSERT IMAGE>

You can save this configuration file and return to the main screen of the Driver Station. After the robot has been restarted, both Hubs should have a solid green LED. On the Expansion Hub, the LED should blink blue every ~5 seconds.

Congratulations, you are now ready to use your combination of Control and Expansion Hubs! You can configure and operate these Hubs as you would an individual Hub.

Using Two Expansion Hubs

Teams without access to a Control Hub may use two Expansion Hubs on their robot.

Additional Equipment Needed

There is some additional equipment required for teams who aren't using a Control Hub on their robot.

Required Item(s)	Image
A FIRST-approved Android smartphone with the FTC Robot Controller app installed. For a list of FIRST-approved Android smartphones, refer to the current Competition Manual, rule R704.	
USB Type A male to type mini-B male cable.	

continues on next page

Table 2 – continued from previous page

Required Item(s)	Image
Micro USB OTG adapter.	
An additional REV Robotics Expansion Hub (REV-31-1153).	

Changing the Address of an Expansion Hub

You can use the Advanced Settings menu of the Robot Controller App to change the address of any connected Expansion Hubs.

Important Note: If both of your Expansion Hubs have the same address or were just removed from the box (by default, the address is set to 2), you need to change the address of one of them _before_ connecting them together. This guide assumes that you will be setting the address of the first Expansion Hub before connecting the second Expansion Hub.

With your first Expansion Hub connected to the 12V battery and to the Robot Controller, launch the Settings menu from the Robot Controller app (note you can also do this from the Driver Station app, if the DRIVER STATION is paired to the Robot Controller).

1. Select the Advanced Settings item to display the Advanced Settings menu.

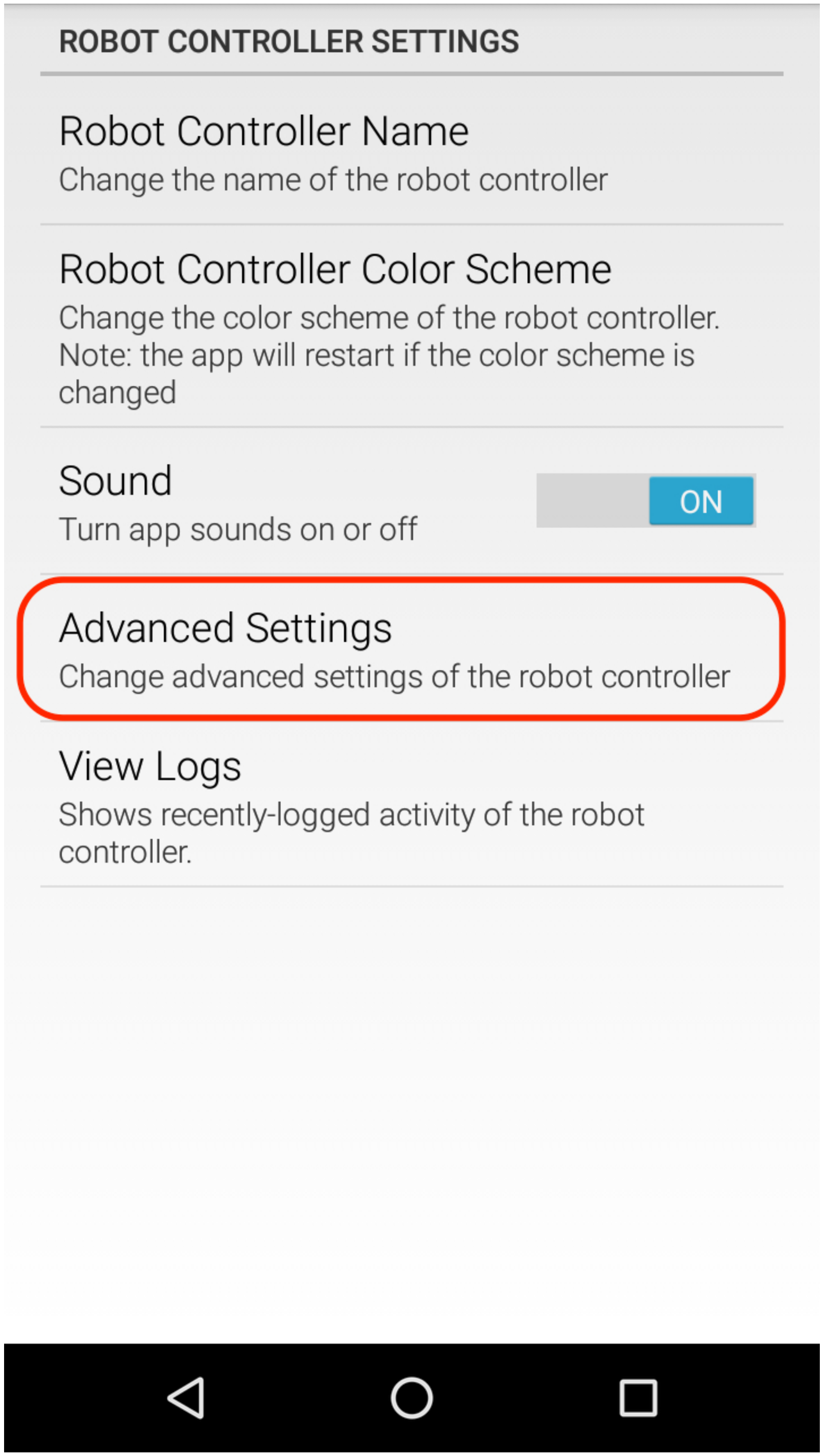
2. Then select the Expansion Hub Address Change item to display the Expansion Hub address screen.

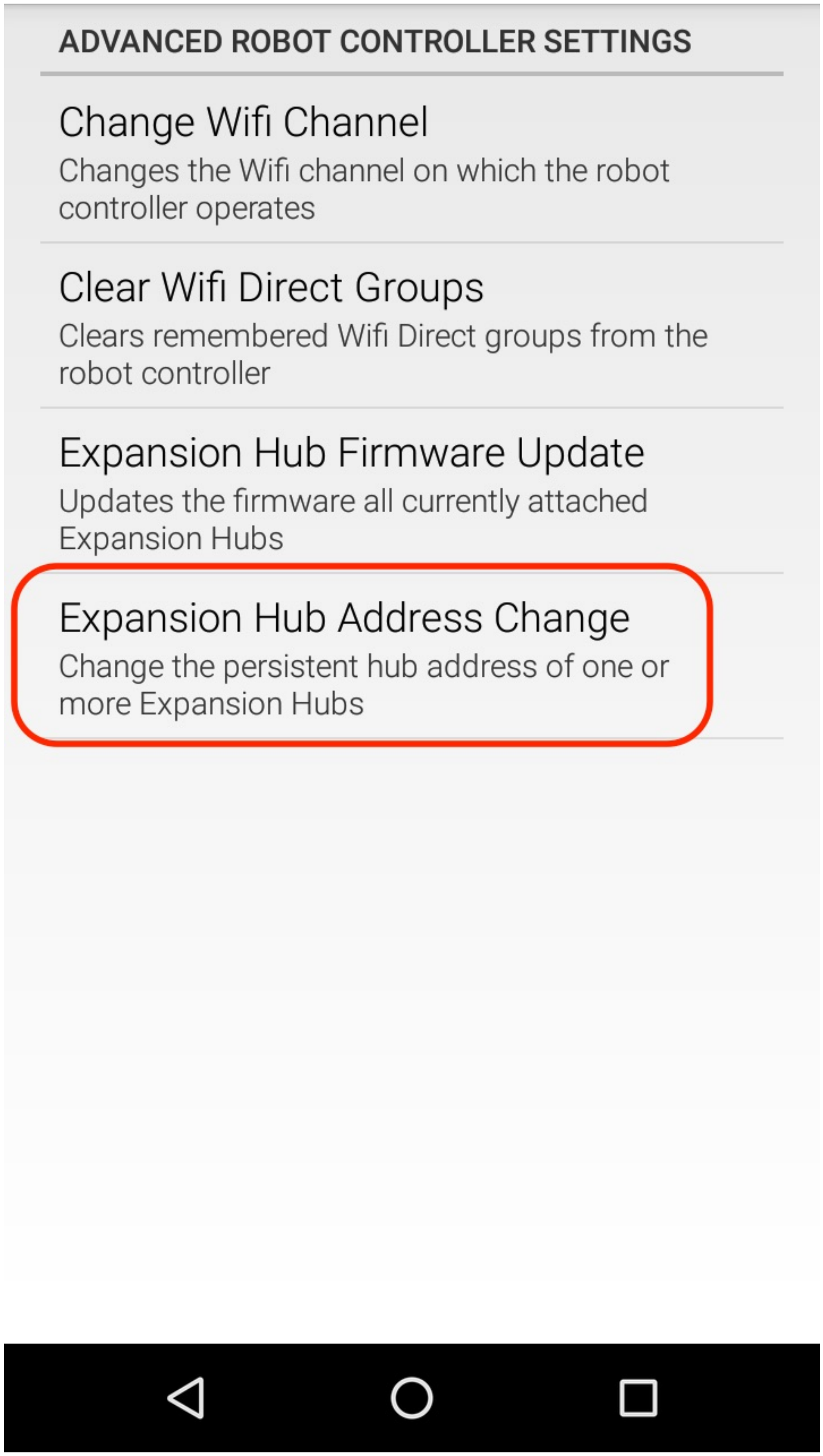
3. The USB serial number of the Expansion Hub and its currently-assigned address should be displayed.

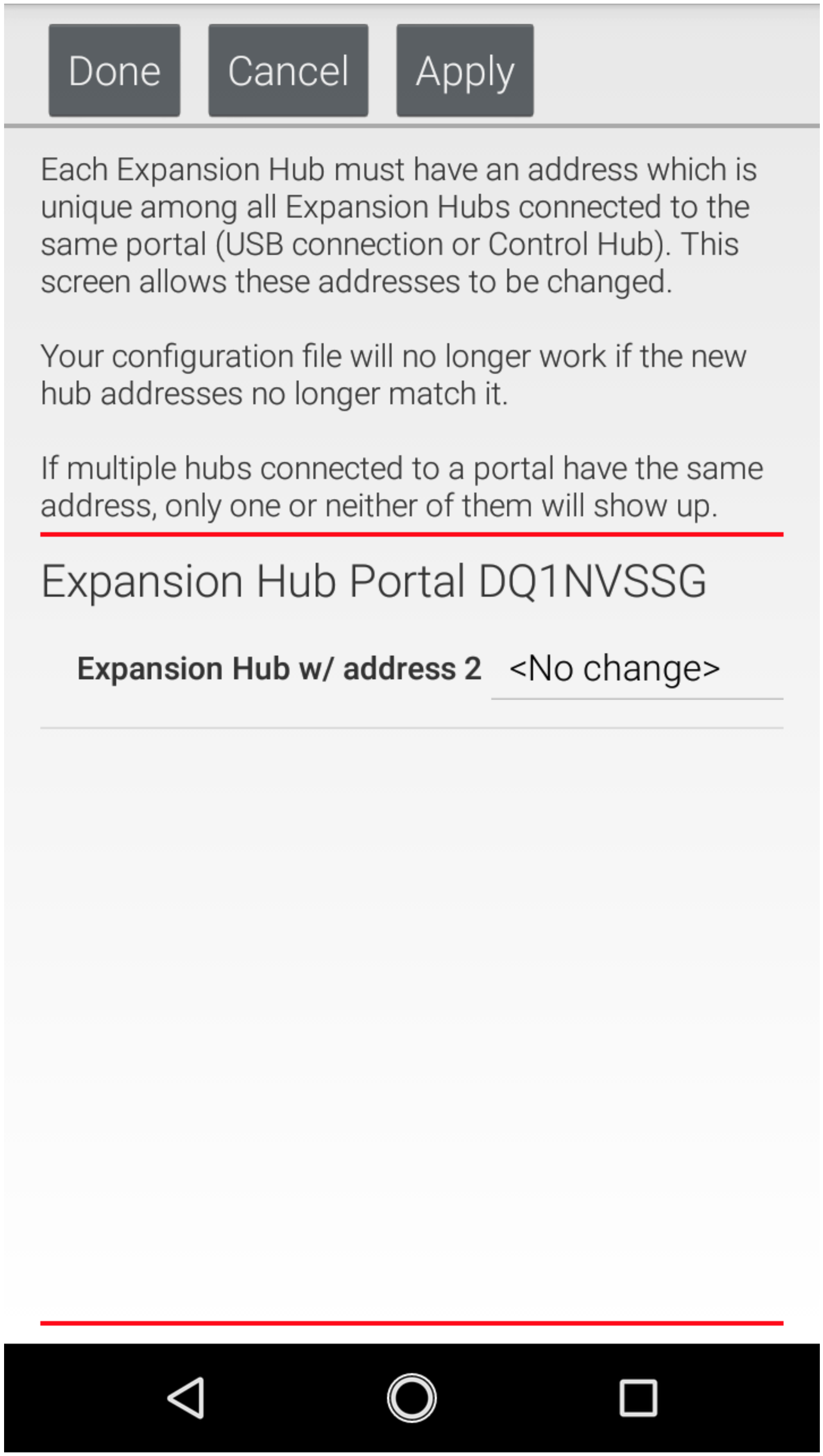
Important Note: If any Expansion Hubs that are physically connected and powered are not displayed, there may be an address conflict. If this happens, disconnect all Expansion Hubs except the one whose address you want to change.

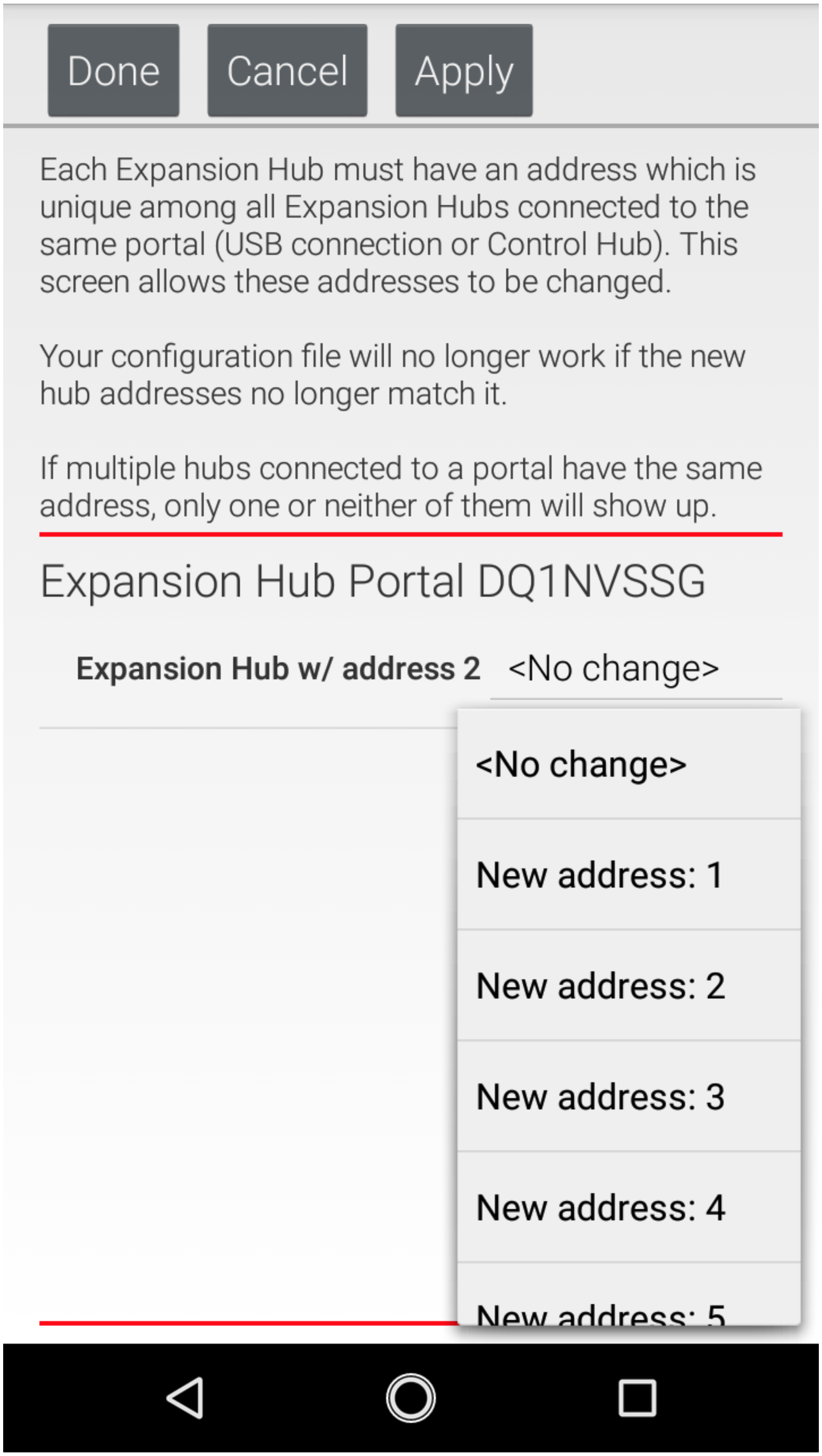
4. Use the dropdown list control on the right hand side to change an Expansion Hub's address. Addresses that conflict with other currently-connected Expansion Hubs won't be available.

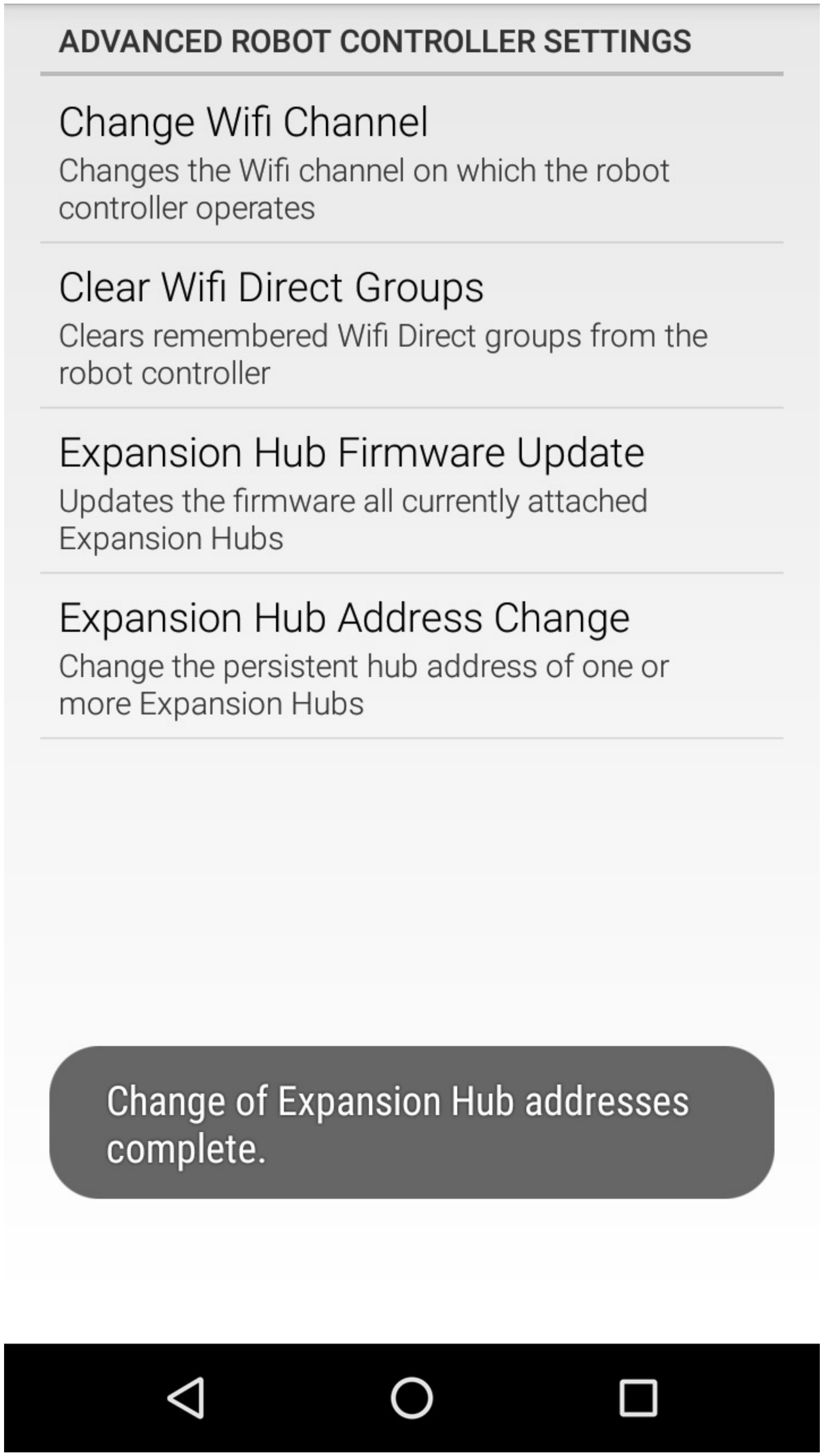
Push the "Done" button to change the address. You should see a message indicating that the Expansion Hub's address has been changed.


ROBOT CONTROLLER SETTINGS
Robot Controller Name
Change the name of the robot controller
Robot Controller Color Scheme
Change the color scheme of the robot controller. Note: the app will restart if the color scheme is changed
Sound
Turn app sounds on or off
ON
Advanced Settings
Change advanced settings of the robot controller
View Logs
Shows recently-logged activity of the robot controller.


ADVANCED ROBOT CONTROLLER SETTINGS
Change Wifi Channel
Changes the Wifi channel on which the robot controller operates
Clear Wifi Direct Groups
Clears remembered Wifi Direct groups from the robot controller
Expansion Hub Firmware Update
Updates the firmware all currently attached Expansion Hubs
Expansion Hub Address Change
Change the persistent hub address of one or more Expansion Hubs


Done
Cancel
Apply
Each Expansion Hub must have an address which is unique among all Expansion Hubs connected to the same portal (USB connection or Control Hub). This screen allows these addresses to be changed.
Your configuration file will no longer work if the new hub addresses no longer match it.
If multiple hubs connected to a portal have the same address, only one or neither of them will show up.
Expansion Hub Portal DQ1NVSSG
Expansion Hub w/ address 2
<No change>

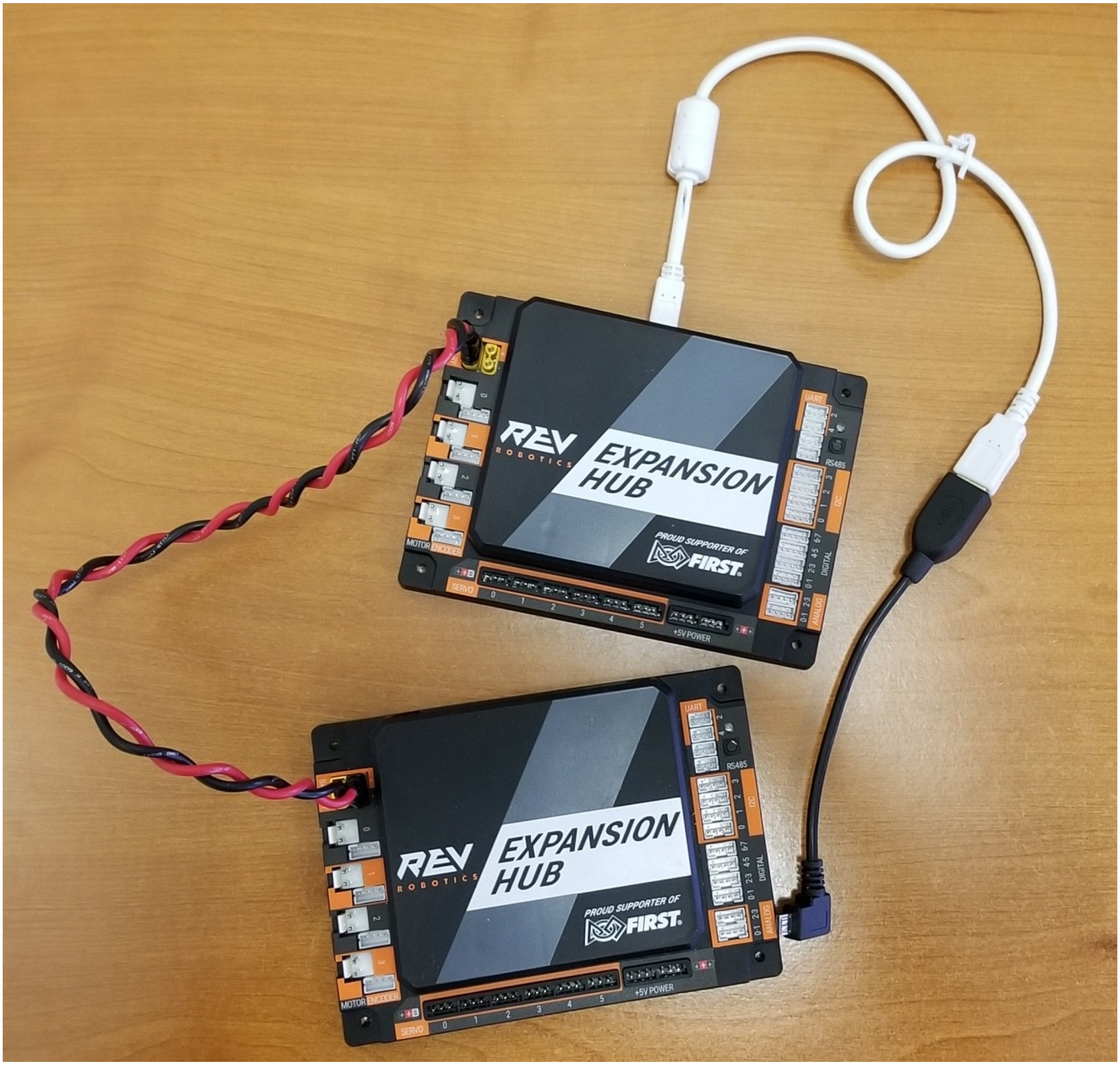

Done
Cancel
Apply
Each Expansion Hub must have an address which is unique among all Expansion Hubs connected to the same portal (USB connection or Control Hub). This screen allows these addresses to be changed.
Your configuration file will no longer work if the new hub addresses no longer match it.
If multiple hubs connected to a portal have the same address, only one or neither of them will show up.
Expansion Hub Portal DQ1NVSSG
Expansion Hub w/ address 2
<No change>
<No change>
New address: 1
New address: 2
New address: 3
New address: 4


ADVANCED ROBOT CONTROLLER SETTINGS
Change Wifi Channel
Changes the Wifi channel on which the robot controller operates
Clear Wifi Direct Groups
Clears remembered Wifi Direct groups from the robot controller
Expansion Hub Firmware Update
Updates the firmware all currently attached Expansion Hubs
Expansion Hub Address Change
Change the persistent hub address of one or more Expansion Hubs
Change of Expansion Hub addresses complete.

Connecting the Two Expansion Hubs

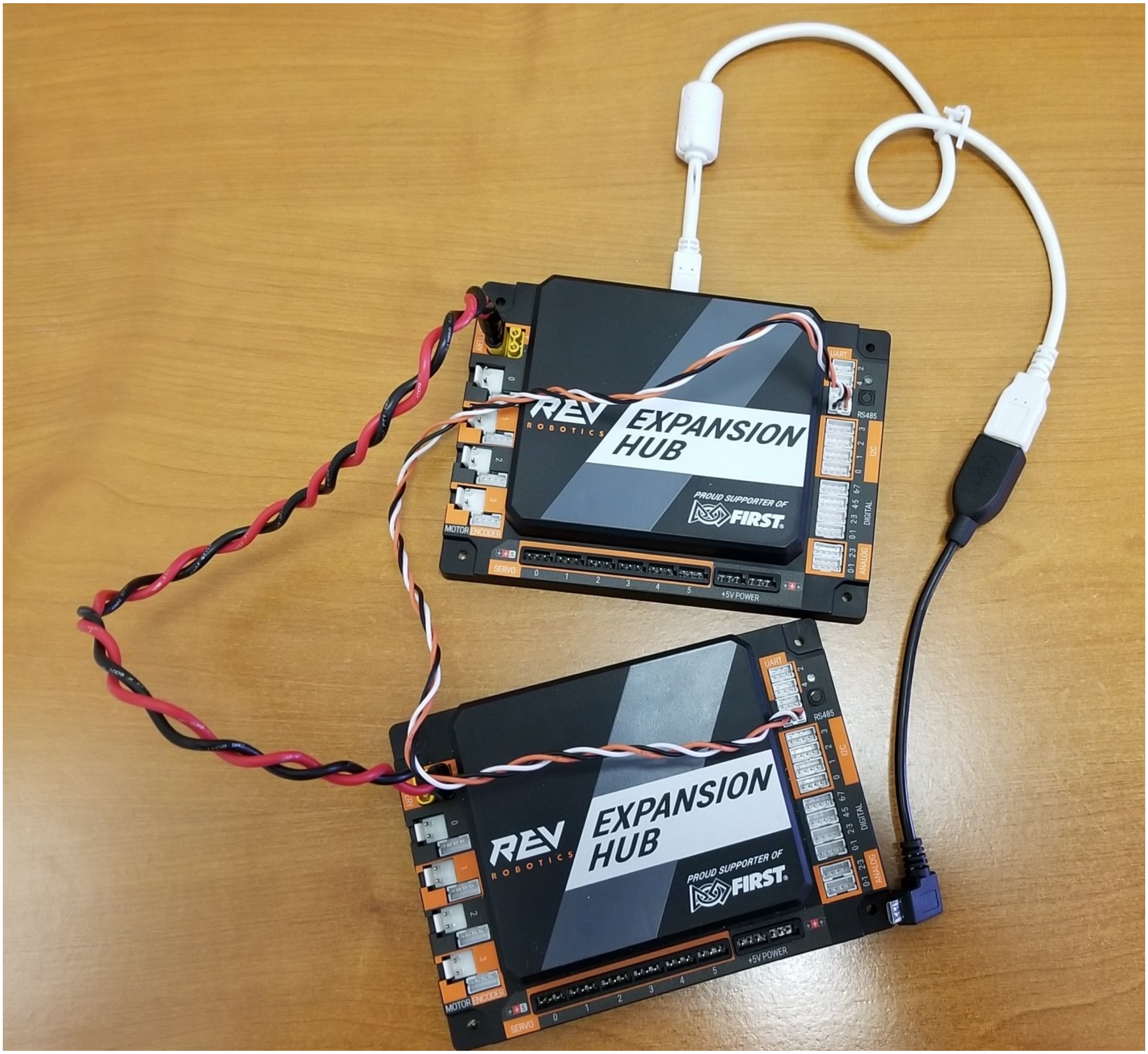
5. After you have changed the address of one of the Hubs, you can use the 3-pin JST PH cable and the XT30 cable to daisy chain the two Hubs together. Before you do this, disconnect the 12V battery and power switch from the first Expansion Hub.

Use the XT30 extension cable to connect an XT30 power port on one of the Expansion Hubs to an XT30 power port on the other Hub.

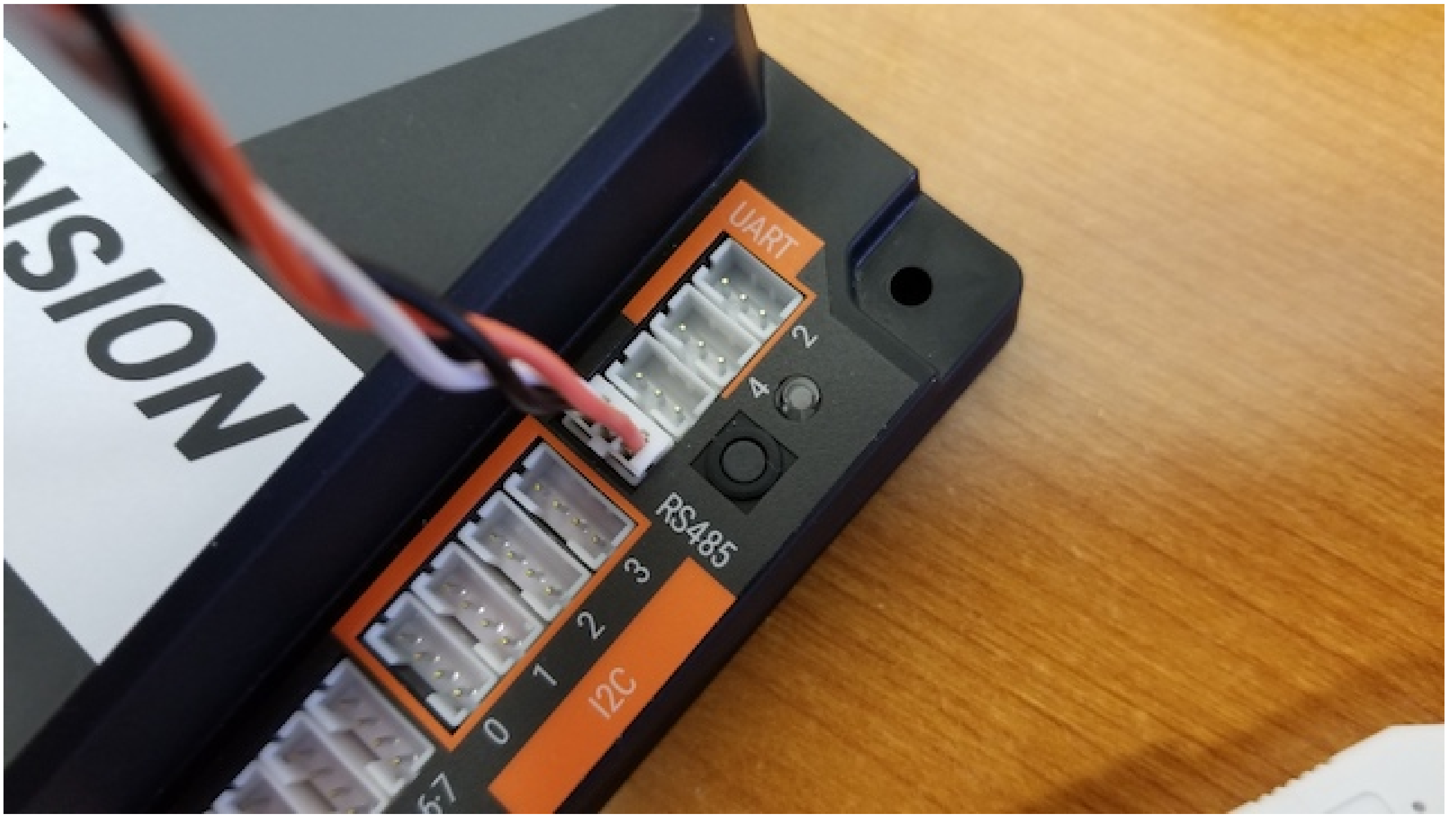


6. The Expansion Hubs use the RS-485 serial bus standard to communicate between devices. You can use the 3-pin JST PH cable to connect one of the ports labeled "RS485" on one Expansion Hub to one of the ports labeled "RS485" on the other Expansion Hub.

Note that it is not important which "RS485" port that you select on an Expansion Hub. Either port should work.



REV ROBOTICS
EXPANSION HUB
PROUD SUPPORTER OF FIRST



7. Once you have the two devices daisy chained together (12V power and RS-485 signal) you can reconnect the battery and power switch, and then connect the Robot Controller and power on the devices.

Configuring Your Expansion Hubs

If you successfully daisy chained your two Expansion Hubs, then you should be able to create a new configuration file that includes both devices.

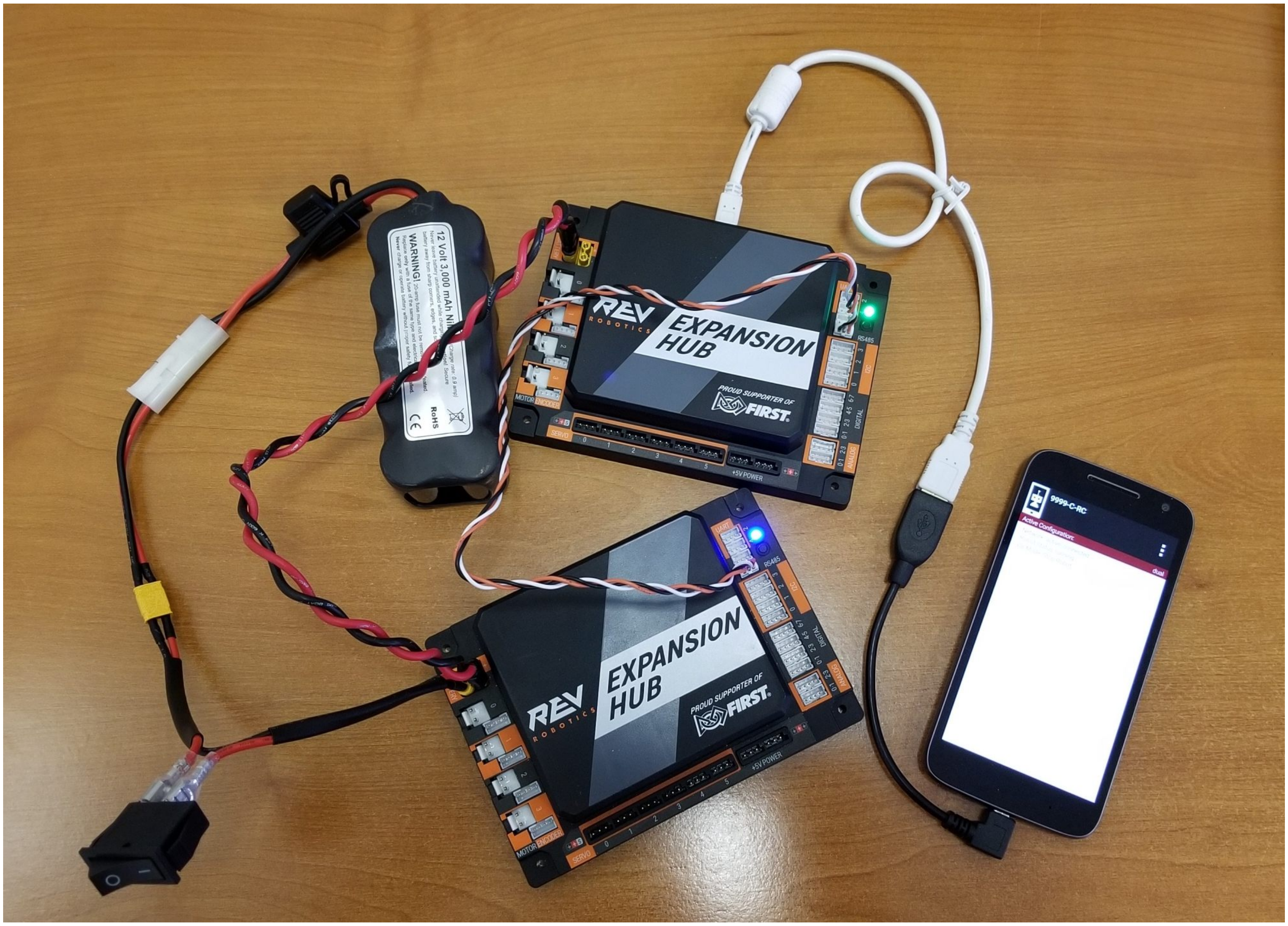
Note: If you already have a configuration that contains just the USB-connected Expansion Hub, you can add the second Expansion Hub by editing the configuration and pressing the "Scan" button.

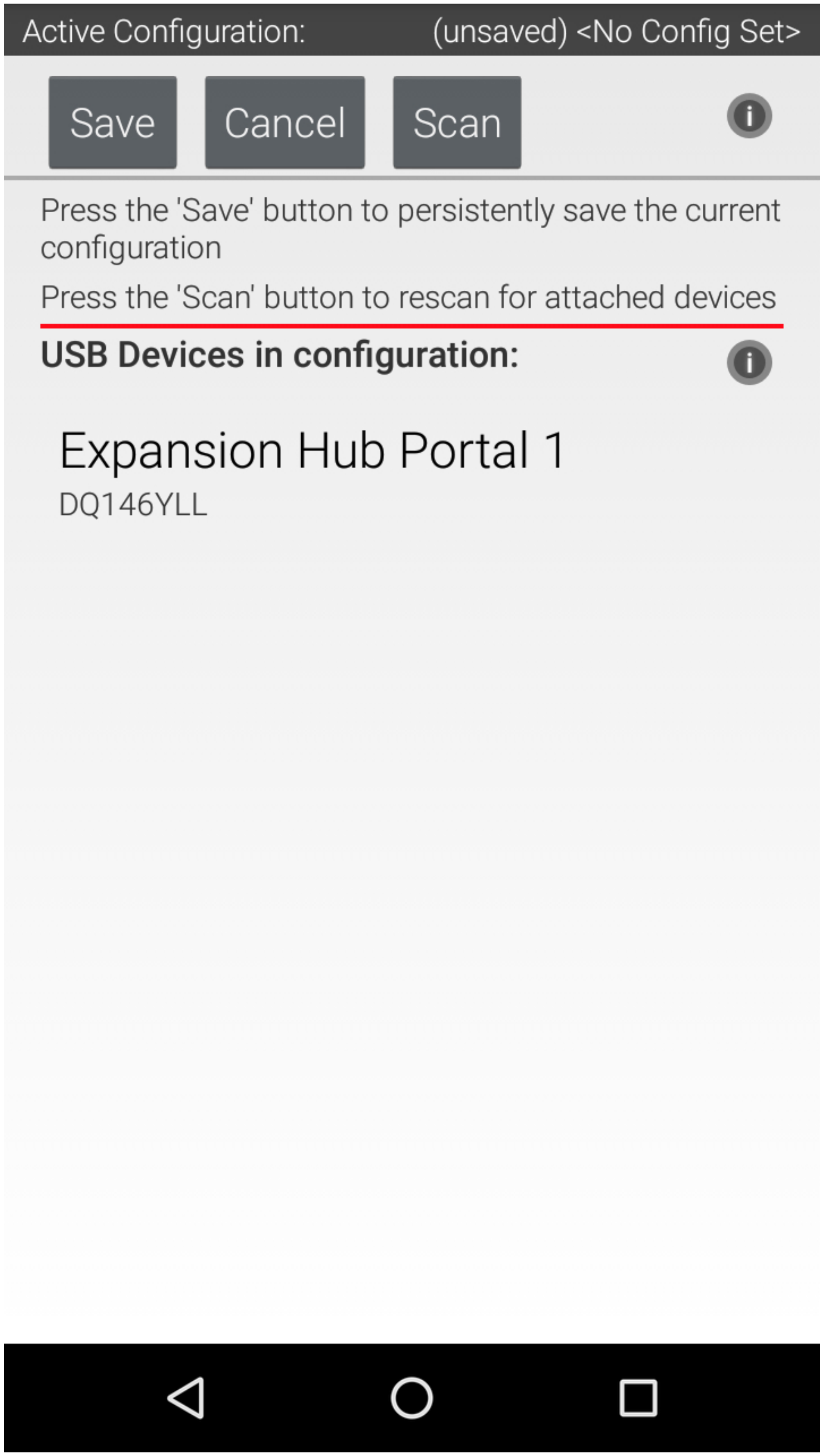
Connect the Robot Controller and select the Configure Robot option from the Settings menu. Press the New button to create a new configuration file. When you first scan for hardware, your Robot Controller should detect the Expansion Hub that is immediately connected to the Robot Controller via the OTG adapter and USB cable. The Robot Controller will automatically label this device as an Expansion Hub "Portal". The Robot Controller will communicate through this portal to the individual Expansion Hubs.

If you click on the Portal item in the configuration screen, you should see two Expansion Hubs listed, each with their respective addresses as part of their default device name.

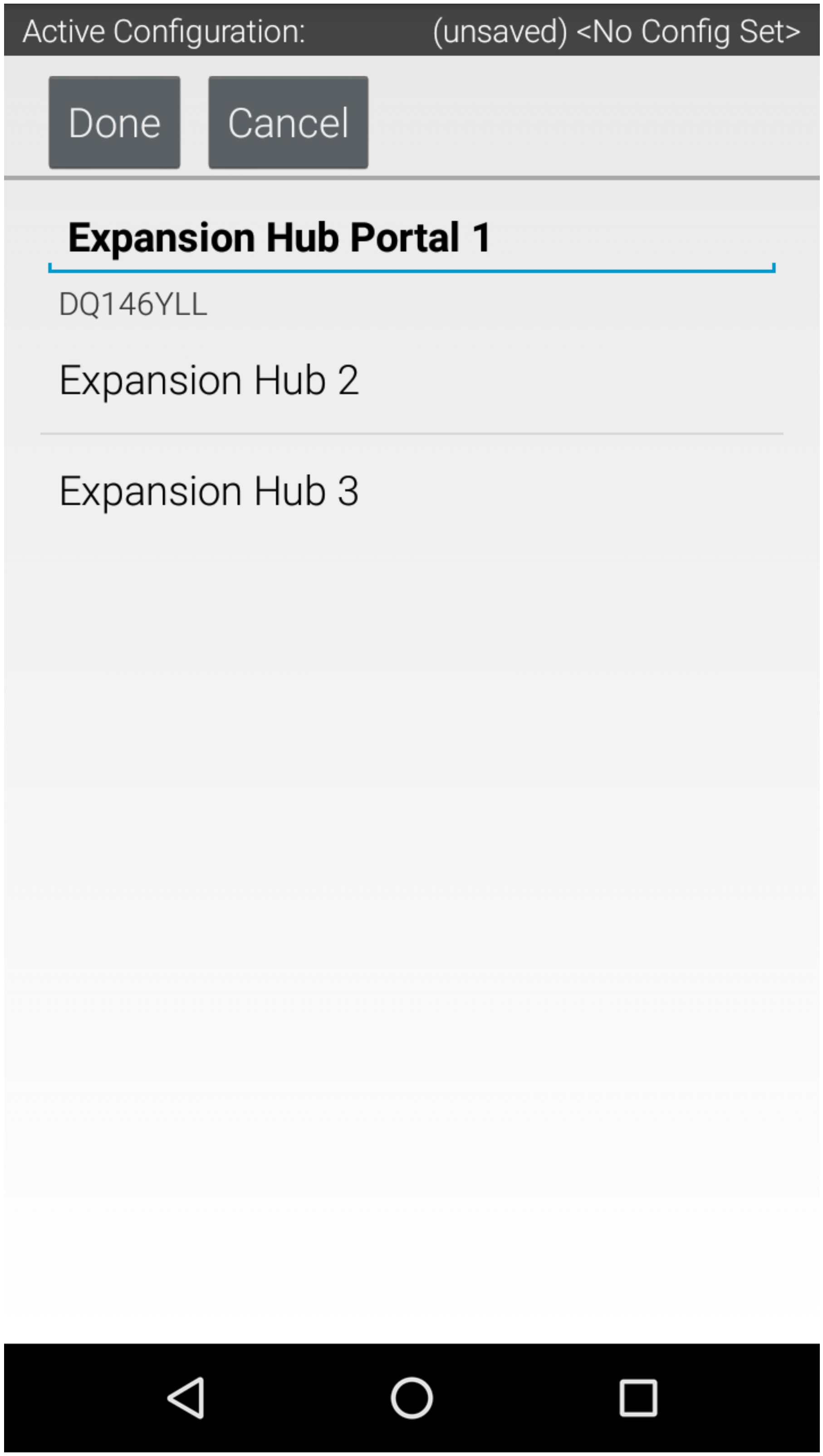
You can save this configuration file and return to the main screen of the Robot Controller. After the robot has been restarted, each Hub's LED should be blinking in the manner that indicates its individual address.

Congratulations, you are now ready to use your dual Expansion Hubs! You can configure and operate these Hubs as you would an individual Hub.


REV
EXPANSION HUB
PROUD SUPPORTER OF
FIRST
REV
EXPANSION HUB
PROUD SUPPORTER OF
FIRST
12 Volt 3,000 mAh



Active Configuration: (unsaved) <No Config Set>
Save
Cancel
Scan
Press the 'Save' button to persistently save the current configuration
Press the 'Scan' button to rescan for attached devices
USB Devices in configuration:
Expansion Hub Portal 1
DQ146YLL

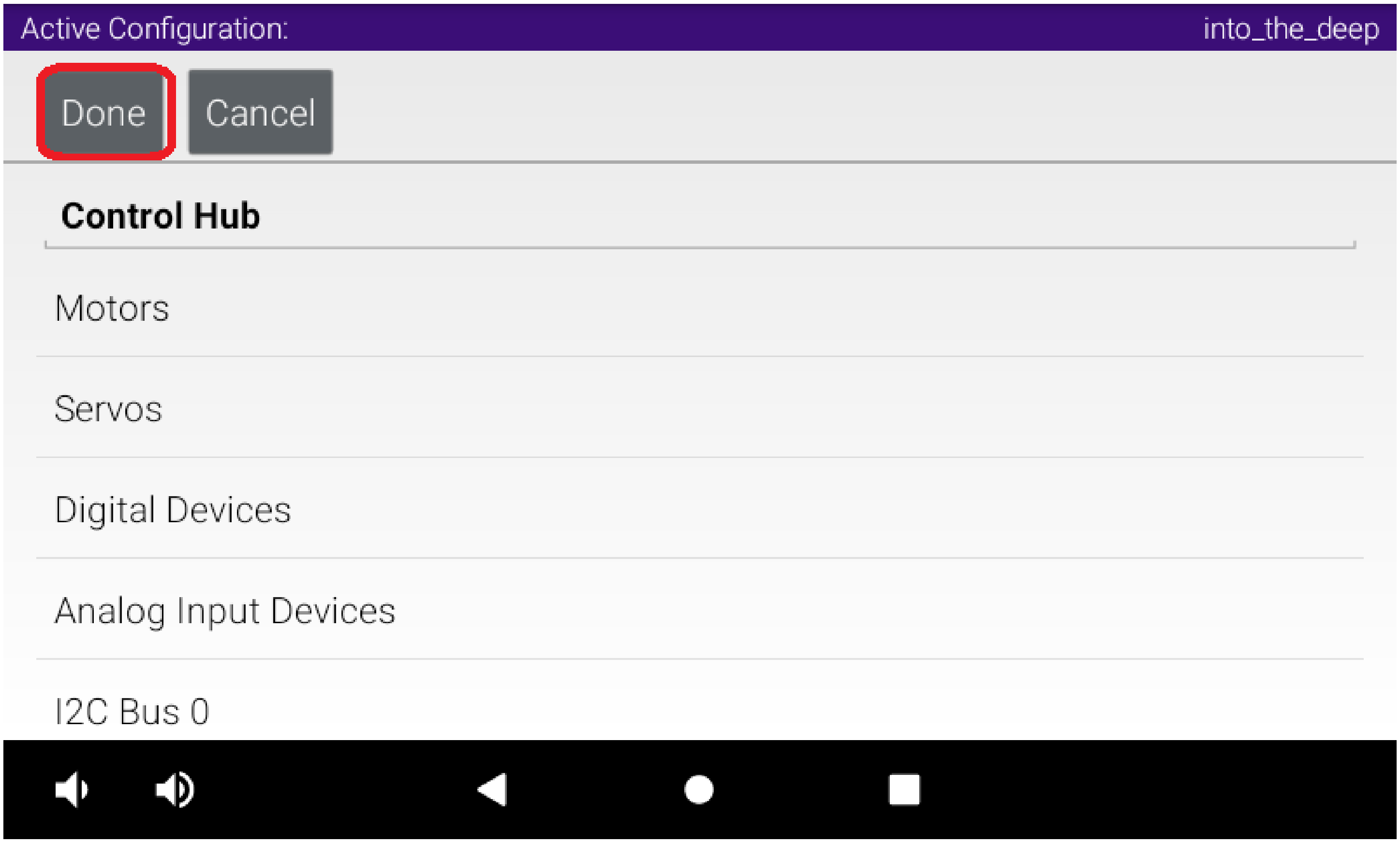

Active Configuration:
(unsaved) <No Config Set>
Done
Cancel
Expansion Hub Portal 1
DQ146YLL
Expansion Hub 2
Expansion Hub 3

19.2.9 Saving the Configuration Information

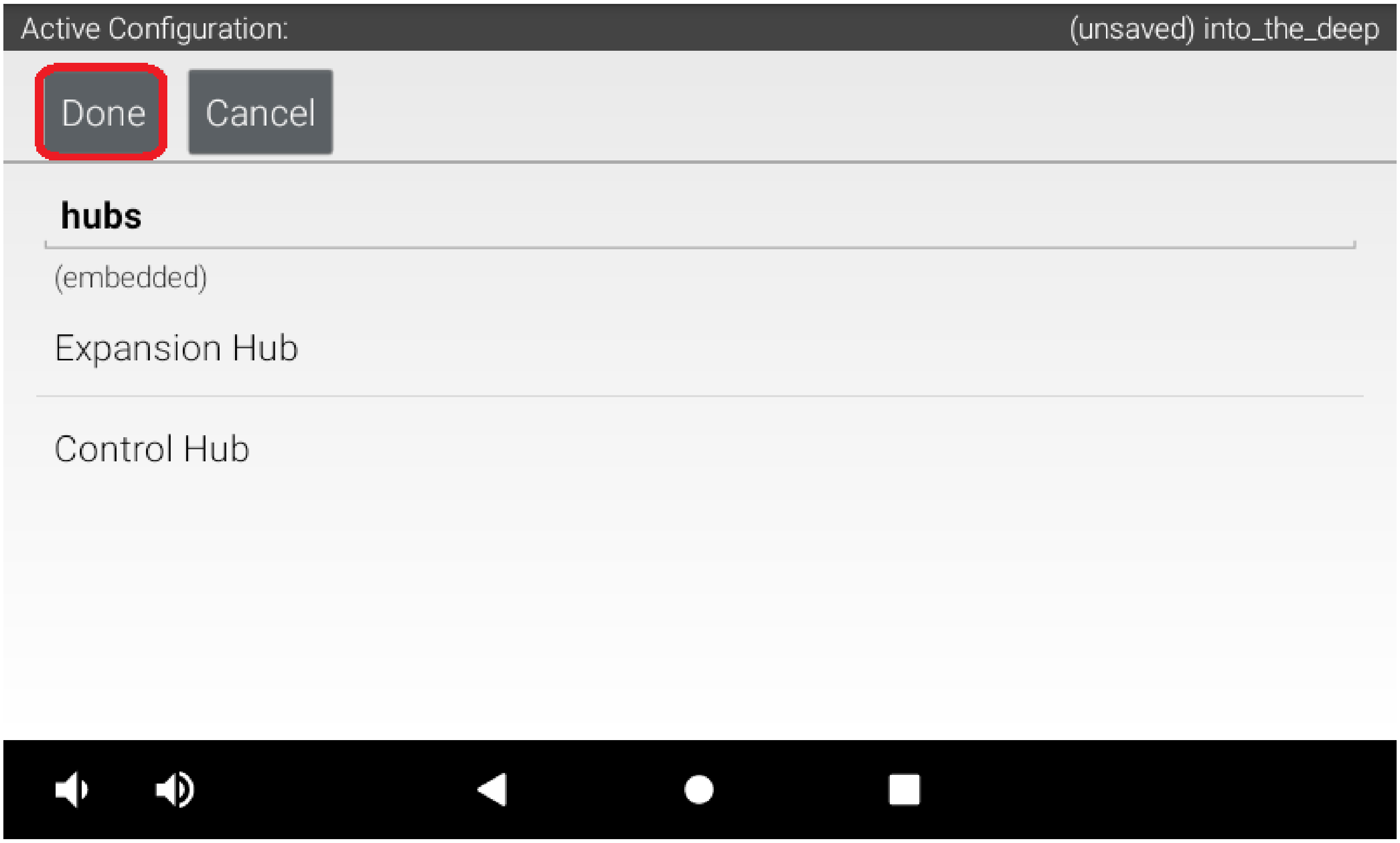
Once you have configured your hardware, you must save the information to the configuration file. If you do not save this information, it will be lost and the robot controller will be unable to communicate with your hardware.

Saving the Configuration Information Instructions

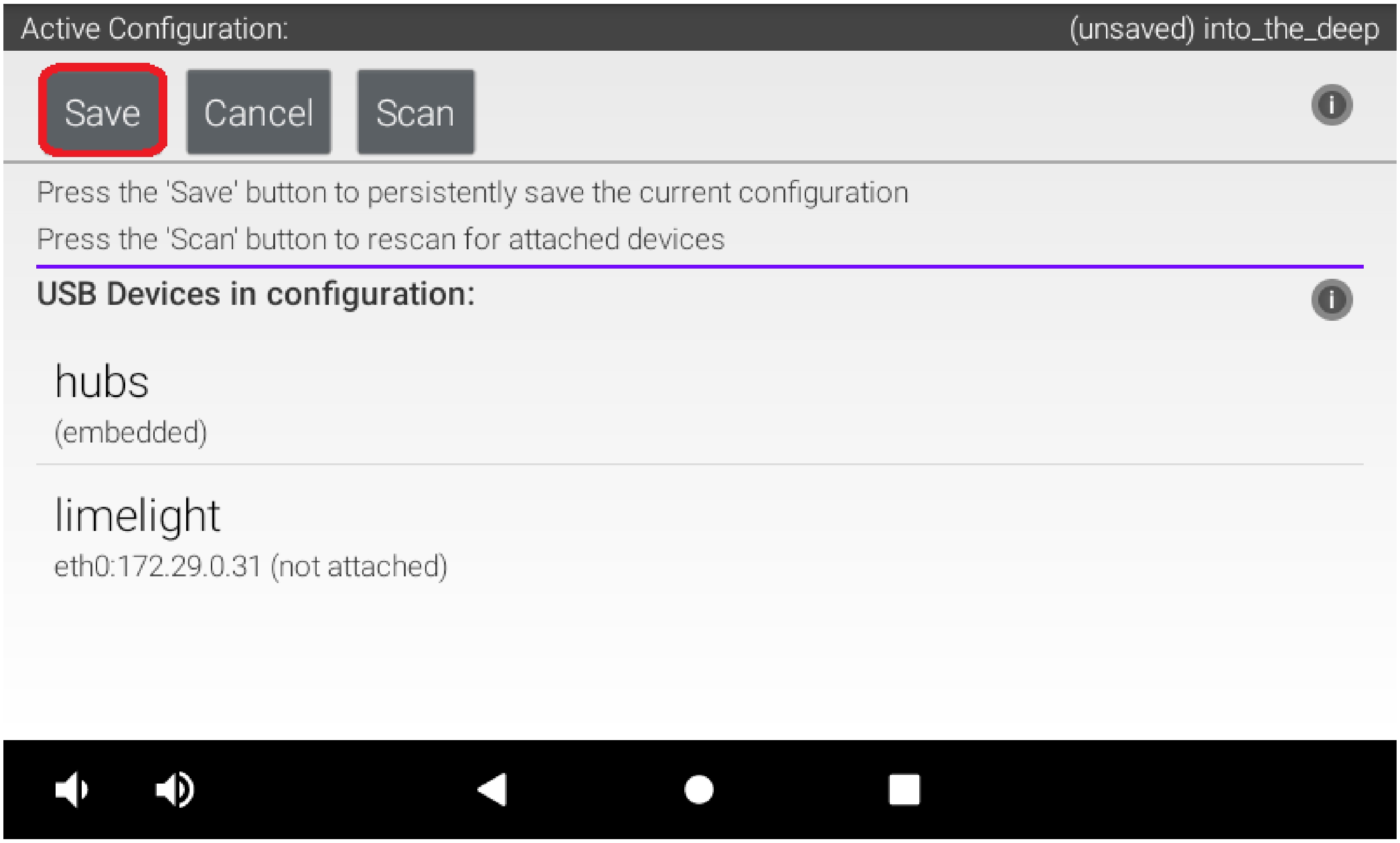
1. Press the **Done** button to go up one level in the configuration screens.



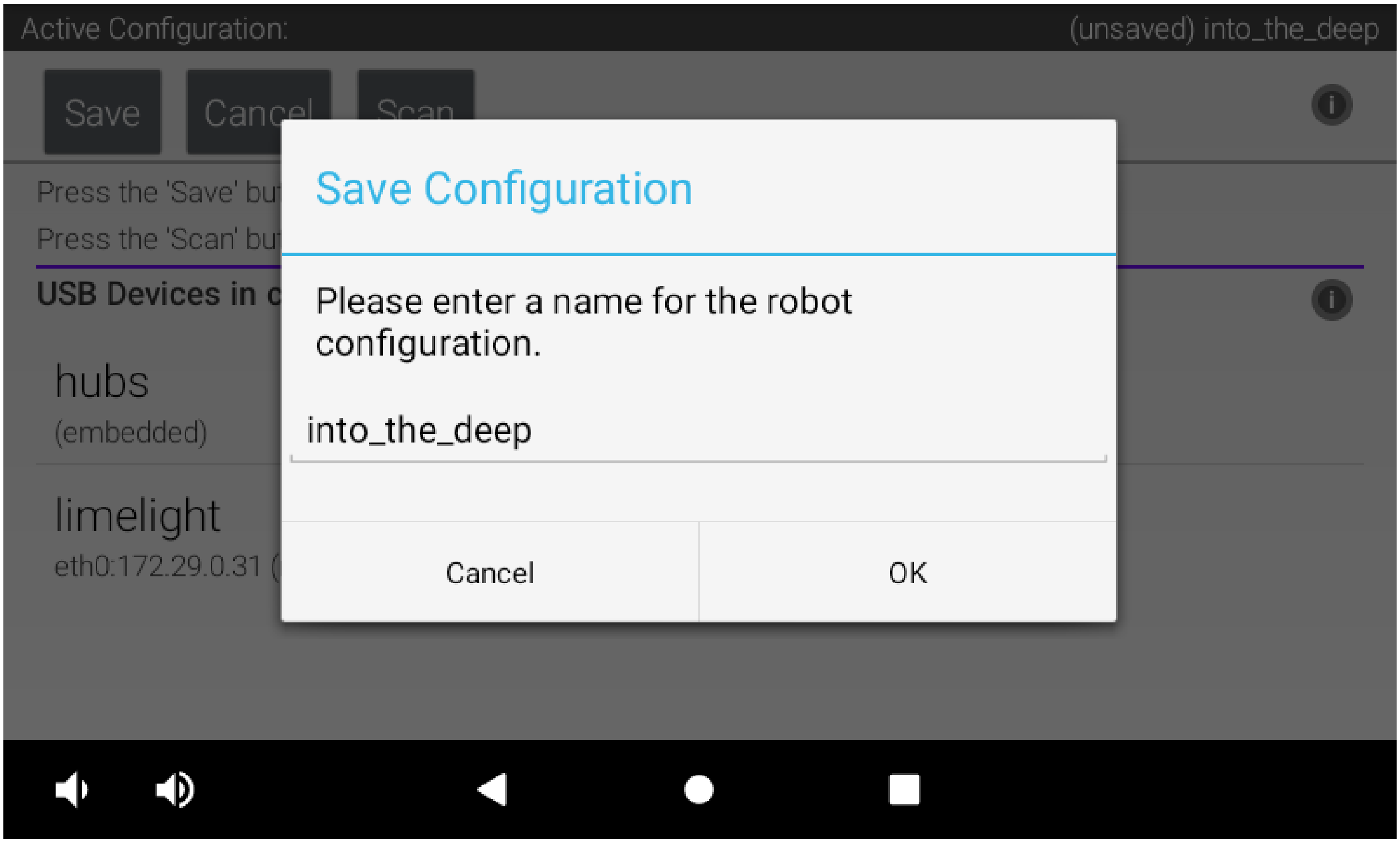
2. Press the **Done** button again to return to the highest level in the configuration screens.



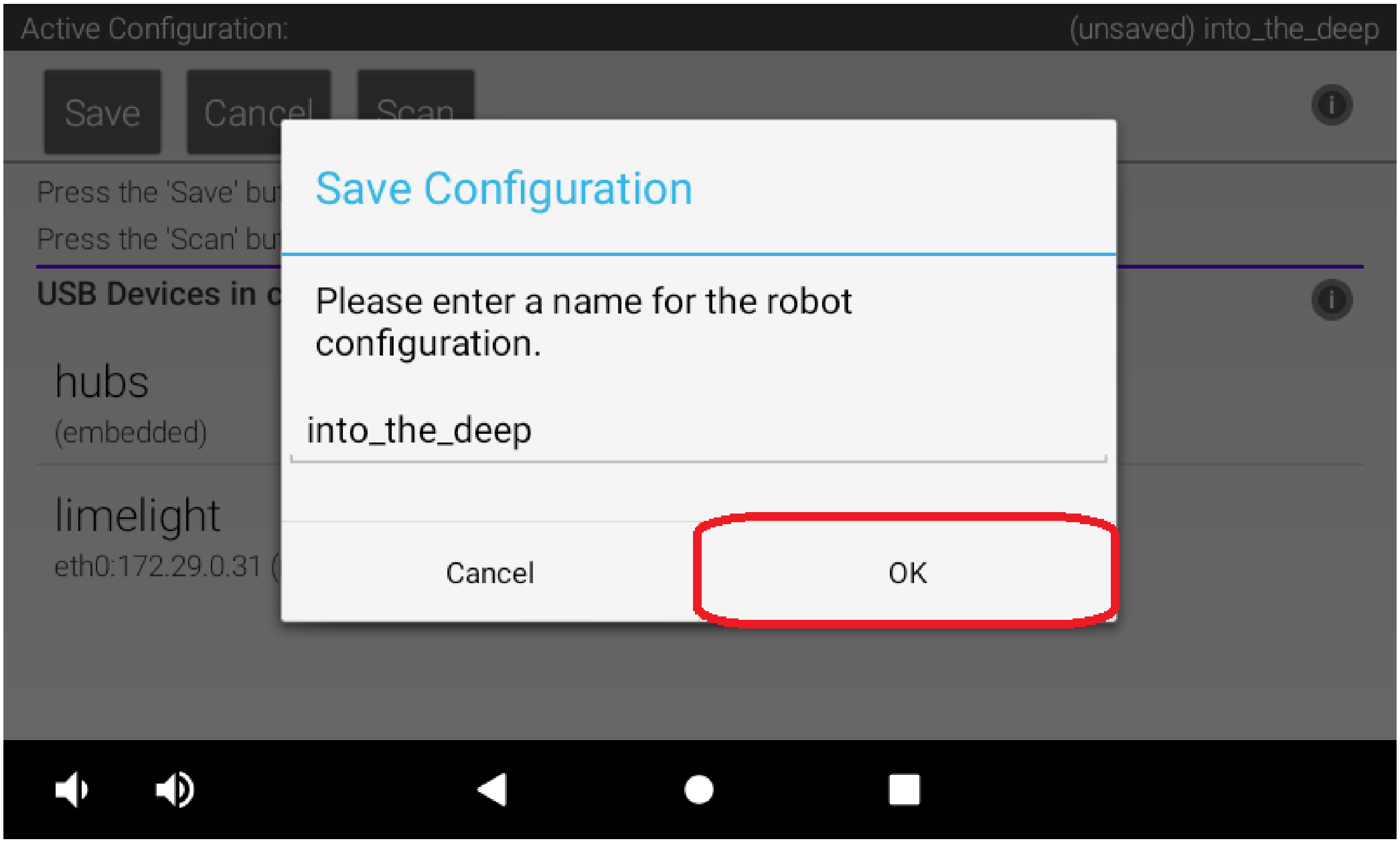
3. Press the **Save** button.



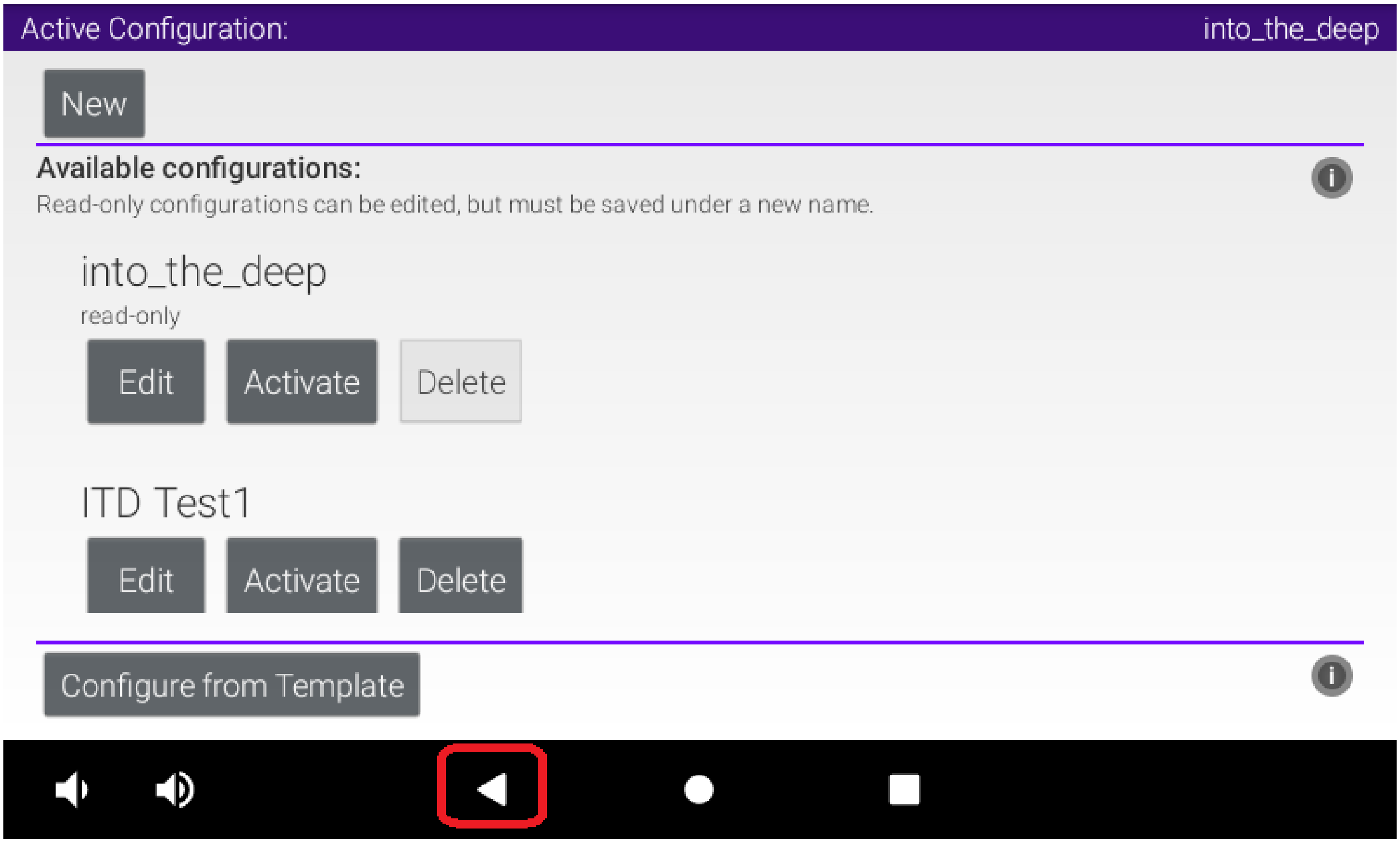
4. When prompted, specify a configuration file name using the touchscreen's keypad ("into_the_deep" for this example).



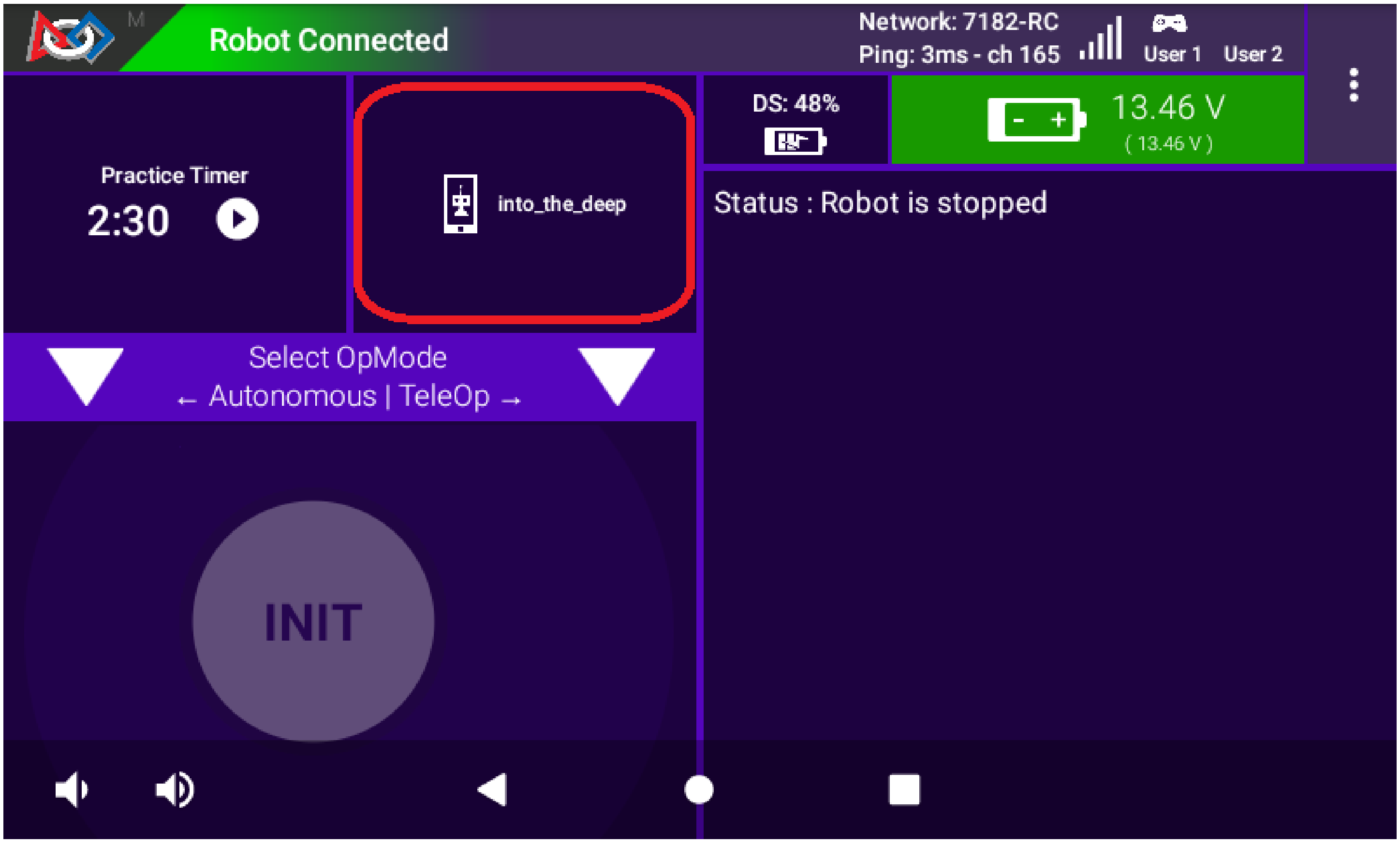
1. Press the **OK** button to save your configuration information using that file name.



6. After the configuration file has been saved, touch the Android back-arrow button to return to the main screen of the app.



7. Verify that the configuration file is the active configuration file on the main DRIVER STATION screen.


Robot Connected
Network: 7182-RC
Ping: 3ms - ch 165
User 1
User 2
DS: 48%
13.46 V
(13.46 V)
Practice Timer
2:30
into_the_deep
Status : Robot is stopped
Select OpMode
← Autonomous | TeleOp →
INIT

Chapter 20

FIRST *Tech Challenge Self-Inspect*

20.1 Introduction

This page describes the Self Inspect screens in the FTC Driver Station (DS) app and the FTC Robot Controller (RC) app.

A Self Inspect screen provides a snapshot of device status, as it relates to FTC rules for the control system. These rules are described in each season's Game Manual 1, and many are summarized in that manual's **Field Inspection Checklist**.

The Self Inspect screen is provided only as a quick, handy reference to help teams confirm that certain control system elements are up-to-date and properly configured. Self Inspect may be reviewed in Field Inspection at an FTC tournament, but is **not** a comprehensive or official standard of compliance with FTC rules.

Each inspection screen updates automatically, with or without a Restart Robot. This allows quick verification that issues have been resolved.

The challenge is to maximize useful information in a small screen. The Self Inspect layout and graphics evolve with FTC requirements; this page clarifies some of the brief but meaningful captions.

Note: These images show Version 7.0 of the FTC apps. Please refer to Game Manual Part 1 for the correct allowed software system versions.

20.2 Device Pairing

Pairing technology is a key aspect of Self Inspect reporting. Remember that RC phones host via **Wi-Fi Direct**, while Control Hubs host via **Standard, or 'infrastructure', Wi-Fi**.

In the DS app's Settings, the selected Pairing Method (Wi-Fi Direct or Control Hub) will influence acceptance/rejection in the DS Self Inspect report, as described in examples below.

RC and DS phones must have Airplane Mode **ON**, and Wi-Fi **ON** but not connected to any Standard/infra Wi-Fi host such as an internet router or hotspot. Devices should be set to **Forget** any local Wi-Fi networks.

FTC control devices may use these combinations:

- DS phone, RC phone
- DS phone, Control Hub
- Driver Hub, RC phone
- Driver Hub, Control Hub

A DS device (phone or Driver Hub) can display its own DS Self Inspect **and** an RC Self Inspect (for paired RC phone or Control Hub). An RC phone can display only its own RC Self Inspect.

This means that the Self Inspect screens can report as follows:

DRIVER STATION

- *DS Self Inspect 1*, on DS phone paired to RC phone
- *DS Self Inspect 2*, on DS phone paired to Control Hub
- *DS Self Inspect 3*, on Driver Hub paired to RC phone
- *DS Self Inspect 4*, on Driver Hub paired to Control Hub

ROBOT CONTROLLER

- *RC Self Inspect 1*, appearing on RC phone paired with DS phone
- *RC Self Inspect 2*, appearing on DS phone paired to RC phone
- *RC Self Inspect 3*, appearing on RC phone paired with Driver Hub
- *RC Self Inspect 4*, appearing on Driver Hub paired to RC phone
- *RC Self Inspect 5*, appearing on DS phone paired to Control Hub
- *RC Self Inspect 6*, appearing on Driver Hub paired to Control Hub

These combinations can display **slightly different** Self Inspect categories, status phrases, and pass/fail results. They are described below; click the **blue link** to explore the Self Inspect screen for that device and combination.

20.3 DS Self Inspect 1, on DS phone paired to RC phone

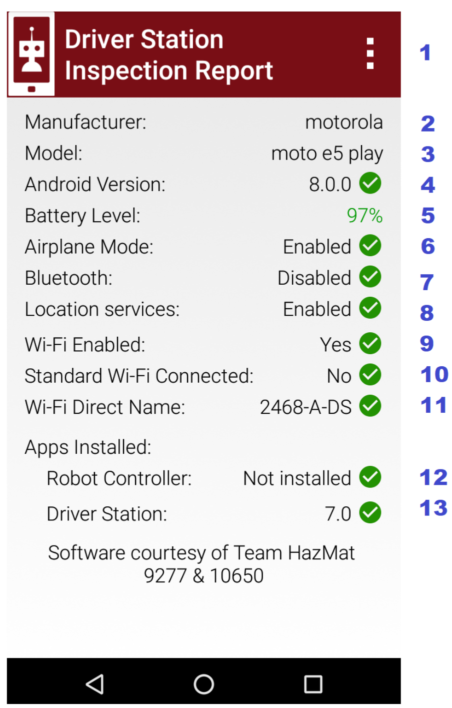


Fig. 1: DS Self Inspect 1, on DS phone paired to RC phone

- Item 1 is a menu with one choice: Disconnect from Wi-Fi Direct. It does work, but sometimes the apps re-pair automatically.

- Item 5 shows the battery level of the device being reported. Fun fact: the green color of the percentage value changes towards **orange** as the charge level goes down.
- Item 8 `Location services` appears only on devices running **Android 8** or higher. This is an SDK/Android technology requirement, not an FTC rule.
- Items 9 and 10 here must be Yes and No. `Wi-Fi Enabled` means the DS device's Wi-Fi radio is **ON**, to use Wi-Fi Direct. It must **not** be connected to a Standard/infrastructure Wi-Fi source, such as an internet router or a Control Hub, when intending to pair with an RC phone.
- Item 11 indicates whether the **device's Wi-Fi Direct name** meets FTC format requirements. It does not check whether the paired device has a matching RC name (team number). In this case, the phones are legally named 2468-A-DS and 2468-A-RC. DS Settings (Driver Station Name) allows only FTC-legal names, but any name can be entered in the DS phone's Android Wi-Fi Direct settings.
- Item 12 verifies that a DS device does **not** have an RC app installed.
- Item 13 ensures the DS app meets the minimum version **for the current FTC season**, based on the device's system date. An "incorrect" red mark here can be cleared by correcting the date in the **Android device Settings**.

Here's a report from the same phone, with many items **rejected** by Self Inspect.

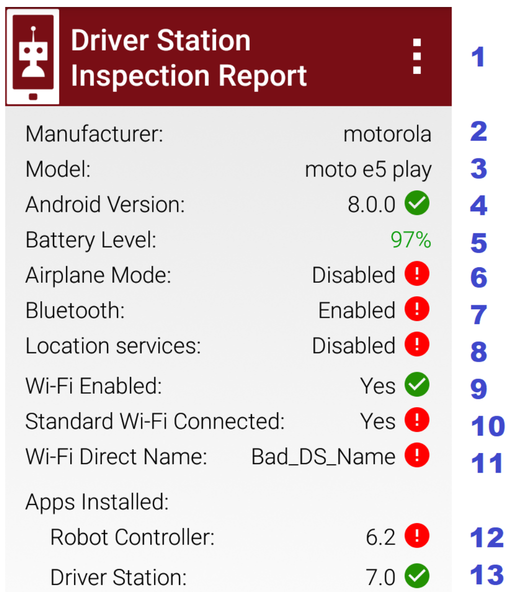


Fig. 2: DS Self Inspect 1, on DS phone – with issues!

- Item 6 rejects `Airplane Mode` for being **OFF**; it must be on for FTC phones. This is an Android system setting, available at the phone's Settings menu, or easily accessed by swiping down twice from the top of the screen. Turning on Airplane Mode automatically turns off the Wi-Fi radio, as an Android 'convenience'. An FTC user will need to manually turn the Wi-Fi radio back on (although not connected to a local hotspot or internet router).
- Item 7 rejects `Bluetooth` for being **ON**; it must be off for FTC use. This is also an Android system setting; swipe down twice or see Settings menu.
- Item 8 rejects `Location services` for being **OFF**. For devices running **Android 8** or higher, the FTC apps require Location enabled. This is also an Android system setting; swipe down twice or see Settings menu.
- Item 9 shows the DS phone's Wi-Fi radio is **ON**, as required for Wi-Fi Direct **or** Standard Wi-Fi pairing to an RC device.
- Item 10 rejects the DS phone's connection via Standard/infrastructure Wi-Fi, because the DS Pairing Method is set to Wi-Fi Direct – thus intending to connect with an RC phone. In this case, the phone is connected to a home Wi-Fi network. This and other such networks must be set to **Forget**, in the device's Android Wi-Fi menu. If temporary internet access is needed, 'Forget' that network afterwards. Also **Remove Account** for any Google or other accounts that may

have been used during the internet session. Such accounts can cause background activity, notifications and updates – at the worst possible times.

- Item 11 rejects the device's **Wi-Fi Direct name** for not meeting FTC format requirements. The bad name shown here was created in the DS phone's Android Wi-Fi Direct settings; not possible using the app's DS Settings (Driver Station Name).
- Item 12 rejects the presence of an RC app installed on this DS device. The rejection is not for the older version (6.2), but simply for being an RC app.

20.4 DS Self Inspect 2, on DS phone paired to Control Hub

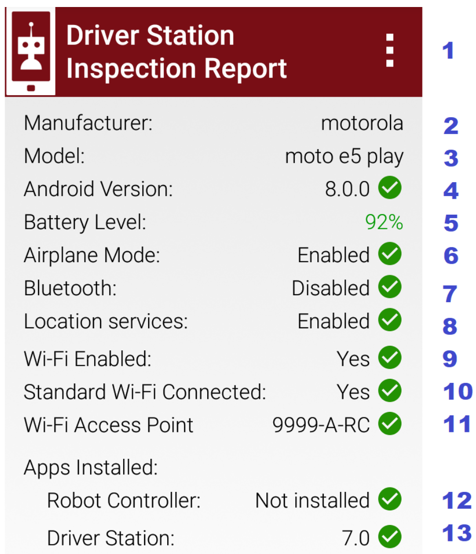


Fig. 3: DS Self Inspect 2, on DS phone paired to Control Hub

The same points apply as for DS Self Inspect 1 (immediately above), except:

- Items 9 and 10 must now be Yes and Yes. The DS phone's Wi-Fi radio is **ON**, and connected via Standard/infra Wi-Fi. It does not indicate **what** the DS phone is connected to; that's covered by Item 11.
- Item 10's Yes would be **rejected** if the DS Pairing Method was set to Wi-Fi Direct – thus intending to connect with an RC phone.
- Item 11 shows the Standard Wi-Fi **network name**, or Access Point (AP), that the DS phone is connected to. The check-mark indicates the AP is an FTC legal device (Control Hub) and has a correctly formatted name. This does **not** check that the DS and RC names match (team number). In fact, this phone is 2468-A-DS and this Control Hub is 9999-A-RC, an **illegal combination** to be flagged by the FTC team or the Field Inspector.

20.5 DS Self Inspect 3, on Driver Hub paired to RC phone

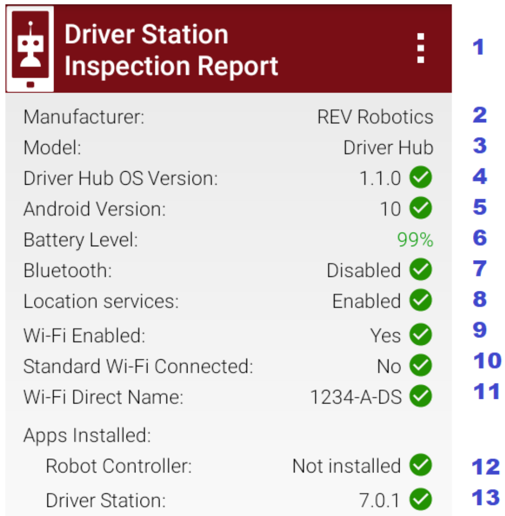


Fig. 4: DS Self Inspect 3, on Driver Hub paired to RC phone

- Item 4 appears only on the Driver Hub. The check-mark indicates the Operating System meets the minimum version requirement in the FTC Game Manual.
- Note that `Airplane Mode` has been omitted from the DS inspection, only for Driver Hub. FTC rules exclude the Driver Hub and Control Hub from the Airplane Mode requirement.
- Item 8 `Location services` appears only on devices running **Android 8** or higher. This is an SDK/Android technology requirement, not an FTC rule.
- Items 9 and 10 must be Yes and No. `Wi-Fi Enabled` means the Driver Hub's Wi-Fi radio is **ON**, to use Wi-Fi Direct for the RC phone. The Driver Hub is technically **able to also be connected** to a Standard/infrastructure Wi-Fi source, including an internet router or a Control Hub. Item 10 confirms this is not happening; see next example.
- Item 10's No would be **rejected** if the DS Pairing Method was set to Control Hub.
- Item 11 indicates whether the **device name** meets FTC format requirements. It does not check whether the paired device has a matching RC name (team number).
- Item 12 verifies that the Driver Hub does **not** have an RC app installed.
- Item 13 ensures the DS app meets the minimum version **for the current FTC season**, based on the device's system date. This particular version 7.0.1 does not exactly match the RC phone's 7.0. Such a "Point mismatch" is allowed under updated FTC rules (was Q&A #176 for 2021-2022 season). Otherwise, an "incorrect" red mark here can be cleared by correcting the date in the **Android device Settings**.

This Self Inspect screen appeared while the Driver Hub was paired to an RC phone, then was *also* connected to to a Control Hub via Standard Wi-Fi. The DS home screen temporarily showed "Connected" (to RC phone) and "No Heartbeat", then recovered its pairing to the RC phone.

- Item 10 shows the discrepancy. The DS app soon closes this Standard Wi-Fi connection, allowing the Driver Hub to remain paired only with the RC phone.

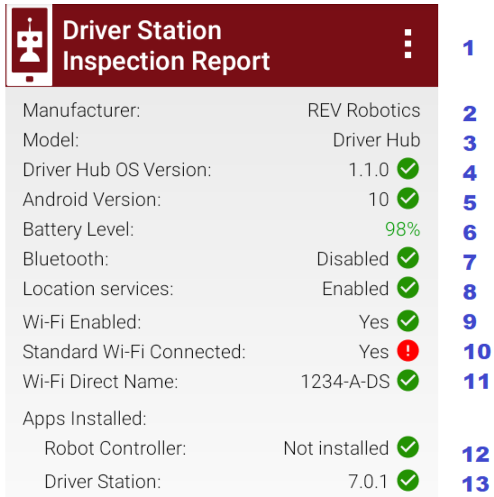


Fig. 5: DS Self Inspect 3, on Driver Hub paired to RC phone

20.6 DS Self Inspect 4, on Driver Hub paired to Control Hub

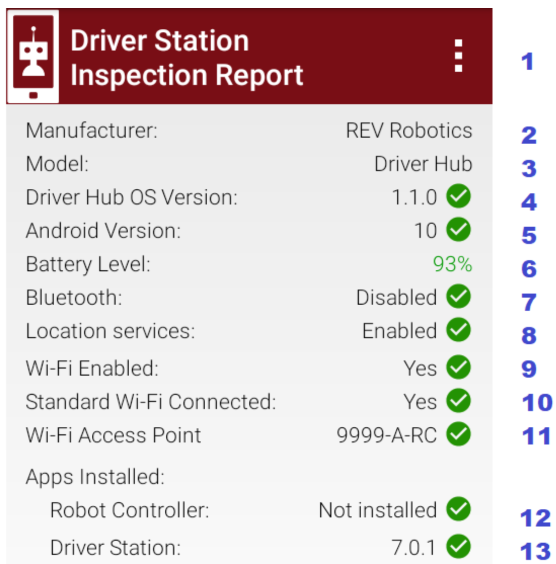


Fig. 6: DS Self Inspect 4, on Driver Hub paired to Control Hub

- Item 1 still offers one choice, "Disconnect from Wi-Fi Direct". But now, touching that selection gives this message "There was an error disconnecting from Wi-Fi Direct". That's because the Driver Hub is paired to a Control Hub, thus **not** via Wi-Fi Direct.
- Item 10's Yes would be **rejected** if the DS Pairing Method was set to Wi-Fi Direct – thus intending to connect with an RC phone.
- Item 11 shows the Standard Wi-Fi **network name**, or Access Point (AP), that the Driver Hub is connected to. The check-mark indicates the AP is an FTC legal device (Control Hub) and has a correctly formatted name. This does **not** check that the DS and RC names match (team number). In fact, this Driver Hub is 1234-A-DS and this Control Hub is 9999-A-RC, an **illegal combination** to be flagged by the FTC team or the Field Inspector.

This Self Inspect screen appeared after the Driver Hub was paired to a Control Hub, then was connected to a Wi-Fi internet router.

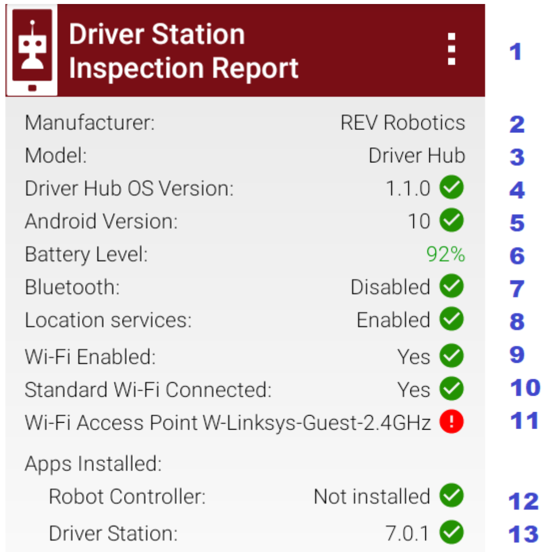


Fig. 7: DS Self Inspect 4, on Driver Hub paired to Control Hub

- Item 11 shows the error. The Driver Hub can connect via Standard Wi-Fi to only one AP at a time; this network is not an FTC RC device.

20.7 RC Self Inspect 1, appearing on RC phone paired with DS phone

Now we change to **Robot Controller** Self Inspect screens. Again, RC screens can be viewed from the DS device **or** from an RC phone, with slight differences.

- Item 5 lists the Expansion Hub addresses and firmware levels. This example shows one Expansion Hub, but two can be listed here. A check-mark indicates all firmware is up-to-date based on the current version of the RC app. This item shows "N/A" if no Hubs are connected.
- Item 10 RC `Password` appears only in RC Self Inspect, not in DS Self Inspect. It checks the FTC requirement for a Control Hub password different than the factory default ("password"). Although aimed only at the Control Hub, this item does appear on RC phones (as here) which don't have a default password and thus always get the check-mark.
- Item 14 ensures the RC app meets the minimum version **for the current FTC season**, based on the device's system date. It does not check for a match with the DS app version. An "incorrect" red mark here can be cleared by correcting the date in the **Android device Settings**.
- Item 15 verifies that the RC device does **not** have an DS app installed.

20.8 RC Self Inspect 2, appearing on DS phone paired to RC phone

This RC Self Inspect screen displayed on the paired DS phone is the "same" as the previous one on the RC phone, with two differences:

- The 3-dots menu is missing from the header. This menu offered a single choice, to disconnect the Wi-Fi Direct. But this cannot be performed as an RC action, from a DS phone connected by that same Wi-Fi Direct.
- Item 14 did not appear on the RC phone's display of this RC Self Inspect. Here is the verification that the DS app and RC app have matching versions; in this case both apps are version 7.0. Any "Point mismatch" (e.g. 7.0 vs. 7.0.1) is allowed under updated FTC rules (was Q&A #176 for 2021-2022 season).

Fig. 8: RC Self Inspect 1, appearing on RC phone paired with DS phone

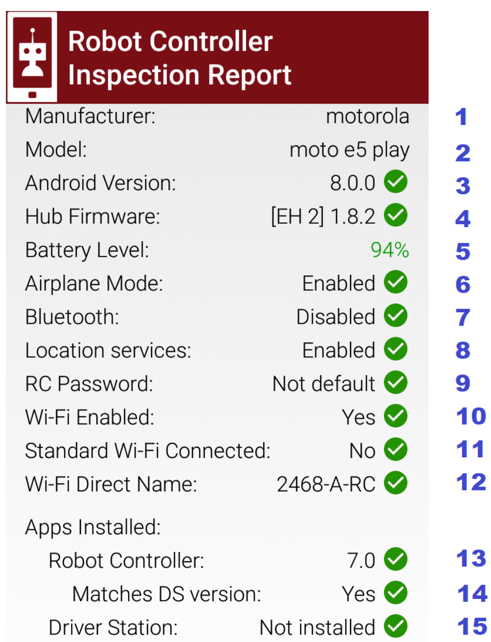


Fig. 9: RC Self Inspect 2, appearing on DS phone paired to RC phone

20.9 RC Self Inspect 3, appearing on RC phone paired with Driver Hub

Fig. 10: RC Self Inspect 3, appearing on RC phone paired with Driver Hub

The above screen is the same as RC Self Inspect 1, where the DS device is a DS phone. See the notes there.

This is also the same screen, except the RC phone was connected to an internet router, while paired with a Driver Hub. The Standard Wi-Fi connection caused the RC phone to temporarily lose that pairing, which was able to be restored.

- Item 12 shows the rejection: connected via Standard Wi-Fi, but **not** to an FTC DS device.

20.10 RC Self Inspect 4, appearing on Driver Hub paired to RC phone

This display on a paired Driver Hub is the "same" RC Self Inspect screen as the one immediately above, but there are two differences:

- The 3-dots menu is missing from the header. This menu offered a single choice, to disconnect the Wi-Fi Direct. But this cannot be performed as an RC action, from a Driver Hub connected by that same Wi-Fi Direct.
- Item 14 did not appear on the RC phone's display of this RC Self Inspect. Here is the check for matching versions of the DS app and RC app. In this case, the DS app is 7.0.1 and the RC app is 7.0, rejected here as a mismatch. Such a "Point mismatch" is allowed under updated FTC rules (was Q&A #176 for 2021-2022 season).

Fig. 11: RC Self Inspect 3, appearing on RC phone paired with Driver Hub

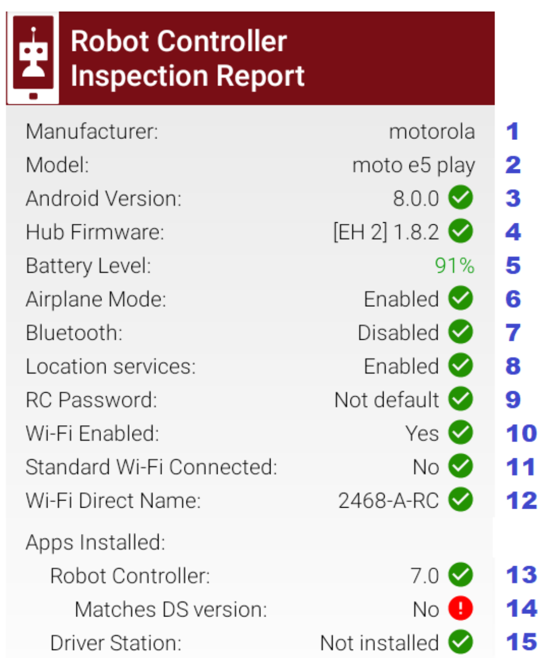


Fig. 12: RC Self Inspect 4, appearing on Driver Hub paired to RC phone

20.11 RC Self Inspect 5, appearing on DS phone paired to Control Hub

Looking now at the **Control Hub**, the Self Inspect screen has a few differences. In this example, the robot is configured with **two** Hubs.

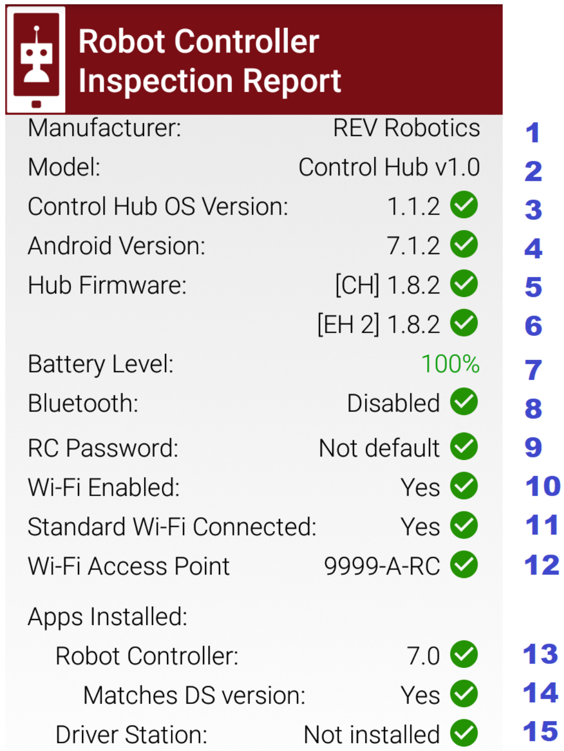


Fig. 13: RC Self Inspect 5, appearing on DS phone paired to Control Hub

- Again the 3-dots menu is missing from the header. This menu offered a single choice, to disconnect the Wi-Fi Direct. But the Control Hub hosts with Standard Wi-Fi, not with Wi-Fi Direct. In any case, the connection cannot be managed as an RC action, from a DS phone using that same connection.
- Item 3 appears only on RC Self Inspect screens for Control Hub. It verifies the Operating System is up-to-date for the current version of the RC app.
- `Location services` does **not** appear here, since the Control Hub's Android version (Item 4) is **lower** than Android 8.
- Item 5 shows the firmware version of the Expansion Hub embedded in the Control Hub; it's up-to-date for the current version of the RC app.
- Item 6 shows the firmware version and address of the standalone Expansion Hub, also up-to-date.
- Item 7 should always show a high battery charge here, indicating at least the nominal 12V charge level from the robot battery.
- Note that `Airplane Mode` has been omitted from the RC inspection, only for Control Hub. FTC rules exclude the Driver Hub and Control Hub from the Airplane Mode requirement.
- Item 9 does apply here to the Control Hub. Its password must be changed from the factory default ("password").
- Items 10 and 11 should be Yes and Yes for Control Hub, which uses only Standard/infra Wi-Fi. Item 11 does not indicate **what** the Control Hub is connected to (but it must be the DS phone displaying this screen).
- Item 12 shows the Standard Wi-Fi **network name**, or Access Point (AP), that is broadcast by the Control Hub. The check-mark indicates the AP has a correctly formatted FTC name. This does **not** check that the DS and RC names

match (team number). In fact, this DS phone is 2468-A-DS and this Control Hub is 9999-A-RC, an **illegal combination** to be flagged by the FTC team or the Field Inspector.

- Item 14 appears only on DS displays of RC Self Inspect. Here is the check for matching versions of DS app and RC app; in this case both apps are version 7.0. Any "Point mismatch" (e.g. 7.0 vs. 7.0.1) is allowed under updated FTC rules (was Q&A #176 for 2021-2022 season).
- Item 15 verifies that an RC device does **not** have an DS app installed. This would be quite a mistake for a Control Hub, lacking an onboard screen.

20.12 RC Self Inspect 6, appearing on Driver Hub paired to Control Hub

For a Control Hub, the Self Inspect categories displayed on Driver Hub are the same as on DS phone, immediately above.

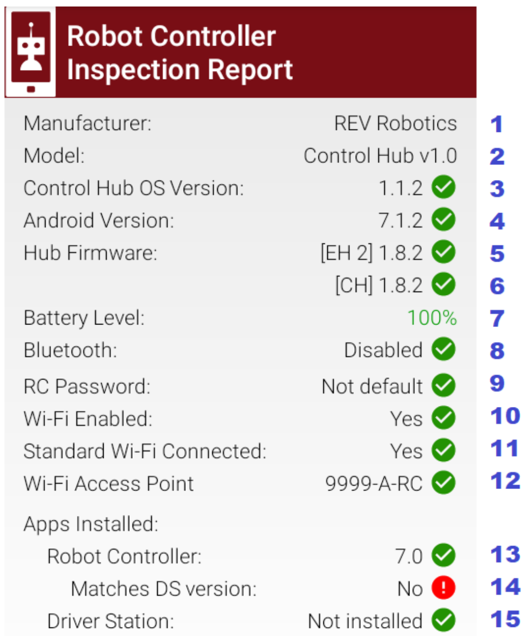


Fig. 14: RC Self Inspect 6, appearing on Driver Hub paired to Control Hub

The only reporting difference here is the 'mismatch' between the Driver Hub's DS app version of 7.0.1 and the Control Hub's 7.0. This is likely to happen since Driver Hubs are typically auto-updated, in this case to a DS version intended only for old Android 6 phones. Such a "Point mismatch" is allowed under updated FTC rules (was Q&A #176 for 2021-2022 season).

Lastly… with no active connection, a DS device cannot display any information about the RC device status.

20.13 Summary

The Self Inspect screen is a quick, handy reference to help teams confirm that certain control system elements are up-to-date and properly configured.

Self Inspect may be reviewed in Field Inspection at an FTC tournament, but is **not** a comprehensive or official standard of compliance with FTC rules.

Each inspection screen updates automatically, with or without a Restart Robot. This allows quick verification that issues have been resolved.

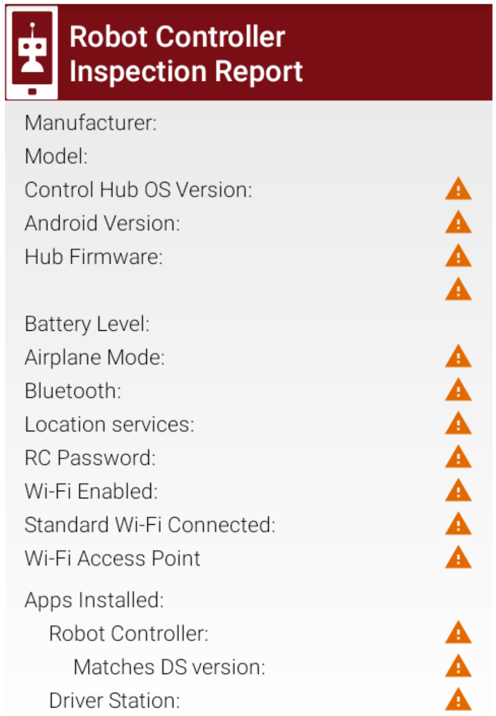


Fig. 15: RC Self Inspect 6, appearing on Driver Hub previously paired to Control Hub

Questions, comments and corrections to westsiderobotics@verizon.net

Chapter 21

Programming Resources

This page contains programming tutorials and related Control System documentation useful for configuring and programming Control System components.

21.1 Programming Tutorials

FIRST Tech Challenge Programming Tutorials

- *Choosing a Programming Tool*
- *Blocks Tutorial*
- *Onbot Java Tutorial*
- *Android Studio Tutorial*

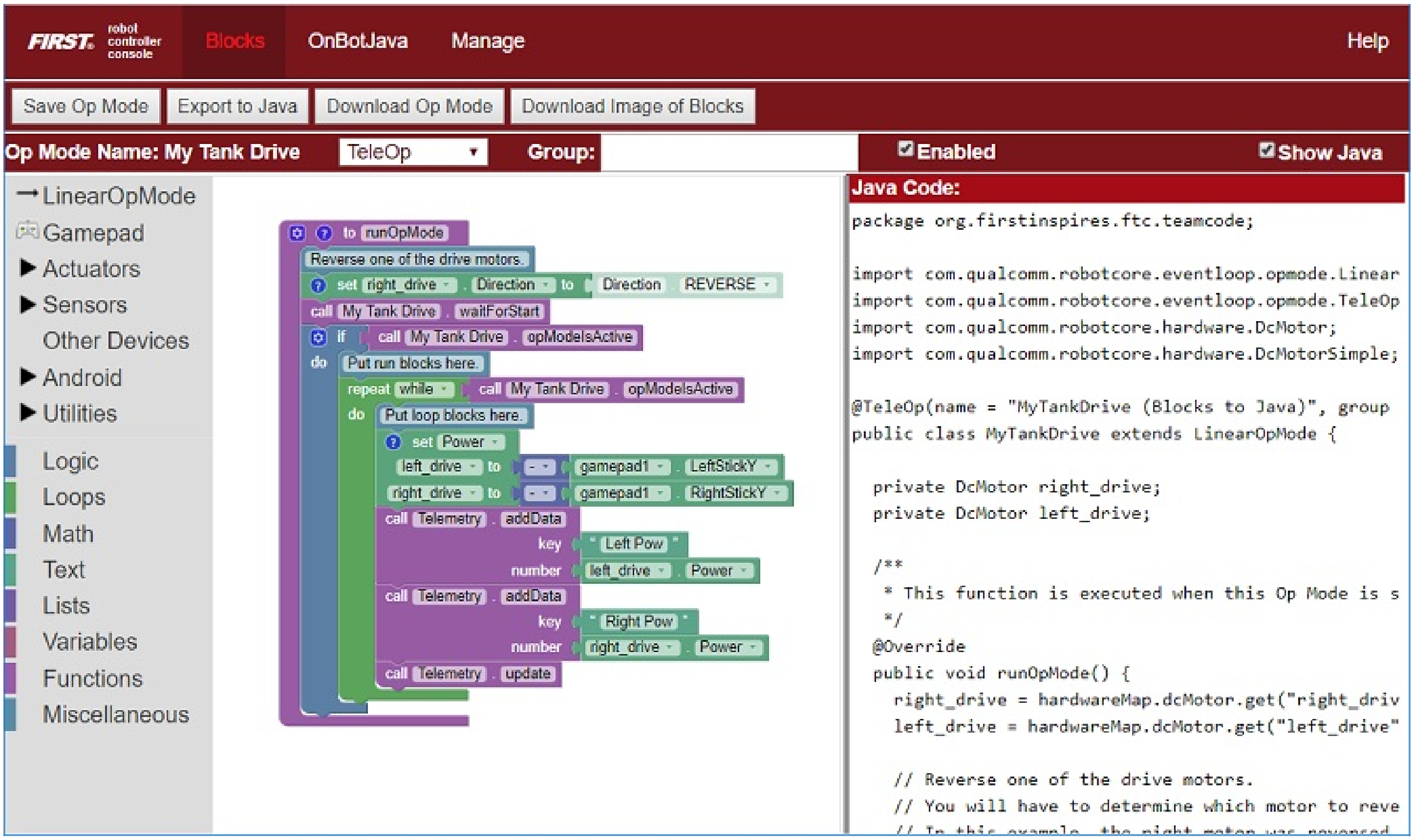
21.1.1 Choosing a Programming Tool

You need to select a programming tool to be able to create op modes for your competition robot. An Op Mode or Operational Mode is program that tells the robot what to do. There are three programming tools that are available for you to use.

FIRST strongly recommends that **all users** begin by learning how to use the *Blocks programming tool*.

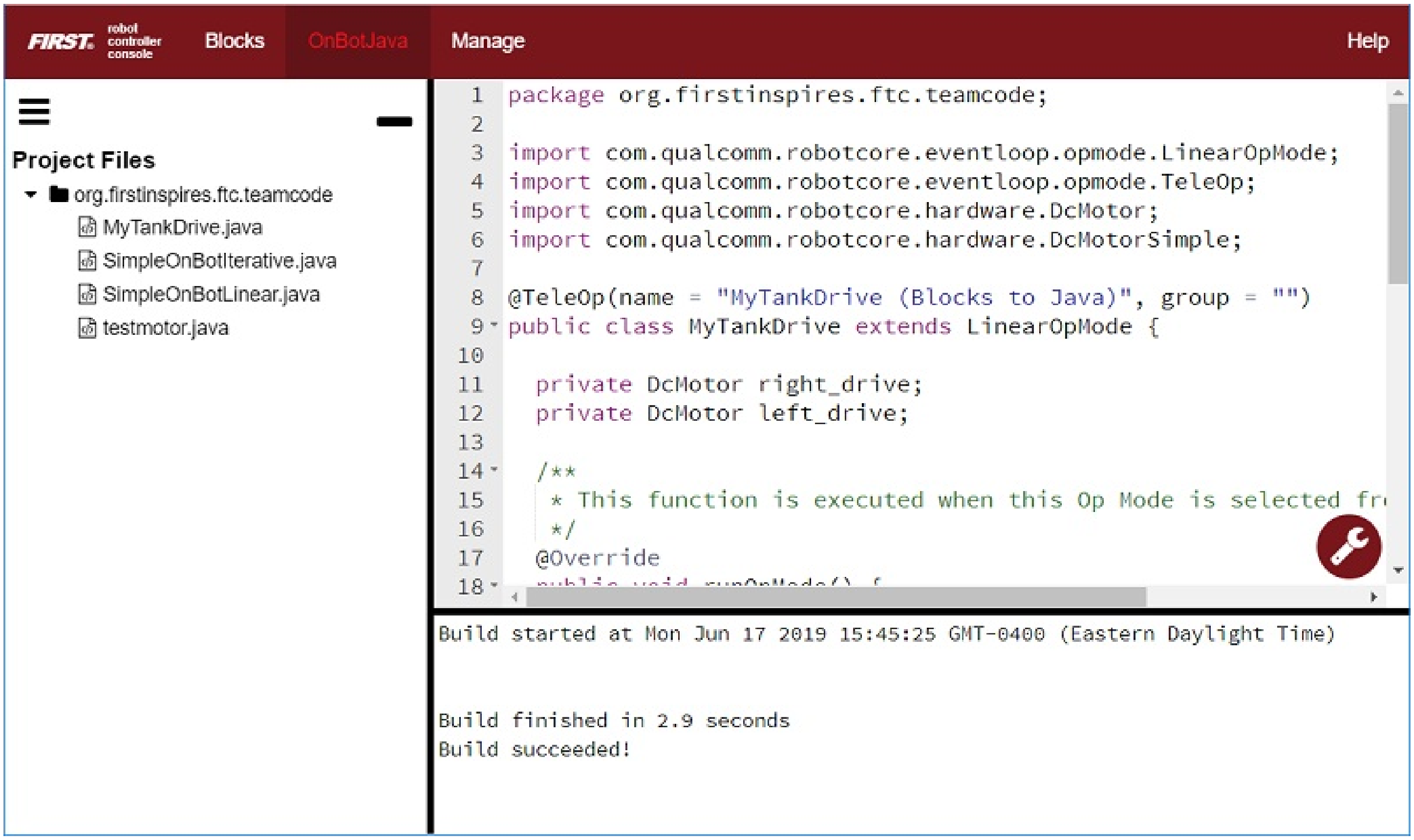
The Blocks Programming Tool

A visual programming tool that lets programmers use a web browser to create, edit and save their op modes. This tool is recommended for novice programmers and for users who prefer to design their op modes visually, using a drag-and-drop interface.



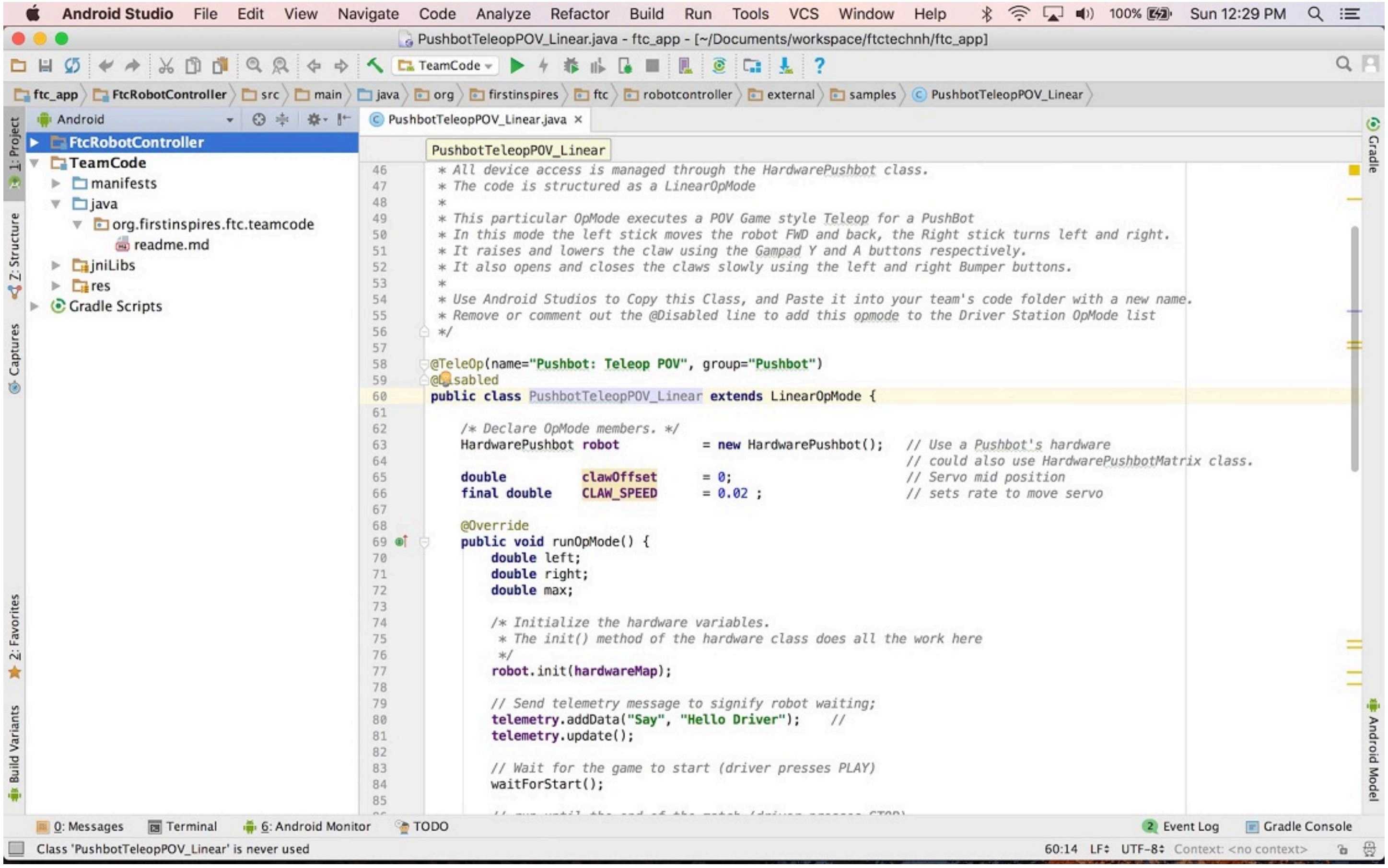
The OnBot Java Programming Tool

A text-based programming tool that lets programmers use a web browser to create, edit and save their Java op modes. This tool is recommended for programmers who have basic to advanced Java skills and who would like to write text-based op modes.



Android Studio

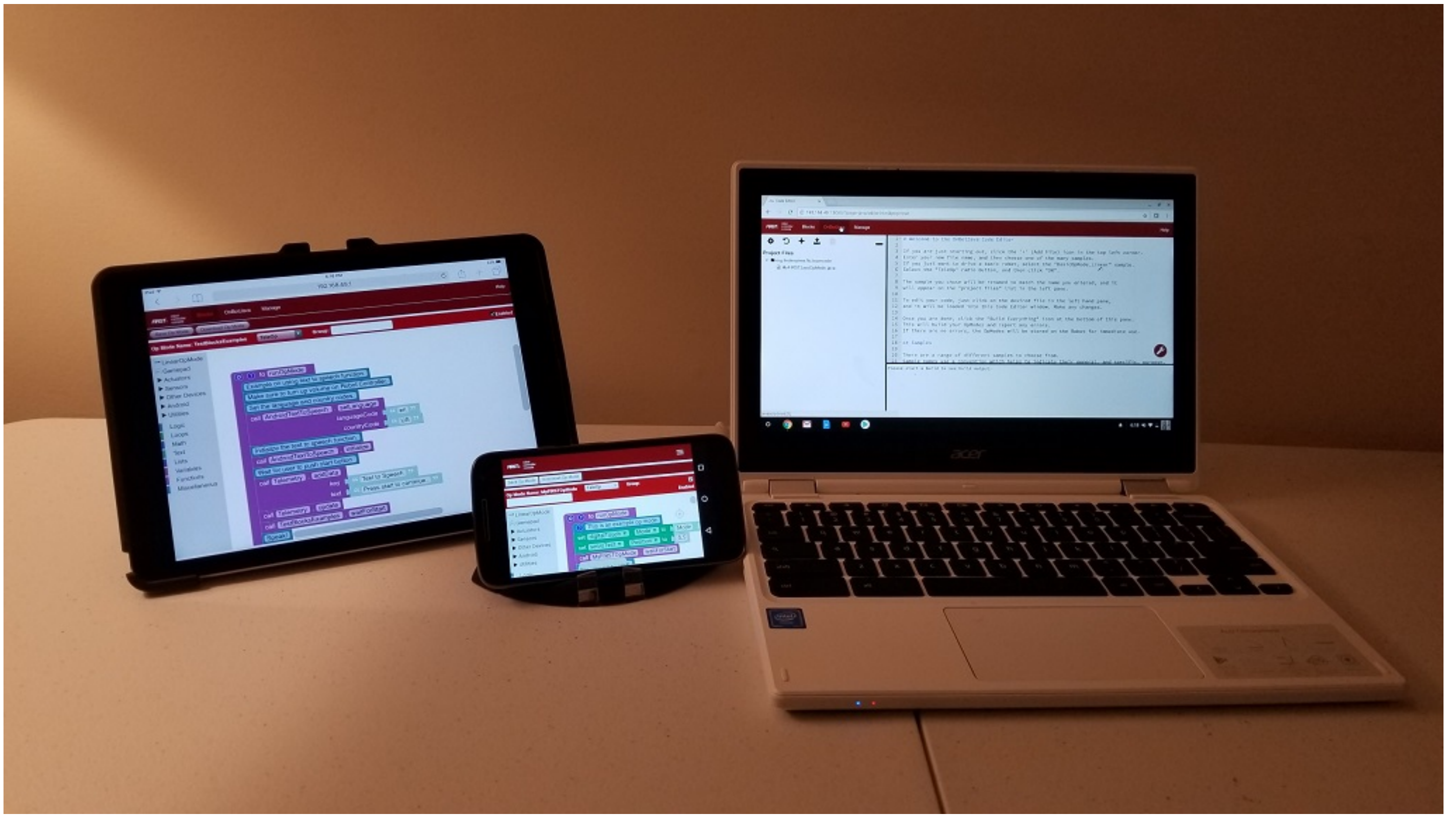
An advanced integrated development environment for creating Android apps. This tool is the same tool that professional Android app developers use. Android Studio is only recommended for advanced users who have extensive Java programming experience.



Recommendations

Each tool has its own merits and weaknesses. For many users (especially rookies and novice programmers), **the Blocks Programming Tool is the best overall tool to use**. The Blocks Programming Tool is intuitive and easy-to-learn. **It is the fastest way to get started programming your robot.**

The OnBot Java Programming Tool is similar to the Blocks Programming Tool. However, OnBot Java is a text-based tool and it requires that the user have a sound understanding of the Java programming language.



It is important to note that with the Blocks Programming Tool and the OnBot Java Programming Tool, a user only needs a web browser to create, edit and build op modes for their robot. A user can even create, edit and build op modes using an iPad, an Android phone, or a Chromebook.

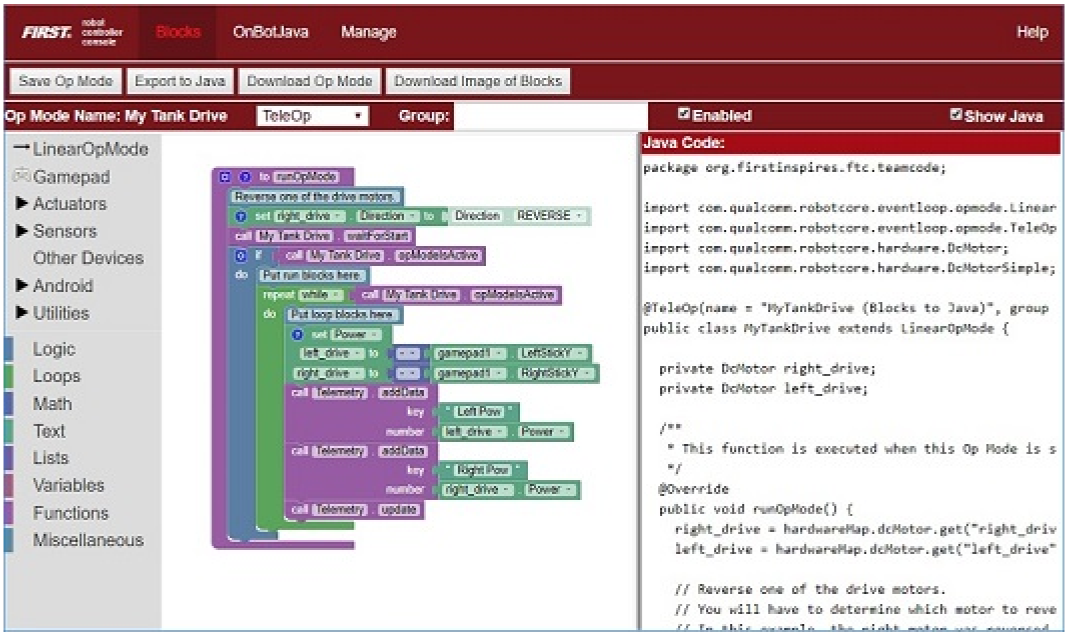
Android Studio is a powerful development tool. However, it requires extensive Java programming knowledge. It also needs a dedicated laptop to run the Android Studio software. Android Studio offers enhanced editing and debugging features that are not available on the OnBot Java Programming Tool. However, it is a more complicated tool and you must spend time learning how to use it. It is only recommended for advanced users.

21.1.2 Blocks Programming Tutorial

This tutorial will take you step-by-step through the process of configuring, programming, and operating your Control System. This tutorial uses the *Blocks Programming Tool* to help you get started quickly.

The Blocks Programming Tool is a visual design tool that lets programmers use a web browser to create, edit and save their *op modes*.

FIRST recommends getting starting with Blocks, even if you are an experienced programmer. Using Blocks is the *easiest* and *fastest* way to get acquainted with the Control System!



Note: Blocks indicates that the content is specific to Blocks Programming

Introduction Blocks

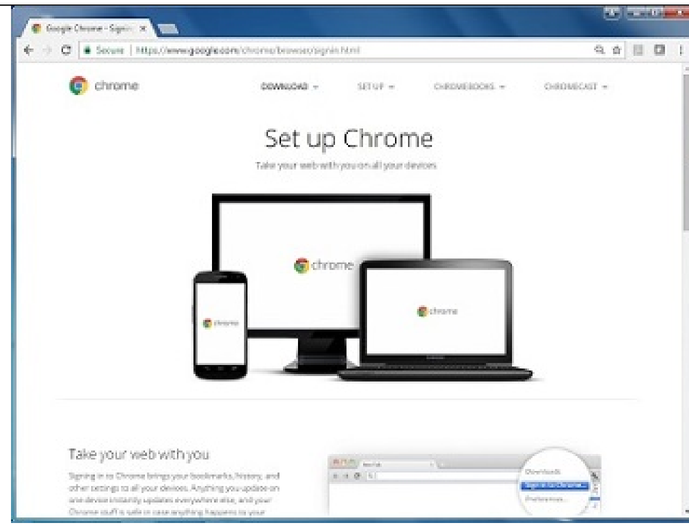
Required Materials

This wiki contains tutorials that demonstrate how to configure, program, and operate the *FIRST* Tech Challenge control system. In order to complete the tutorials, you will need to have the following materials available:

Required Item(s)	Image
A REV Robotics Control Hub to be used as a ROBOT CONTROLLER. Alternately, the Competition Manual does allow teams to instead use a REV Robotics Expansion Hub and a select list of Android phones. See rules R701 and R704 for the full details.	 
A REV Robotics Driver Hub to be used as a DRIVER STATION device. Alternately, the Competition Manual does allow teams to instead use a select list of Android phones. See rule R901 for the full details.	
Wireless Internet access.	

continues on next page

Table 1 – continued from previous page

Required Item(s)	Image
Laptop with Microsoft Windows 7, 8, 10 or 11 and Wi-Fi capability. Note that your laptop should have the most current service packs and system updates from Microsoft. If you are using a different type of machine (such as a Chromebook, Android Tablet, etc.) as your programming device, the steps might differ slightly on how to access the Programming Server on the Robot Controller. Refer to your device's user documentation for details on how to connect to a Wi-Fi network.	
Javascript-enabled web browser (Google Chrome is the recommended browser).	
REV Robotics Switch, Cable, & Bracket (REV-31-1387).	

continues on next page

Table 1 – continued from previous page

Required Item(s)	Image
FIRST-approved* 12V Battery (such as Tetrix W39057 or REV Robotics REV-31-1302).***For a list of FIRST-approved 12V batteries, refer to the current Competition Manual.**	 Or 
FIRST-approved* 12V DC Motor (such as Tetrix W39530, with power cable W41352).***For a list of FIRST-approved 12V motors,refer to the current Competition Manual.**	 
If you are using an approved 12V battery that has a Tamiya connector (like the Tetrix W39057 battery) you will need a REV Robotics Tamiya to XT30 Adapter Cable (REV-31-1382). If you have a REV Robotics Slim Battery (REV-31-1302) then you will not need this adapter since the REV battery already has an XT30 connector.	
REV Robotics Anderson to JST VH Cable (REV-31-1381).	
180-Degree Standard Scale Servo (such as Hitec HS-485HB).	

continues on next page

Table 1 – continued from previous page

Required Item(s)	Image
REV Robotics Color Sensor with 4-Pin Cable (REV-31-1154).	
REV Robotics Touch Sensor with 4-Pin Cable(REV-31-1425).	
Logitech F310 USB Gamepad.	
If you are using a smartphone as your Robot Controller, you will need a USB Type A male to type mini-B male cable. Control Hub users donot need this cable.	
If you are using a smartphone as your Robot Controller, you will need two (2) micro USB OTG adapters. If you are using a Control Hub as your Robot Controller, you will need one(1) micro USB OTG adapter.	

Using Your Android Device

Before you get started with your control system, it is helpful if you familiarize yourself with the basic operation of your Android device.

REV Robotics Driver Hub

Teams who are using the REV Robotics Driver Hub as their DRIVER STATION should refer to the official documentation from REV Robotics for instructions on how to set up and use the REV Robotics Driver Hub.

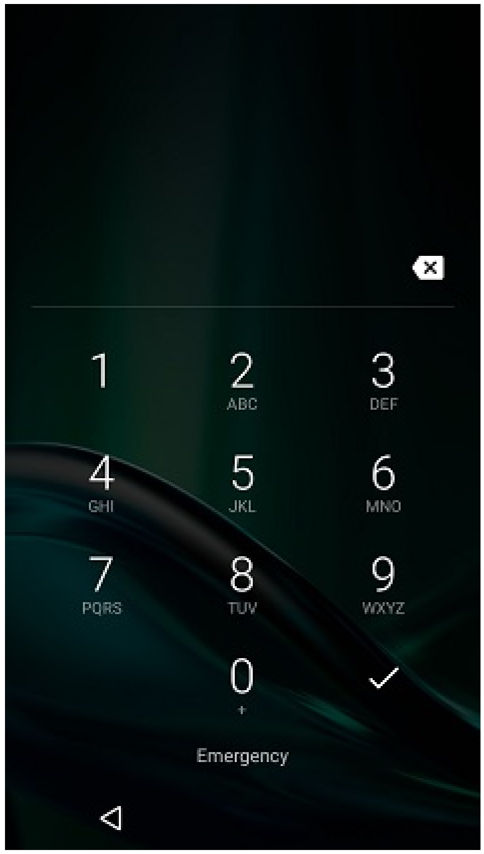
Android Smartphone

While not recommended, the Competition Manual does allow select Android smartphones to be used as a DRIVER STATION or ROBOT CONTROLLER. See rule R704 for the full details.

Unlocking Your Screen

When you first power on an Android phone, it usually starts off with the screen in a "locked" state. For the Motorola smartphones that are used in the *FIRST* Tech Challenge, you must touch the locked screen and then slide your finger upwards along the screen to unlock the phone. Note that different devices might require a slightly different procedure to unlock the screen.

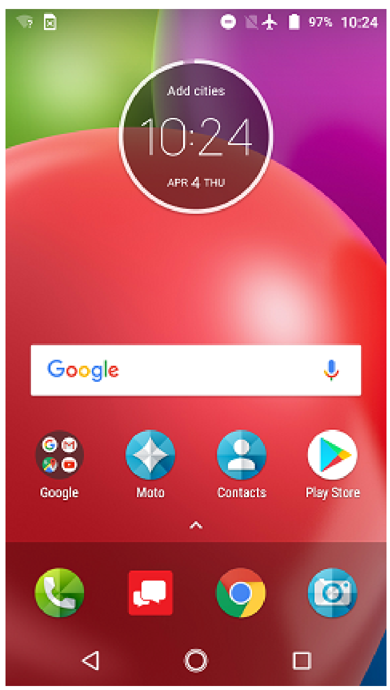
Depending on your security settings, you might be challenged for a pass code or PIN number. Use the touch screen to enter in the pass code or PIN value and tap on the check mark to log into the device.



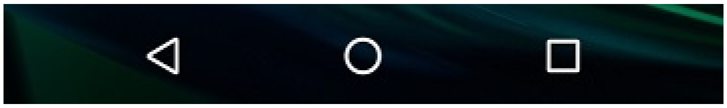
Navigating in Android

Your phone should display its home screen if you just powered it on and unlocked it. Note that the actual screens on your smartphone might differ slightly from the screens depicted in this tutorial.


11:09
Google
11
10
09
Google
Photos
Contacts
Play Store


97% 10:24
Add cities
10:24
APR 4 THU
Google
Google
Moto
Contacts
Play Store

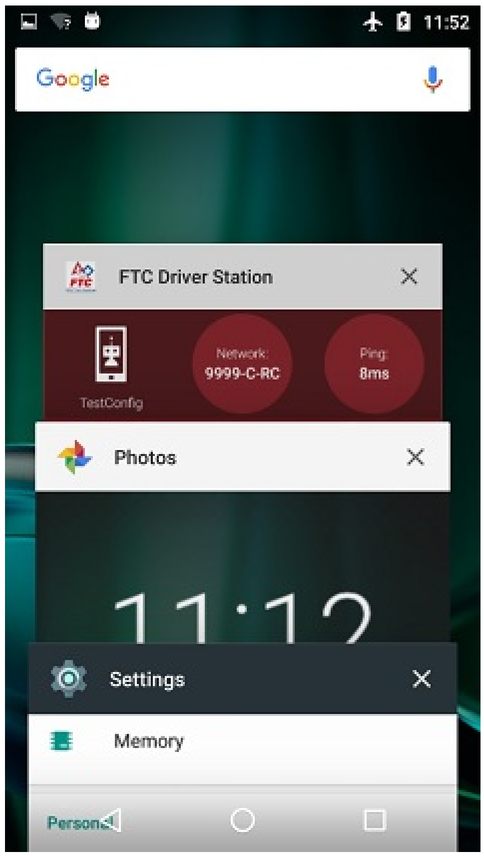
At the bottom of the screen there should be some buttons that you can use to navigate the screens on your Android device.



The leftmost button (see image above) is the "Back" button. You can use this button to return to the previous screen on your Android device.

The center button is the "Home" button. Pressing this button should take you back to the home or opening screen of your Android device.

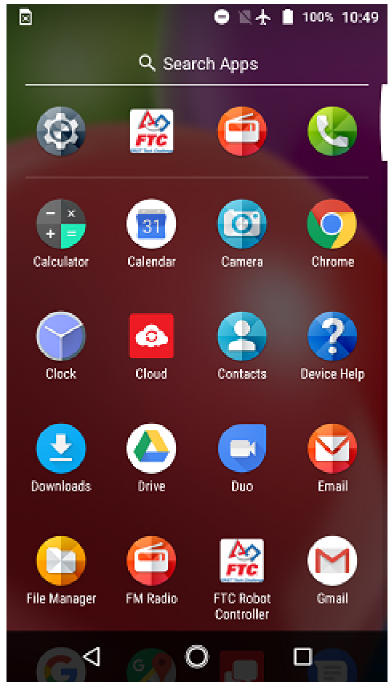
The rightmost button is the "Recent Apps" button. If you click on this button it will display the apps that were recently run and are dormant in the background. You can close a recent app by tapping the "X" button on the app's listing.



Note that some Android smartphones have an auto-hide feature which automatically hides the bottom navigation buttons. If your smartphone has this feature, you might need to swipe up from the bottom of the screen to display the navigation buttons.

Displaying Available Apps on your Android Phone

If you are using a device with Android Nougat (7.x) or newer, you can display the available apps by simply swiping upwards from the bottom of the touchscreen. Newer versions of Android no longer have the *App Drawer* feature.



Configuring Your Hardware Blocks

Configuring your Android Devices

What Needs to Be Configured for My Control System?

Driver Hub Configuration

Teams who are using the REV Robotics Driver Hub as their DRIVER STATION should refer to the official documentation from REV Robotics for instructions on how to set up and use the REV Robotics Driver Hub.

Control Hub Configuration

Note: References to the DRIVER STATION smartphone may instead apply to the REV Robotics Driver Hub, which is preloaded with the Driver Station (DS) app.

Teams who are using a Control Hub (which has an integrated Android Device) will only need to configure a single smartphone for use as a DRIVER STATION. The process is as follows:

- Rename the smartphone to “<TEAM NUMBER>-DS” (where <TEAM NUMBER> is replaced by your team number).

- Install the Driver Station (DS) app onto the DRIVER STATION device. (The DS app is pre-installed on the REV Robotics Driver Hub.)
- Put your phone into Airplane Mode (with the WiFi radio still on).
- Pair (i.e., wirelessly connect) the DRIVER STATION to the Control Hub.

Important: Eventually the Control Hub will need to be renamed so that its name complies with the Competition Manual, but for now we will use the Control Hub with its default name. You can learn how to manage a Control Hub (and modify its name, password, etc.) in *this tutorial*.

Two Android Smartphone Configuration

Teams who have two smartphones and are not using a Control Hub will need to configure one smartphone for use as a Robot Controller and a second smartphone for use as an DRIVER STATION. The process is as follows,

- Rename one smartphone to “<TEAM NUMBER>-RC” (replace <TEAM NUMBER> with your team number).
- Install the Robot Controller app onto the Robot Controller phone.
- Rename a second smartphone to “<TEAM NUMBER>-DS” (where <TEAM NUMBER> is replaced by your team number).
- Install the Driver Station app onto the DRIVER STATION device. (The DS app is pre-installed on the REV Robotics Driver Hub.)
- Put your phones into Airplane Mode (with the WiFi radios still on).
- Pair (i.e., wirelessly connect) the DRIVER STATION to the Robot Controller.

Renaming Your Smartphones

The official rules of the *FIRST* Tech Challenge (see R707) require that you change the Wi-Fi name of your smartphones to include your team number and "-RC" if the phone is a Robot Controller or "-DS" if it is a DRIVER STATION. A team can insert an additional dash and a letter ("A", "B", "C", etc.) if the team has more than one set of Android phones.

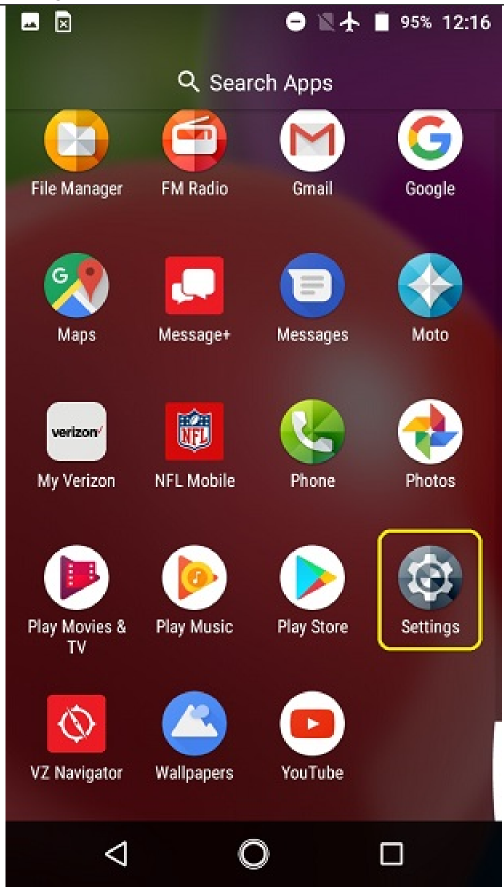
If, for example, a team has a team number of 9999 and the team has multiple sets of phones, the team might decide to name one phone "9999-C-RC" for the Robot Controller and the other phone "9999-C-DS" for the DRIVER STATION. The "-C" indicates that these devices belong to the third set of phones for this team.

The name of a Robot Controller phone can be changed in the RC app, using instructions *found here*. It can also be changed at the *Manage* page from the RC app, a paired DS app, or a connected laptop; click `Apply Wi-Fi Settings` when done.

The name of a DRIVER STATION device can be changed in the DS app, using instructions *found here*.

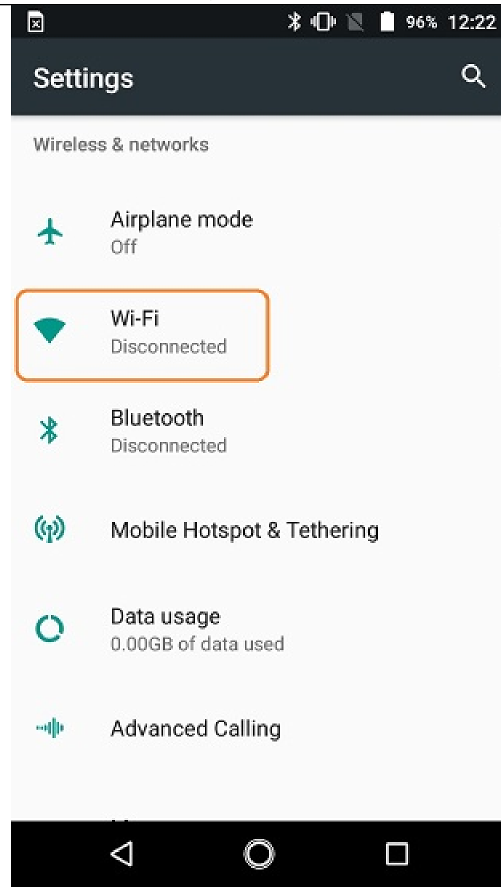
As an alternate, the device names can be changed at the Android system level, as described below.

Note: It will take an estimated 5 minutes per phone to complete this task.

Step	Image
1. Browse the list of available apps on the smartphone and locate the **Settings** icon. Click on **Settings** icon to display the Settings screen.	

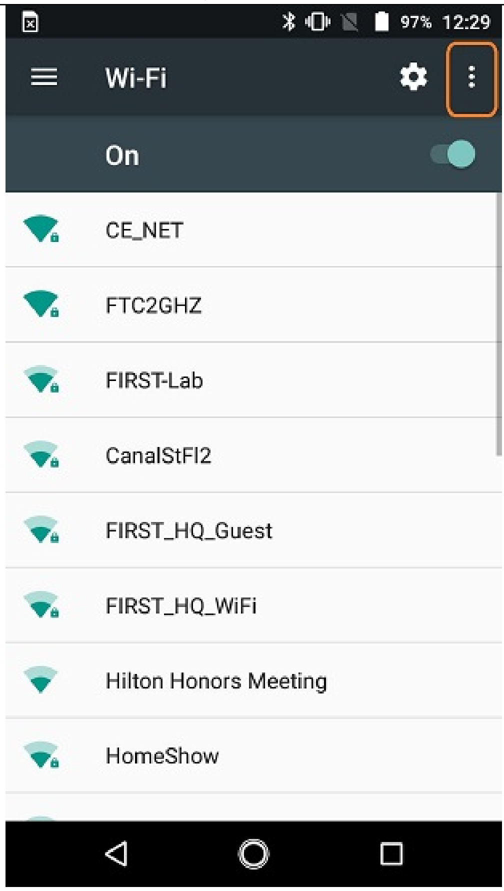
continues on next page

Table 2 – continued from previous page

2. Click on **Wi-Fi** to launch the Wi-Fi screen.	

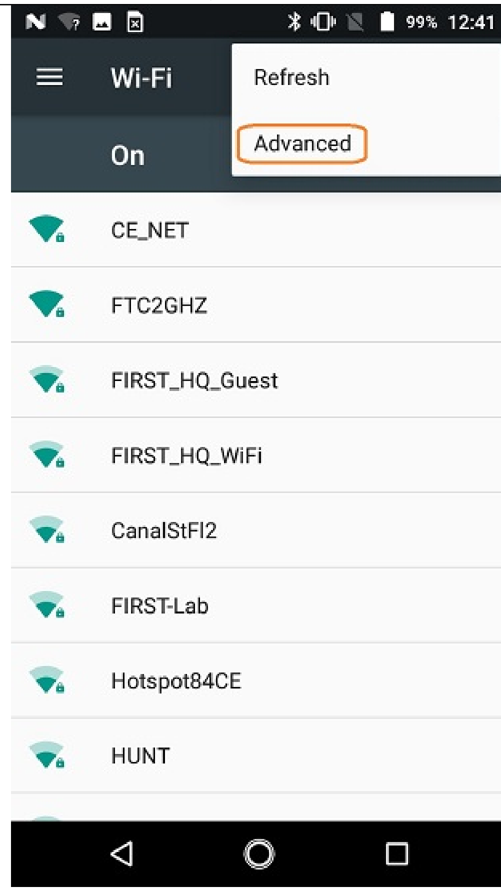
continues on next page

Table 2 – continued from previous page

3. Touch the three vertical dots to display a pop-up menu.	

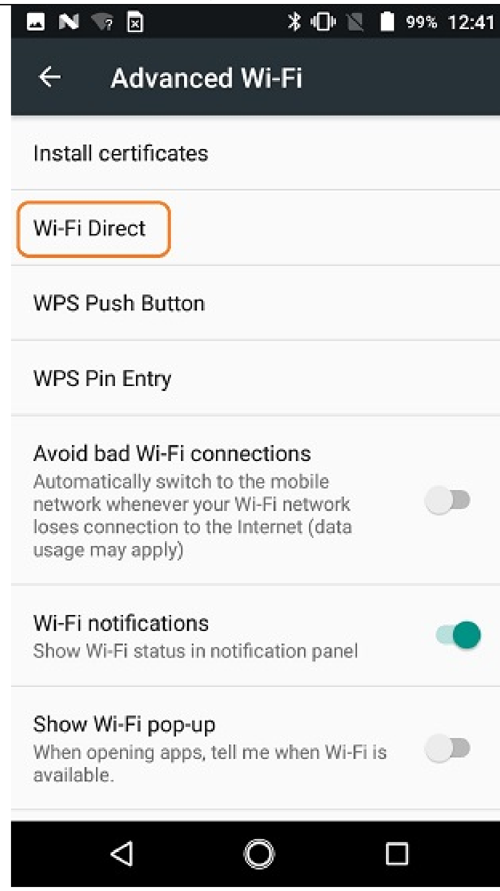
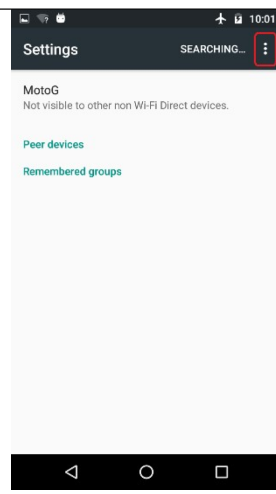
continues on next page

Table 2 – continued from previous page

4. Select **Advanced** from the pop-up menu.	

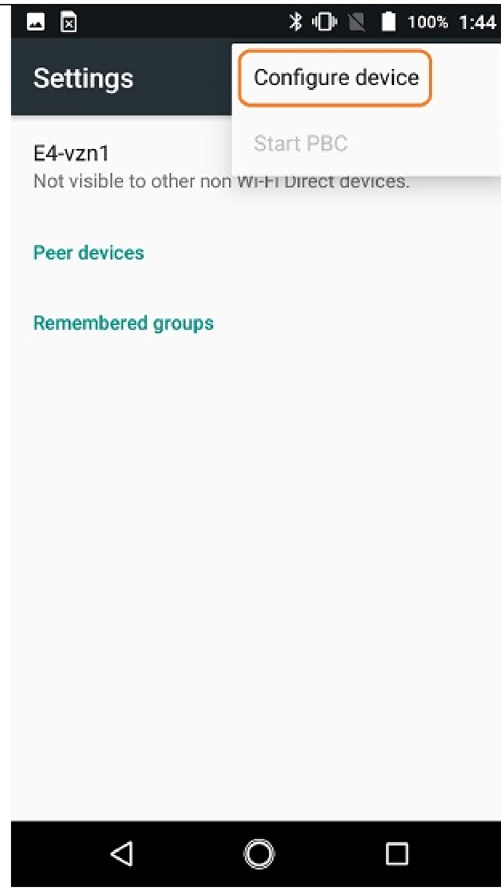
continues on next page

Table 2 – continued from previous page

5. Select **Wi-Fi Direct** from the **Advanced Wi-Fi** screen.	
6. Touch the three vertical dots to display a pop-up menu.	

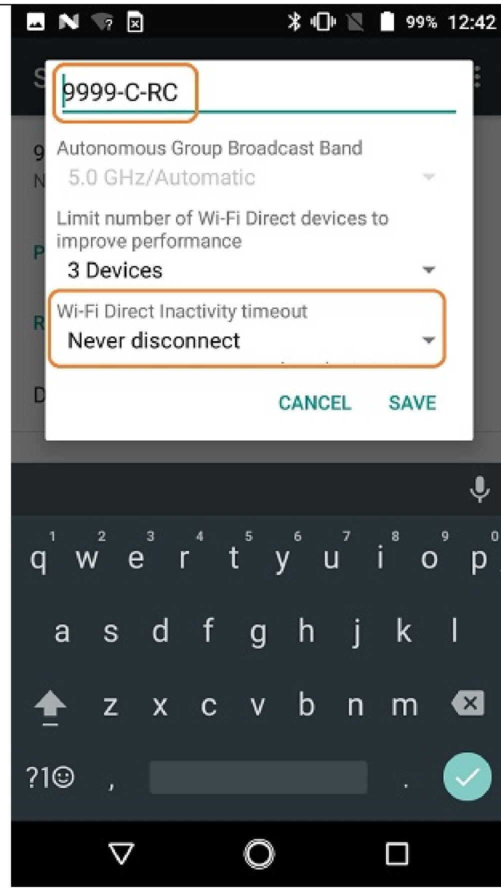
continues on next page

Table 2 – continued from previous page

7. Select **Configure Device** from the pop-up menu.	

continues on next page

Table 2 – continued from previous page

<table>
<tr>
<td>8. Use touch pad to enter new name of device. If the device will be a Robot Controller, specify your team number and -RC. If the device will be a DRIVER STATION, specify your team number and -DS. You can also set the Wi-Fi Direct inactivity timeout to Never disconnect and then hit the SAVE button to save your changes. Note that in the screenshot shown to the right, the team number is 9999. The “-C” indicates that this is from the third pair of smartphones for this team. The -RC indicates that this phone will be a Robot Controller.</td>
<td>
</td>
</tr>
<tr>
<td>9. After renaming your phone, power cycle the device.</td>
<td></td>
</tr>
</table>

Installing the *FIRST* Tech Challenge Apps

For detailed instructions on how to install and update apps, please see these other pages:

ROBOT CONTROLLER app

DRIVER STATION app

As of 2021, the SDK apps (v 6.1 and higher) are no longer available on Google Play.

The REV Hardware Client software will allow you to download the apps to devices: REV Robotics Control Hub, REV Robotics Expansion Hub, REV Robotics Driver Hub, and other approved Android devices (*see section below, called Updating Apps on Android Phones*). Here are some of the benefits:

- Connect to a REV Robotics Control Hub via WiFi.
- One Click update of all software on connected devices.
- Pre-download software updates without a connected device.
- Back up and restore user data from Control Hub.
- Install and switch between DS and RC applications on Android Devices.
- Access the Robot Control Console on the Control Hub.

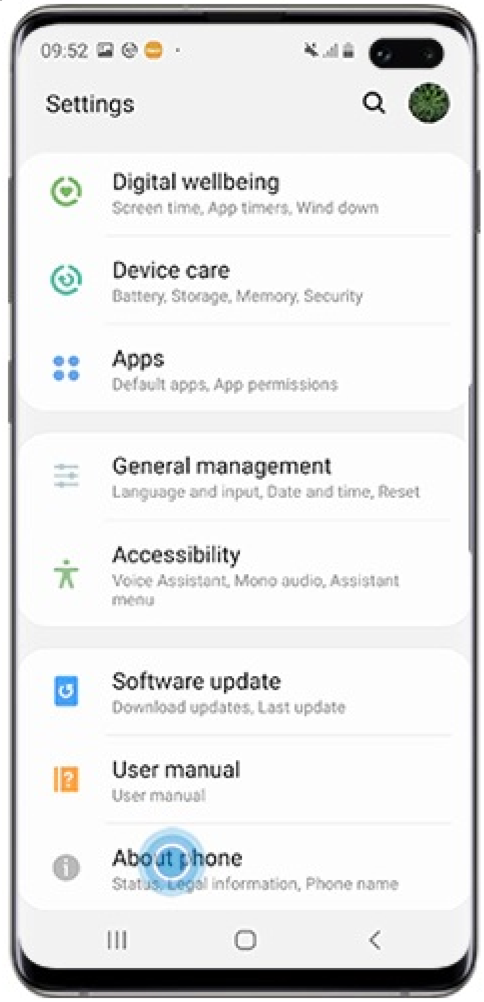
The app releases are also available on the FtcRobotController GitHub repository. It is possible to "side-load" the apps onto the Robot Controller (RC) and Driver Station (DS) phones. However, this section of the document does **not** include such instructions; other document pages describe side-loading the *RC app* and the *DS app*.

Updating Apps and Firmware on REV Robotics Devices (REV Robotics Expansion Hub, REV Robotics Control Hub, REV Robotics Driver Hub)

The REV Hardware Client software is used to install and update apps, firmware and/or operating systems on devices from REV Robotics. Simply connect the device via USB to your PC with the REV Hardware Client installed and running, and the software will detect connected hardware. After detection, the REV Hardware Client can then update the Robot Controller (RC) app on a REV Robotics Control Hub, update the Driver Station (DS) app on a REV Robotics Driver Hub, or update firmware.

Updating Apps on Android Phones

The REV Hardware Client software is used to install, uninstall, and update apps on Android phones. However, the phones must have **Developer Options** enabled in order for the phone to be properly recognized and updated by the REV Hardware Client software. The process for enabling Developer Options is as follows:

Step	Image
1. Go to “Settings”, then tap “About device” or “About phone”.	

continues on next page

Table 3 – continued from previous page

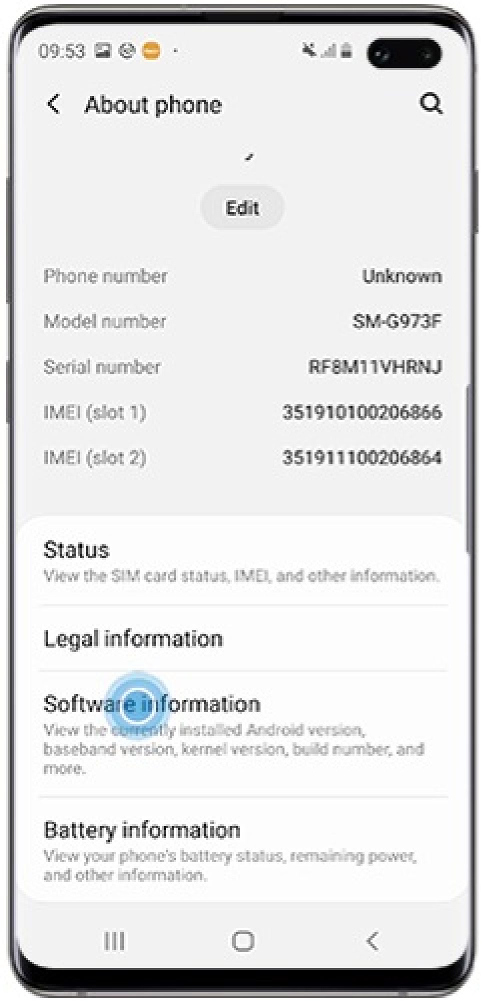
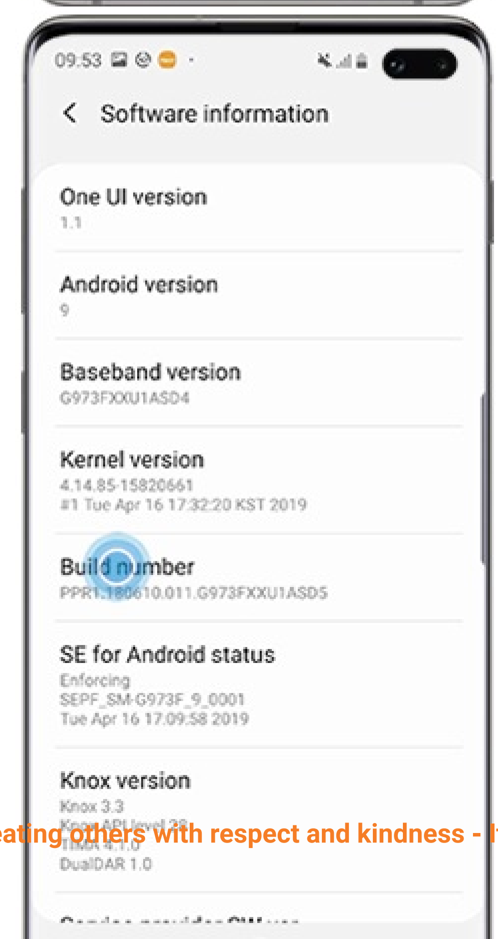
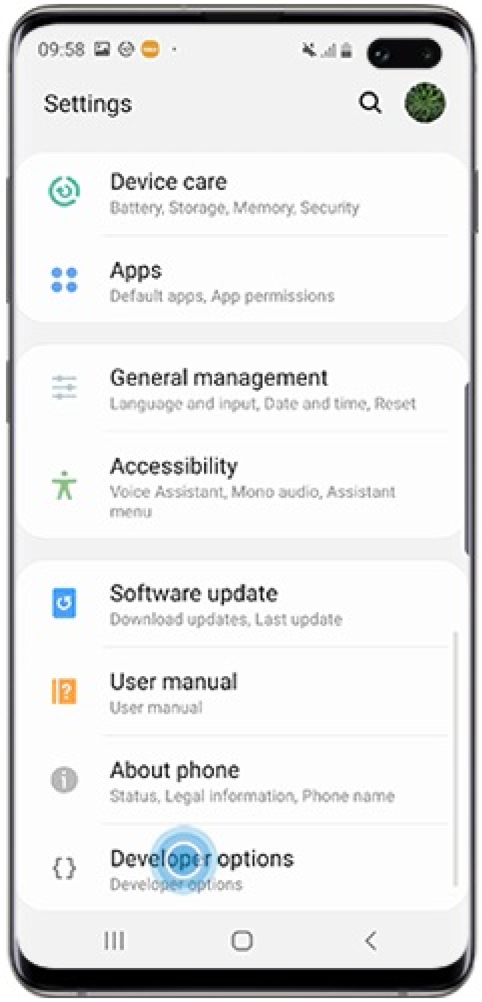
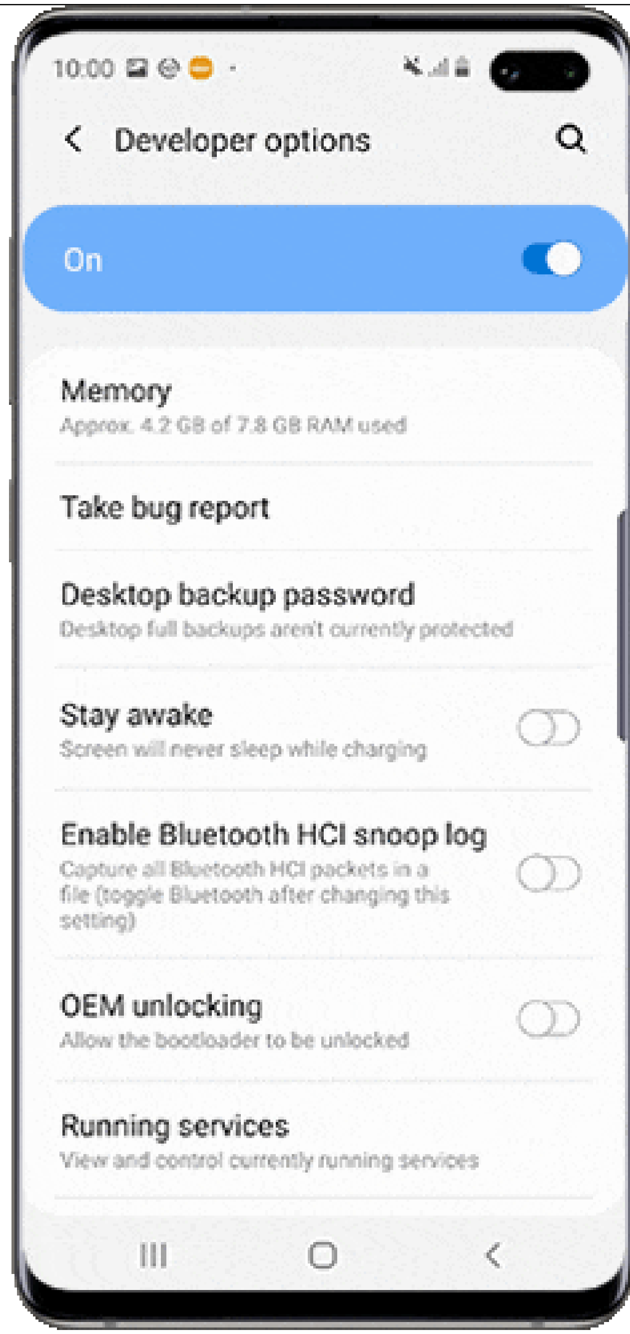
<table>
<tr><th>Step</th><th>Image</th></tr>
<tr><td></td><td>
</td></tr>
<tr><td>2. Scroll down, then tap Build number seven times. Depending on your device and operating system, you may need to tap Software information, then tap Build number seven times.</td><td>
</td></tr>
</table>

Table 3 – continued from previous page

Step	Image
3. Enter your pattern, PIN or password to enable the Developer options menu.	
4. The "Developer options" menu will now appear in your Settings menu. Depending on your device, it may appear under Settings > General > Developer options.	

continues on next page

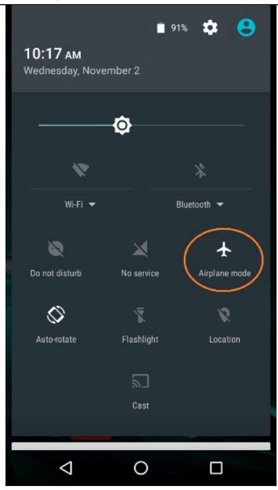
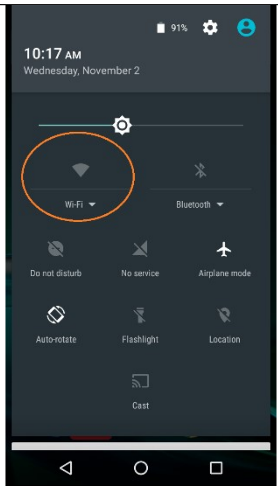
Table 3 – continued from previous page

Step	Image
5. To disable the Developer options at anytime, tap the switch.	

Placing Phones into Airplane Mode with Wi-Fi On

For the *FIRST* Tech Challenge competitions, it is important that you place your Robot Controller and DRIVER STATION devices into Airplane mode but keep their Wi-Fi radios turned on. This is important because you do not want any of the cellular telephone functions to be enabled during a match. The cellular telephone functions could disrupt the function of the robot during a match.

Note: It will take an estimated 2.5 minutes per phone to complete this task. Also note that the screens displayed on your Android devices might differ slightly from the images contained in this document.

Step	Image
1. On the main Android screen of each smartphone, use your finger to slide from the top of the screen down towards the bottom ofthe screen to display the quick configuration screen. Note that for some smartphones you might have to swipe down more than once to display the quick configuration screen, particularly if there are messages or notifications displayed at the top of your screen. Look for the Airplane mode icon (which is shaped like an airplane) and if the icon is not activated, touch the icon to put the phone into airplane mode.	
2. Placing the phone into airplane mode will turn off the Wi-Fi radio. If the Wi-Fi icon has a diagonal line through it (see Step 1 above), then the Wi-Fi radio is disabled. You will need to touch the **Wi-Fi** icon on the quick configuration screen to turn the Wi-Fi radio back on.	

Pairing the DRIVER STATION to the Robot Controller

Control Hub Pairing

The REV Robotics Control Hub should come with the Robot Controller app pre-installed. Once you have successfully installed the Driver Station on an Android phone, you will want to establish a secure wireless connection between the Control Hub and the DRIVER STATION. This connection will allow your DRIVER STATION device to select op modes on your Robot Controller and send gamepad input to these programs. Likewise, it will allow your op modes running on your Robot Controller to send telemetry data to your DRIVER STATION phone where it can be displayed for your drivers. The process to connect the two devices is known as "pairing."

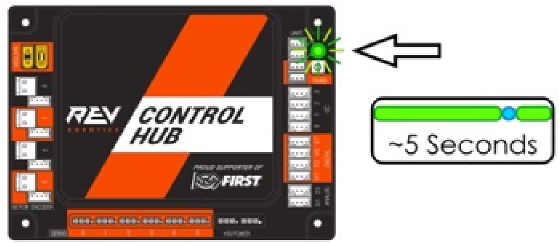
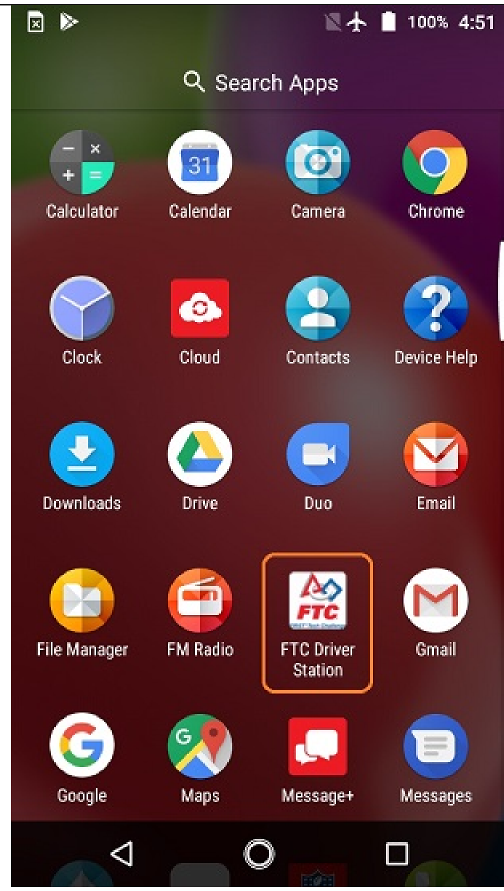
Note: the Control Hub does not have its own internal battery. Before you can connect a Driver Station to the Control Hub, you must connect the Control Hub to a 12V battery.

Also note that it will take an estimated 10 minutes to complete this task.

Step	Image
1. Connect an approved 12V battery to the power switch (REV-31-1387) and make sure the switch is in the off position. Connect the switch to an XT30 port on the Control Hub and turn the switch on. The LED should initially be blue on the Control Hub.	

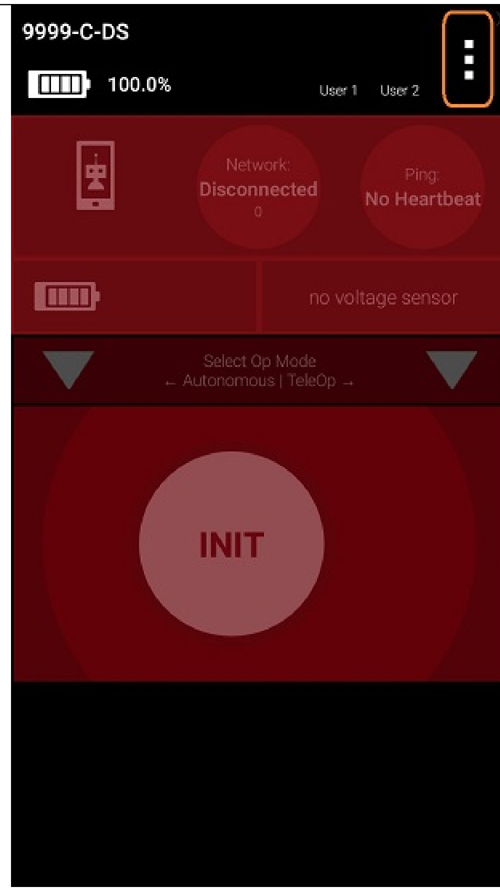
continues on next page

Table 4 – continued from previous page

Step	Image
2. It takes approximately 18 seconds for the Control Hub to power on. The Control Hub is ready to pair with the Driver Station when the LED turns green. Note: the light blinks blue every ~5 seconds to indicate that the Control Hub is healthy.	
3. On the Driver Station device, browse the available apps and locate the **FTC Driver Station** icon. Tap on the icon to launch the Driver Station app. Note that the first time you launch the app your Android device might prompt you for permissions that the app will need to run properly. Whenever prompted, press **Allow** to grant the requested permission.	

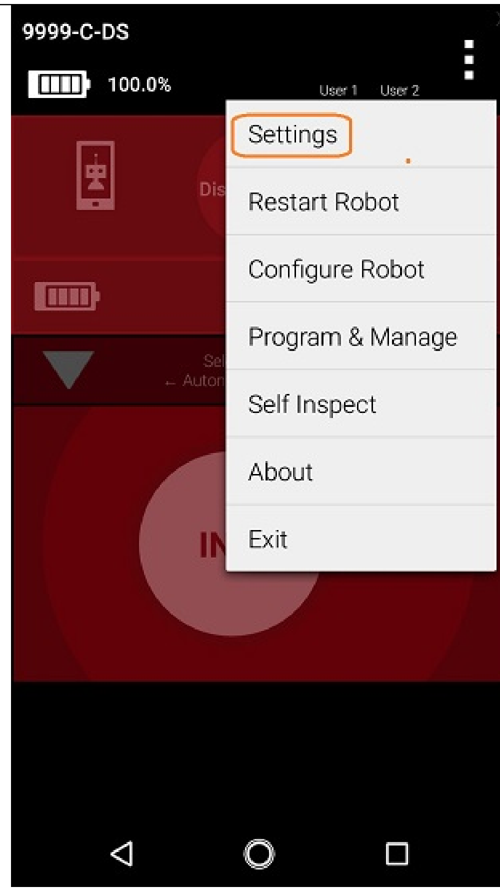
continues on next page

Table 4 – continued from previous page

Step	Image
4. Touch the three vertical dots on the upper right hand corner of the main screen of the Driver Station app. This will launch a pop-up menu.	

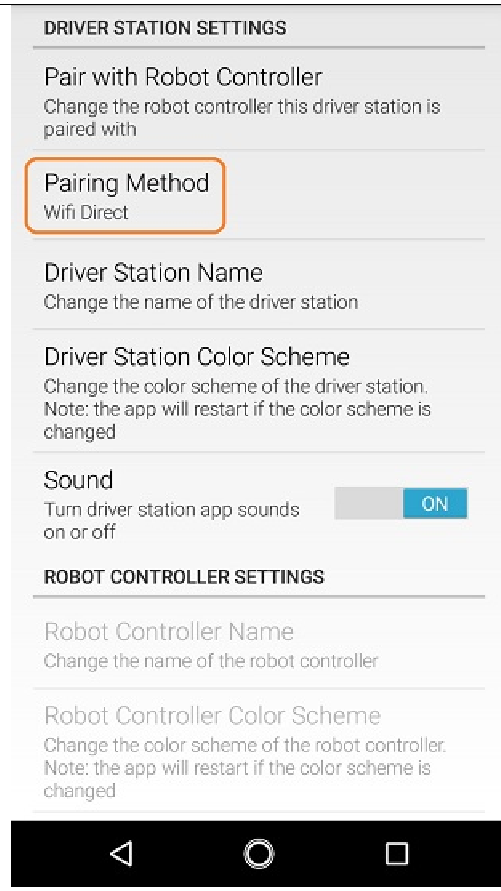
continues on next page

Table 4 – continued from previous page

Step	Image
5. Select **Settings** from the pop-up menu.	

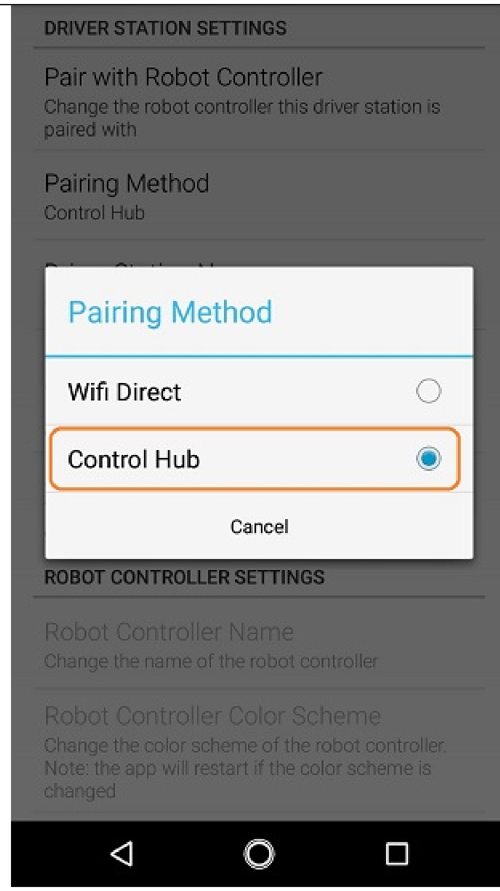
continues on next page

Table 4 – continued from previous page

Step	Image
6. From the **Settings** screen, look for and select **Pairing Method** to launch the **Pairing Method** screen.	

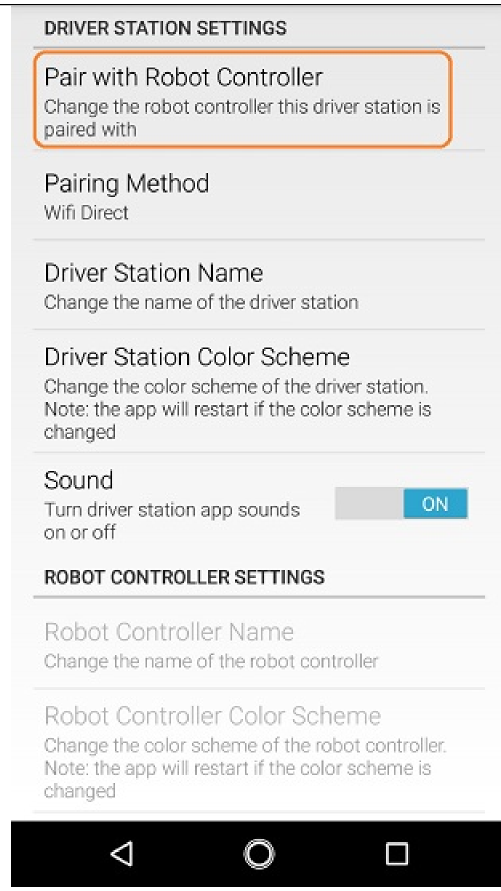
continues on next page

Table 4 – continued from previous page

Step	Image
7. Touch the words **Control Hub** to indicate that this DRIVER STATION will be pairing with a Control Hub.	

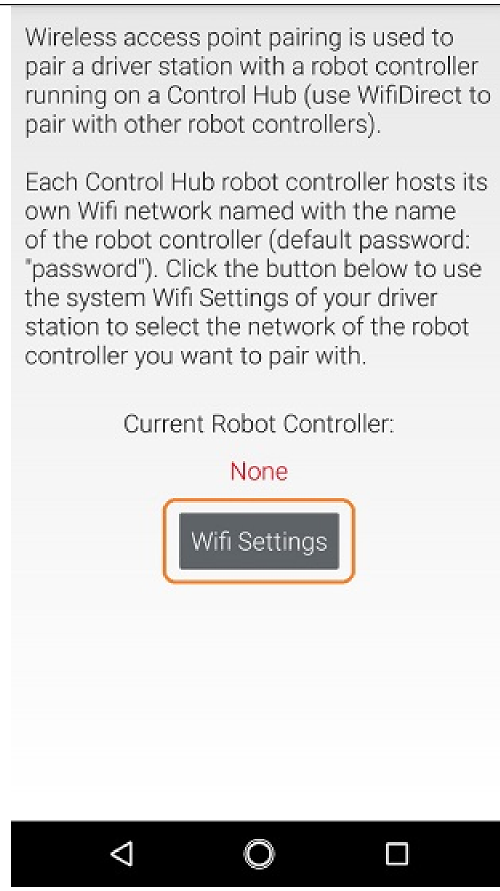
continues on next page

Table 4 – continued from previous page

Step	Image
8. From the **Settings** screen, look for and select **Pair with Robot Controller** to launch the **Pair with Robot Controller** screen.	

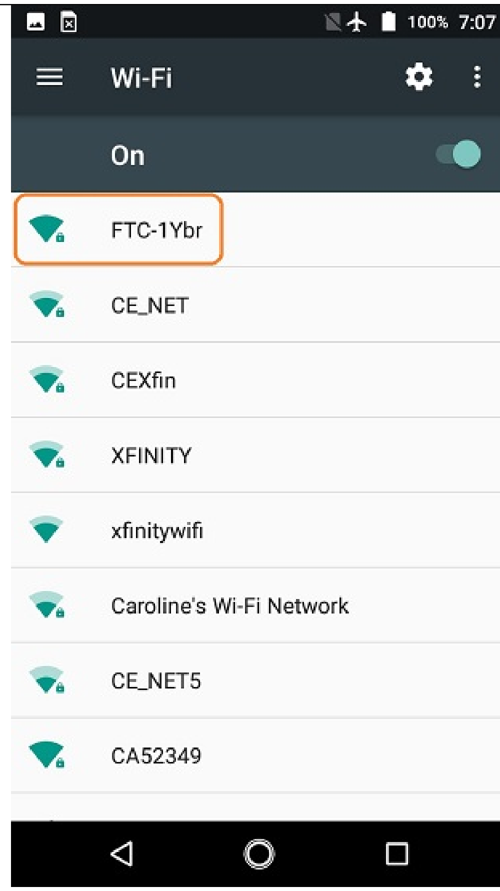
continues on next page

Table 4 – continued from previous page

Step	Image
9. From **Pair with Robot Controller** screen,look for and press the **Wifi Settings** button to launch the device's Android WifiSettings screen.	

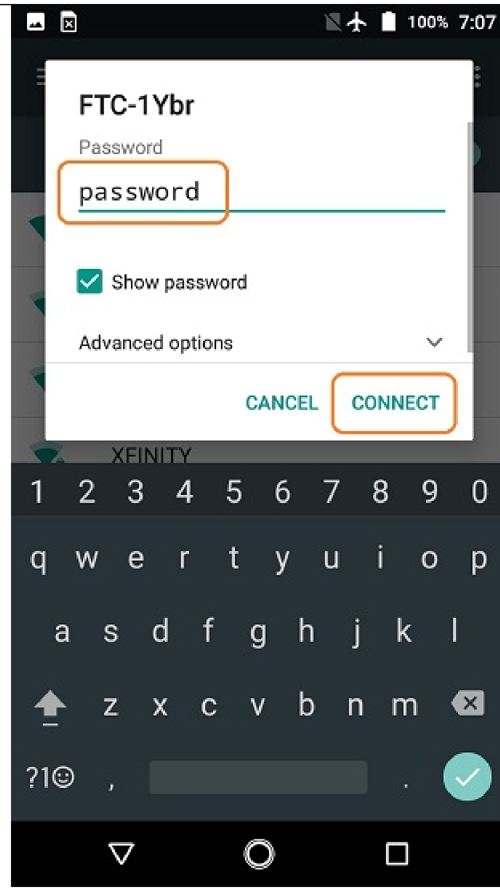
continues on next page

Table 4 – continued from previous page

Step	Image
10. Find the name of your Control Hub's wireless network from the list of available WiFi networks. Click on the network name to select the network. If this is the first time you are connecting to the Control Hub, then the default network name should begin with the prefix FTC- (FTC-1Ybr in this example). The default network name should be listed on a sticker attached to the bottom side of the Control Hub.	

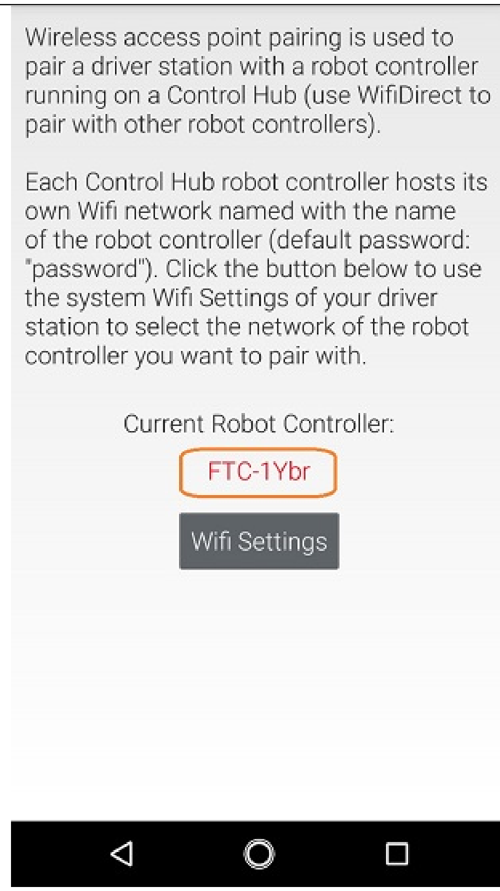
continues on next page

Table 4 – continued from previous page

Step	Image
11. When prompted, specify the password for the Control Hub's WiFi network and press **Connect** to connect to the Hub. Note that the default password for the Control Hub network is `password`. Also note that when you connect to the Control Hub's WiFi network successfully, the DRIVER STATION will not have access to the Internet.	

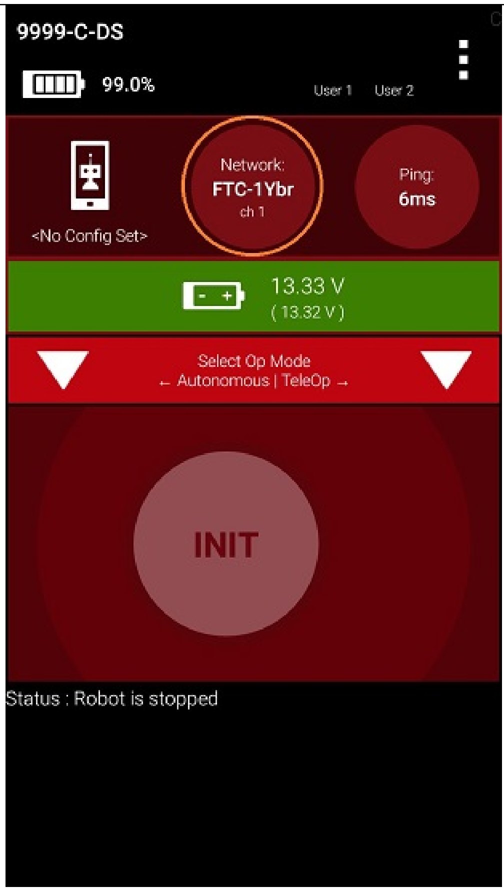
continues on next page

Table 4 – continued from previous page

Step	Image
12. After you successfully connected to the Hub, use the back arrow to navigate to the previous screen. You should see the name of the WiFi network listed under "Current Robot Controller:". Use the back-arrow key to return to the Settings screen. Then press the back-arrow key one more time to return to the main DRIVER STATION screen.	

continues on next page

Table 4 – continued from previous page

Step	Image
13. Verify that the DRIVER STATION screen has changed and that it now indicates that it is connected to the Control Hub. The name of the Control Hub's WiFi network (FTC-1Ybr in this example) should be displayed in the Network field on the Driver Station.	

Two Android Smartphone Pairing

Important: If your DRIVER STATION was previously paired to a Control Hub, and you currently would like to connect to an Android smartphone Robot Controller, then before attempting to pair to the Robot Controller, you should forget the Wi-Fi network for the previous Control Hub (using the Android Wifi Settings screen on the DRIVER STATION) and then power cycle the DRIVER STATION phone. If the previous Control Hub is powered on and if you haven't forgotten this network, then the DRIVER STATION might try and connect to the Control Hub and might be unable to connect to the Robot Controller smartphone.

Once you have successfully installed the apps onto your Android phones, you will want to establish a secure wireless connection between the two devices. This connection will allow your DRIVER STATION device to select op modes on your Robot Controller phone and send gamepad input to these programs. Likewise, it will allow your op modes running on your Robot Controller phone to send telemetry data to your DRIVER STATION device where it can be displayed for your drivers. The process to connect the two phones is known as pairing.

Note that it will take an estimated 10 minutes to complete this task.

continues on next page

Table 5 – continued from previous page

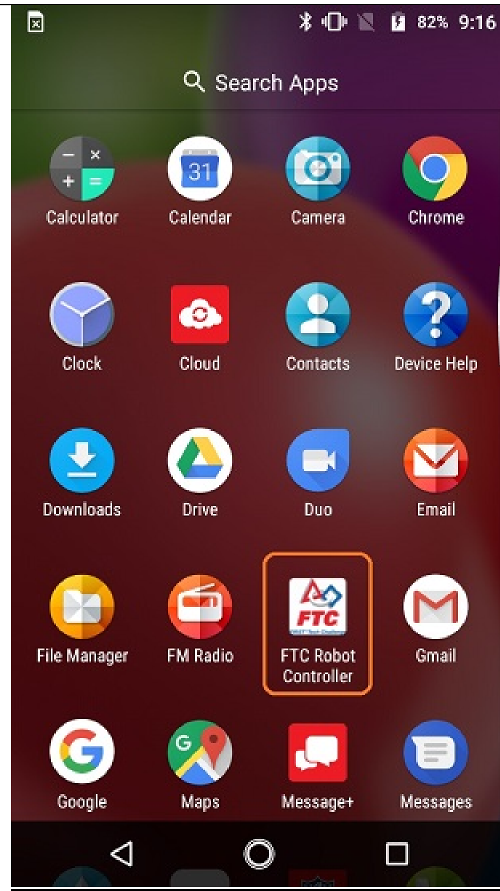
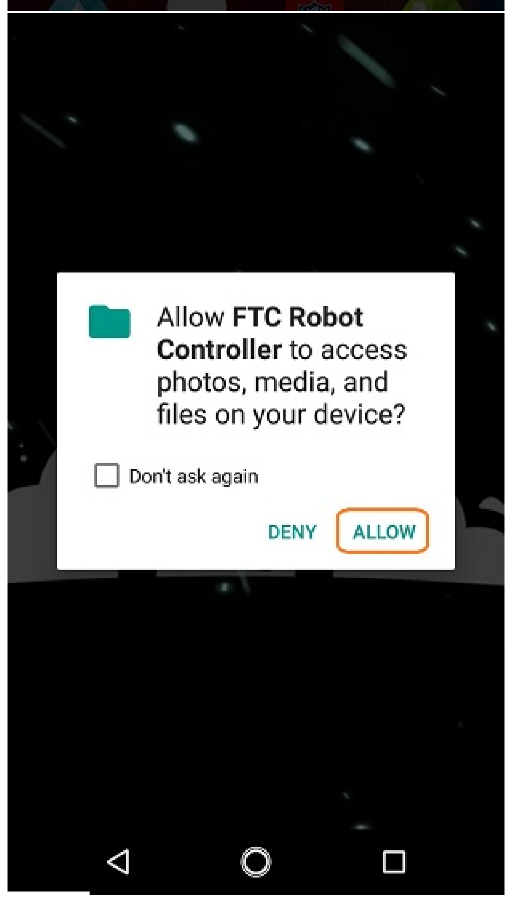
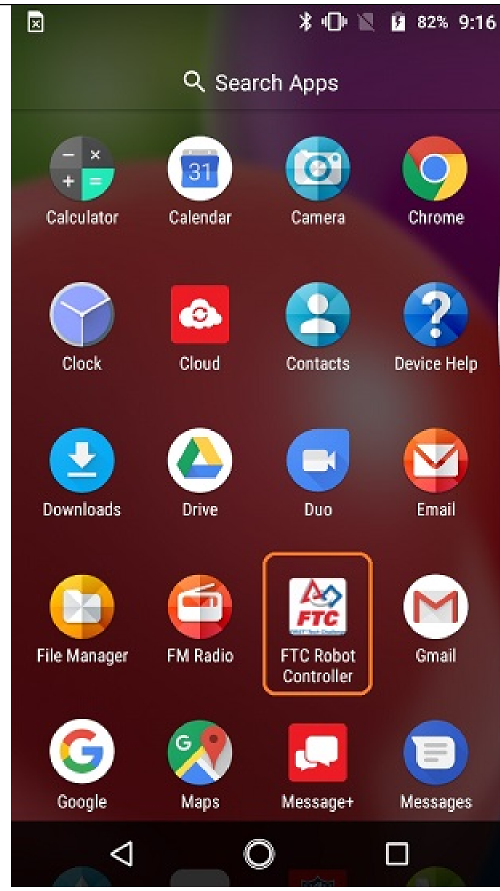
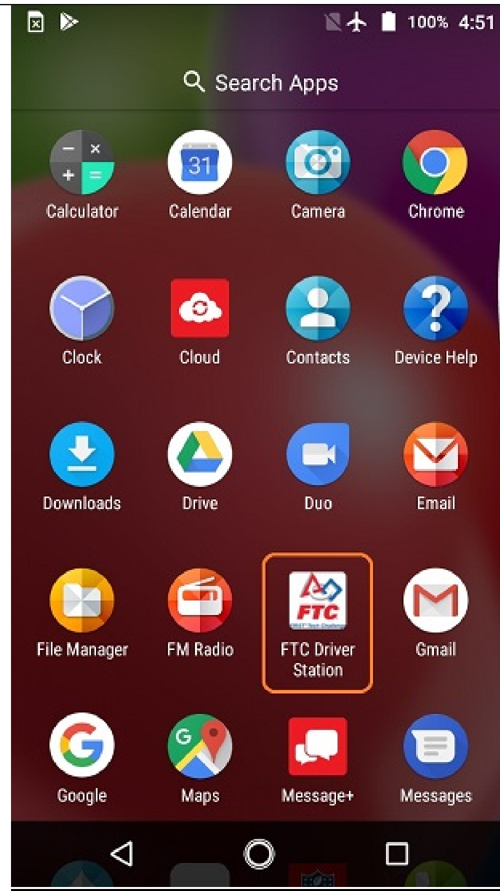
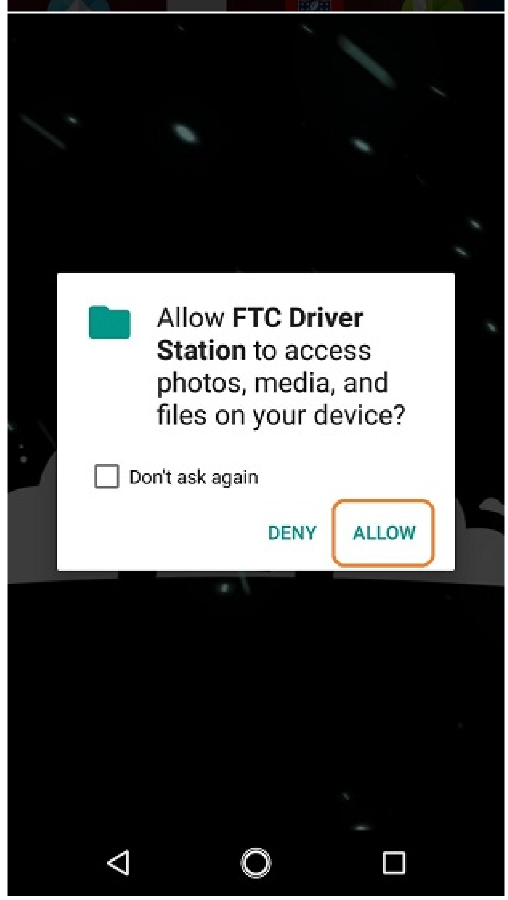
Step	Image
Step	Image
1. On the Robot Controller device, browse the available apps and locate the **FTC Robot Controller** icon. Tap on the icon to launch the Robot Controller app. Note that the first time you launch the app your Android device might prompt you for permissions that the app will need to run properly. Whenever prompted, press **Allow** to grant the requested permission.	

Table 5 – continued from previous page

Step	Image
2. Verify that the Robot Controller app is running. The **Robot Status** field should read running if it is working properly.	

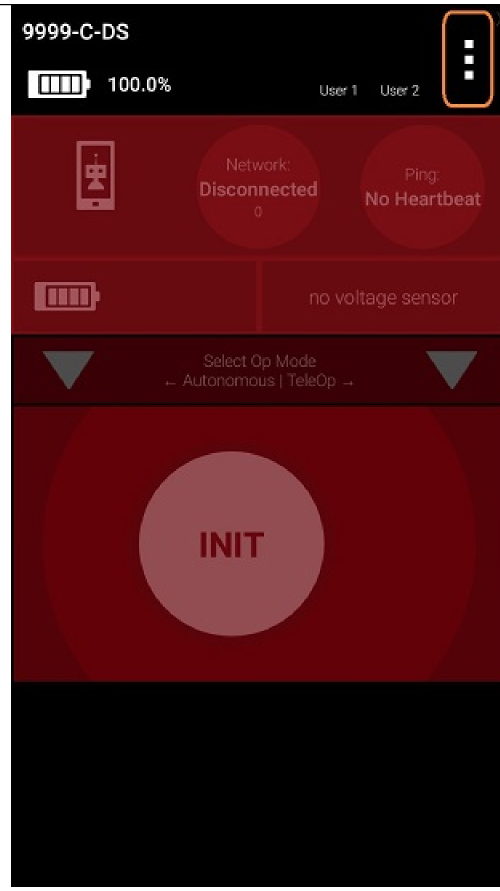
continues on next page

Table 5 – continued from previous page

Step	Image
3. On the DRIVER STATION device, browse the available apps and locate the **FTC Driver Station** icon. Tap on the icon to launch the Driver Station app. Note that the first time you launch the app your Android device might prompt you for permissions that the app will need to run properly. Whenever prompted, press **Allow** to grant the requested permission.	 

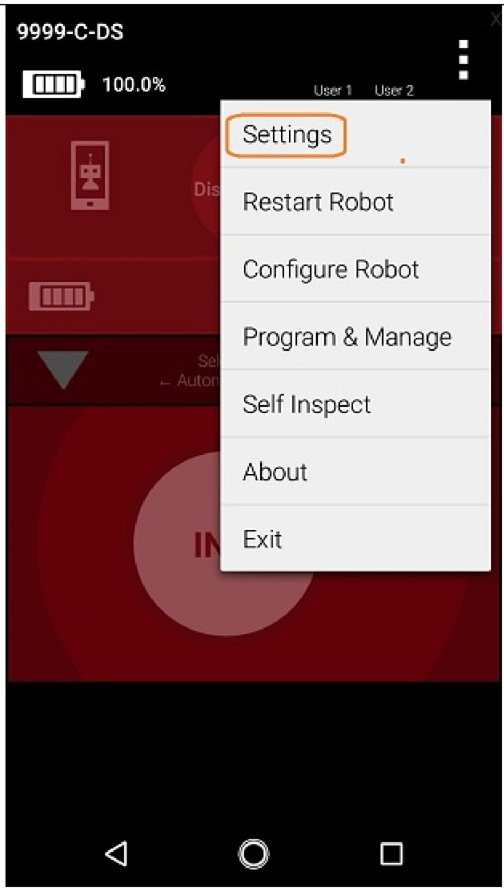
continues on next page

Table 5 – continued from previous page

Step	Image
4. Touch the three vertical dots on the upper right hand corner of the main screen of the Driver Station app. This will launch a pop-up menu.	

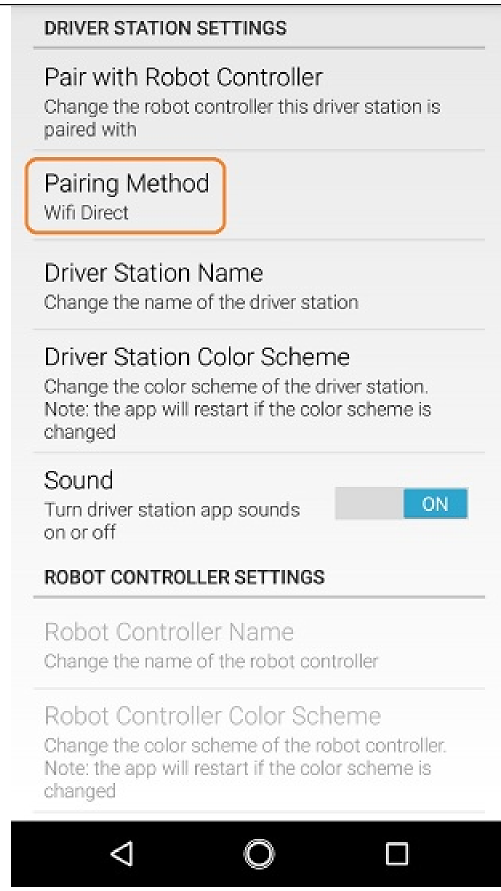
continues on next page

Table 5 – continued from previous page

Step	Image
5. Select **Settings** from the pop-up menu.	

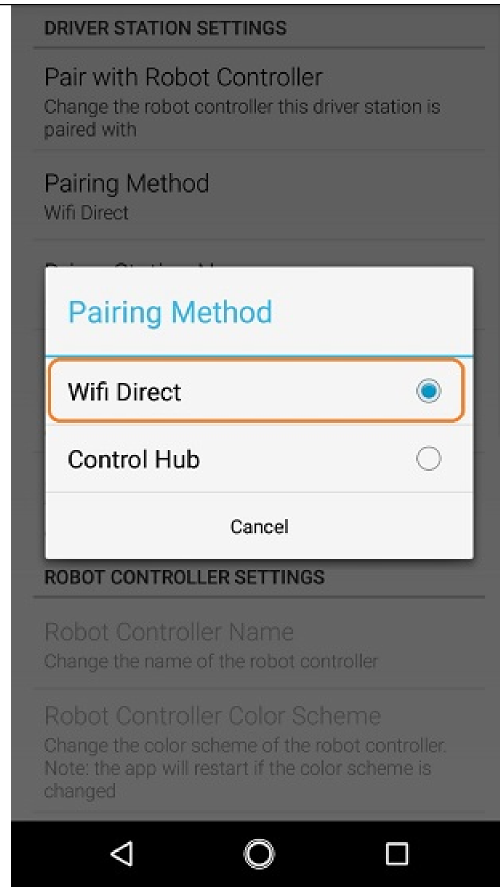
continues on next page

Table 5 – continued from previous page

Step	Image
6. From the **Settings** screen, look for and select **Pairing Method** to launch the **Pairing Method** screen.	

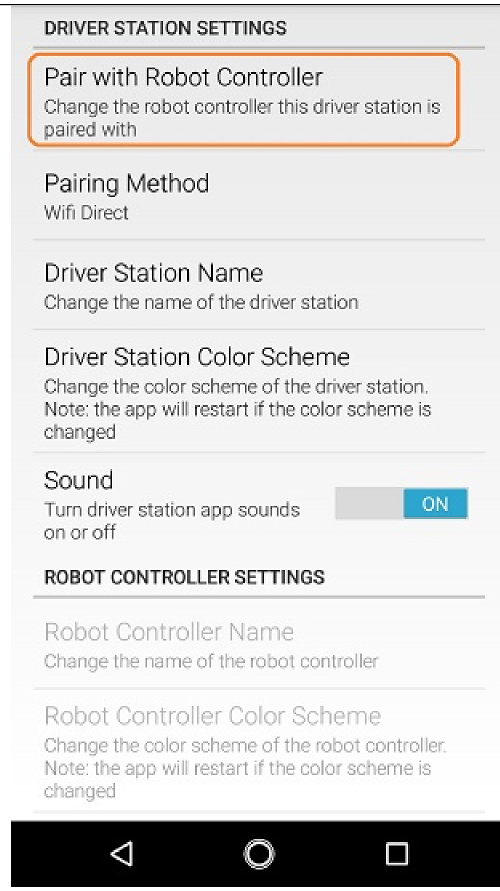
continues on next page

Table 5 – continued from previous page

Step	Image
7. Verify that the **Wifi Direct** mode is selected, which means that this DRIVER STATION will be pairing with another Android device.	

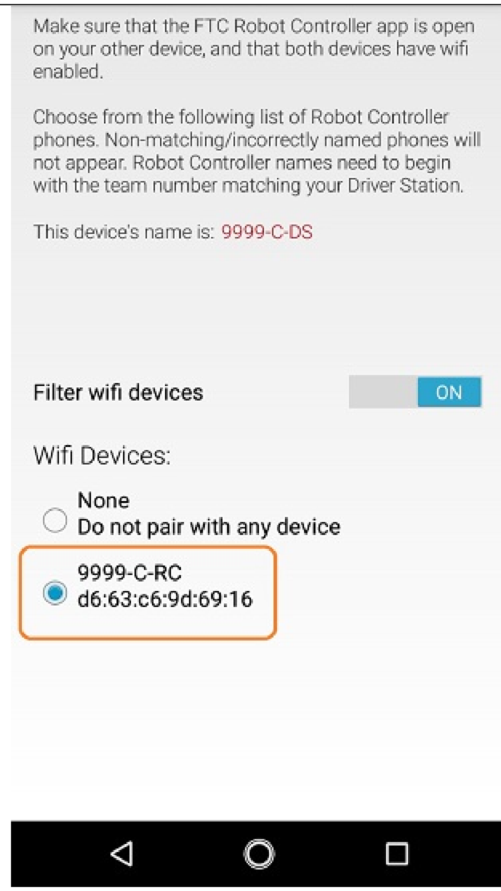
continues on next page

Table 5 – continued from previous page

Step	Image
8. From the **Settings** screen, look for and select **Pair with Robot Controller** to launch the **Pair with Robot Controller** screen.	

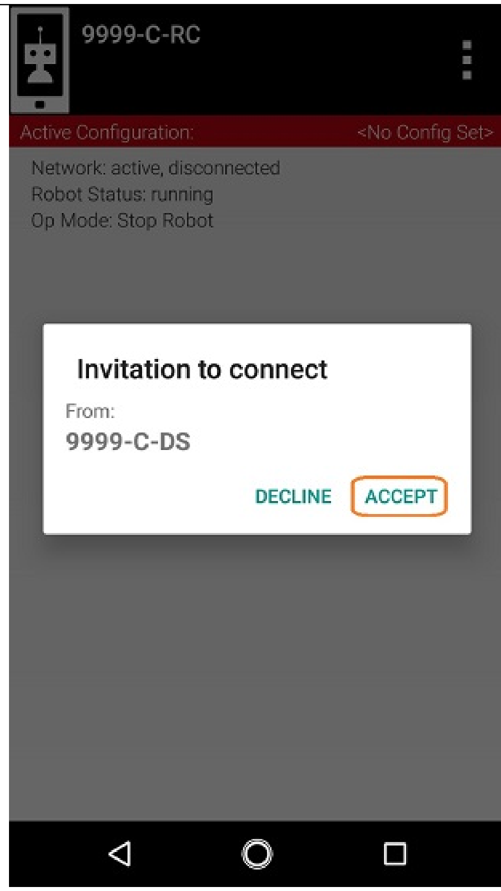
continues on next page

Table 5 – continued from previous page

Step	Image
9. Find the name of your Robot Controller from the list and select it. After you have made your selection, use the back-arrow key to return to the Settings screen. Then press the back-arrow key one more time to return to the main DRIVER STATION screen.	

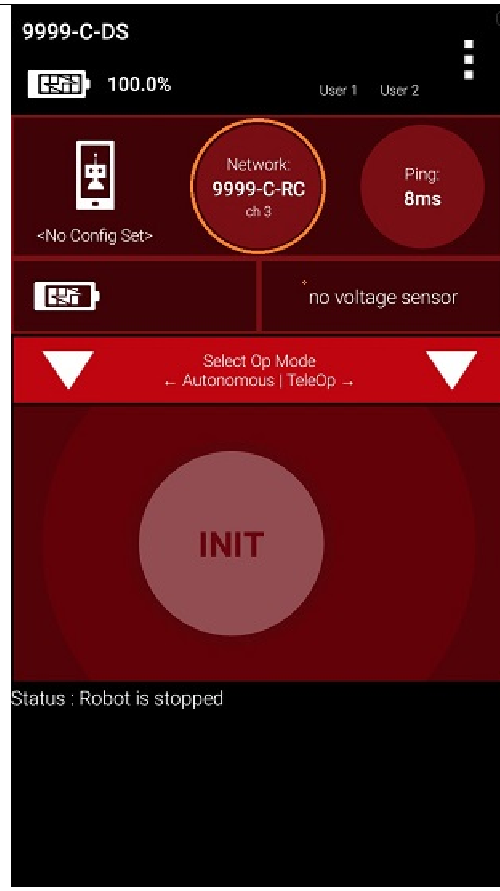
continues on next page

Table 5 – continued from previous page

Step	Image
10. When the DRIVER STATION returns to its main screen, the first time you attempt to connect to the Robot Controller a prompt should appear on the Robot Controller screen. Click on the **ACCEPT** button to accept the connection request from the DRIVER STATION.	

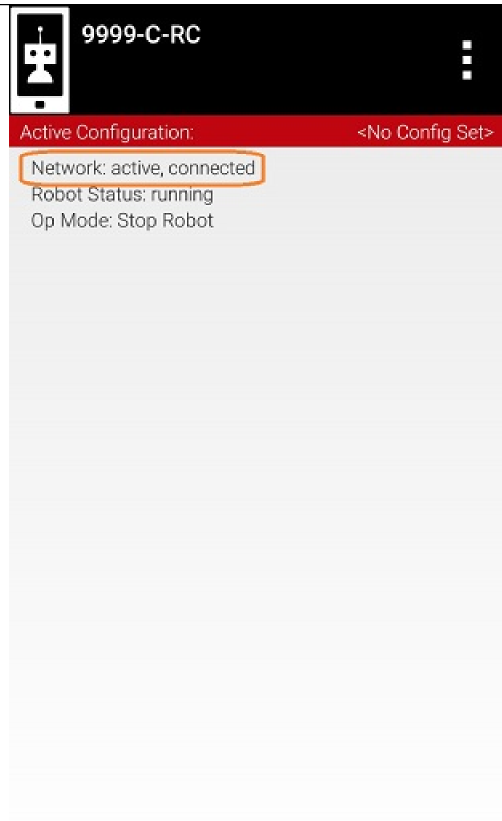
continues on next page

Table 5 – continued from previous page

Step	Image
11. Verify that the DRIVER STATION screen has changed and that it now indicates that it is connected to the Robot Controller. The name ofthe Robot Controller's remote network (9999-C-RC in this example) should be displayed in the Network field on the DRIVER STATION.	

continues on next page

Table 5 – continued from previous page

Step	Image
12. Verify that the Robot Controller screen has changed and that it now indicates that it is connected to the DRIVER STATION.The Network status should read active, connected on the Robot Controller's main screen.	

Connecting to the Program & Manage Server Blocks

Installing a Javascript Enabled Browser

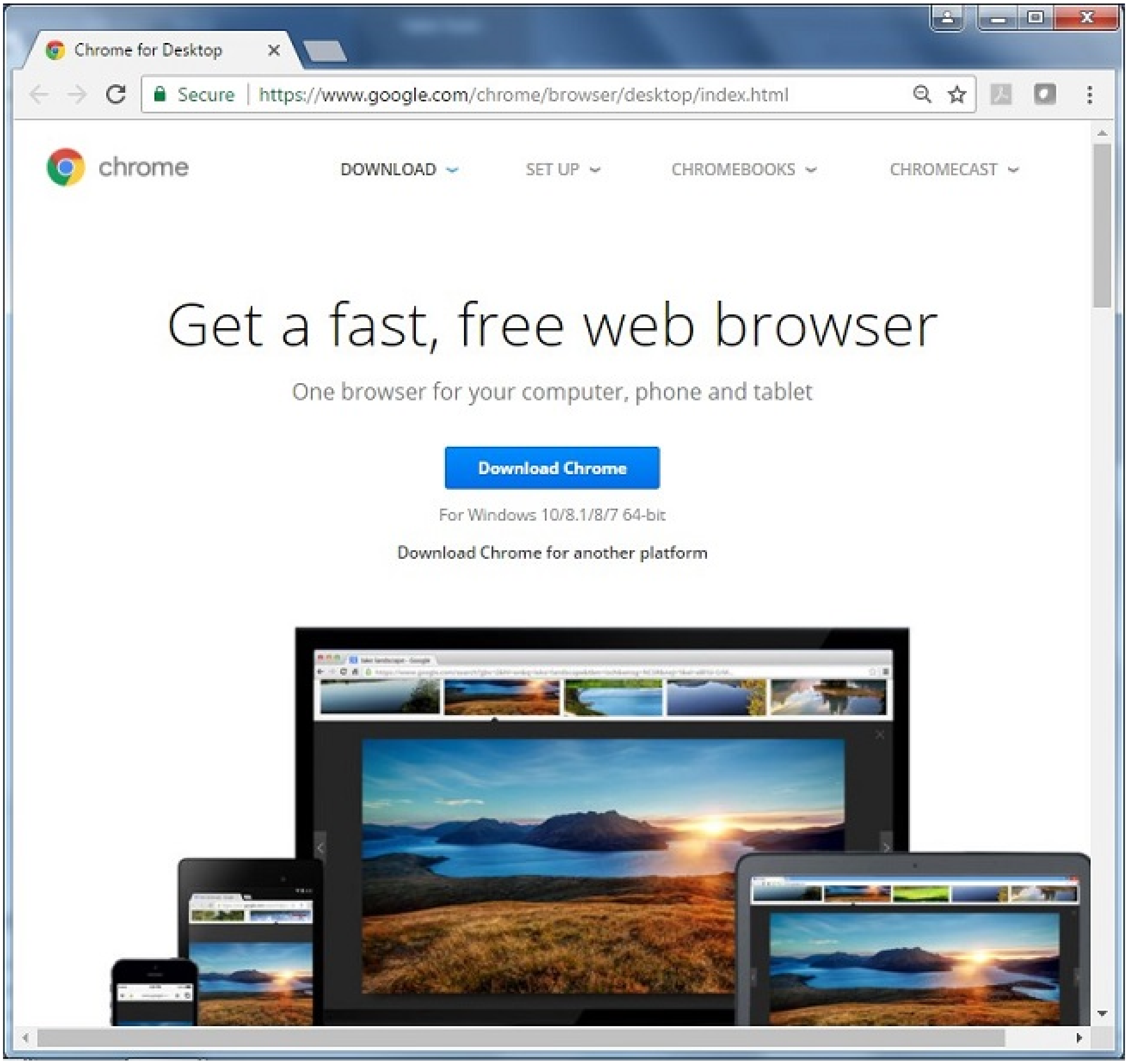
In order to be able to program your Robot Controller using the Blocks Programming Tool or the OnBot Java Programming tool, your laptop will need a Javascript-enabled browser. Both tools are Javascript applications that are served up by the Program and Manage server of the Robot Controller.

The Blocks Programming Tool and the OnBot Java Programming Tool should work with most modern web browsers. However, *FIRST* strongly recommends the use of Google Chrome with these tools. If you would like to use Google Chrome as your browser, you can download it for free from the Google Chrome website.

Note that it will take an estimated 15 minutes (depending on the speed of your Internet connection) to download and install the Javascript-enabled browser.

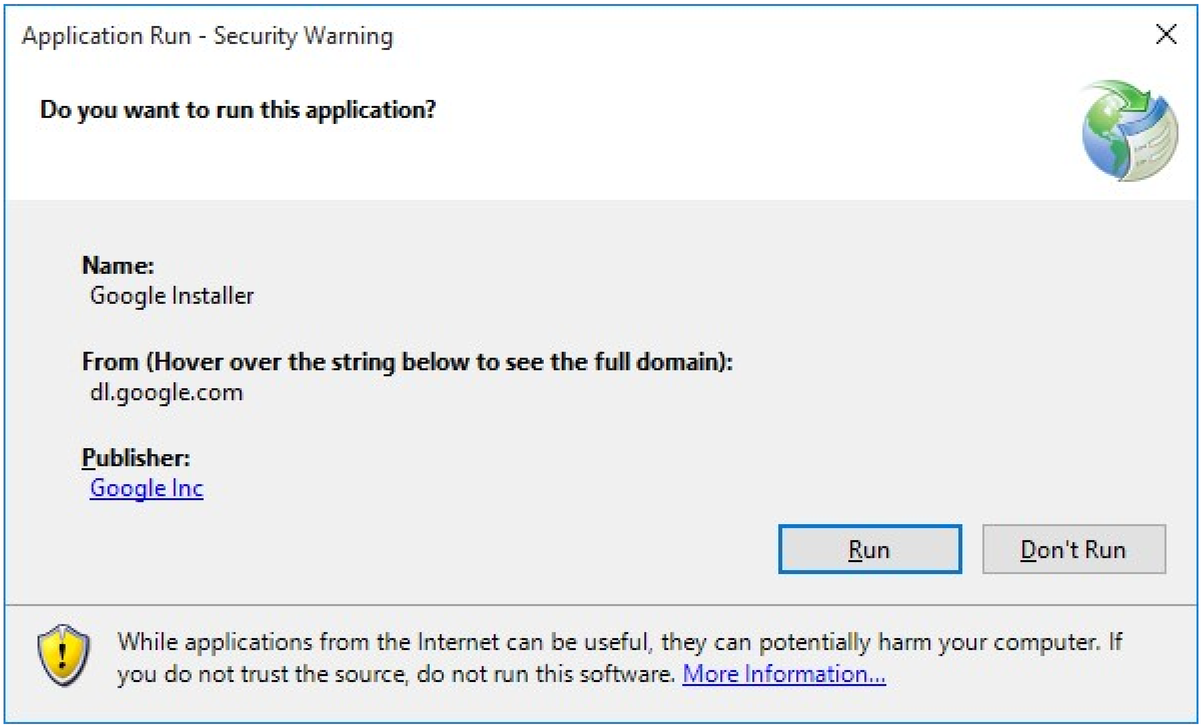
Installing a Javascript-Enabled Browser Instructions

1. Visit the Google Chrome Browser website (using your computer's existing browser) and follow onscreen instructions to download and install Chrome.



Chrome Browser Website Link

2. Note that your computer might prompt you with a security warning during the installation process. If you are prompted with this warning, click on the "Run" button to continue with the installation.



Connecting a Laptop to the Program & Manage Network

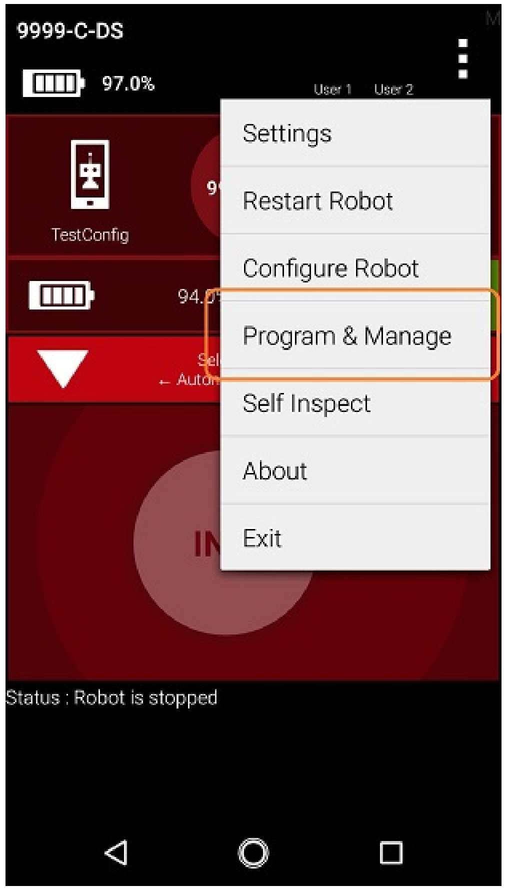
Connecting Your Laptop to the Program & Manage Network

In order to write an Op Mode, you will need to connect your programming laptop to the Program & Manage Wi-Fi network. The Program & Manage Wi-Fi network is a wireless network created by your Robot Controller. Before you begin this exercise, please make sure that your Windows laptop has the most current service pack and system update from Microsoft installed.

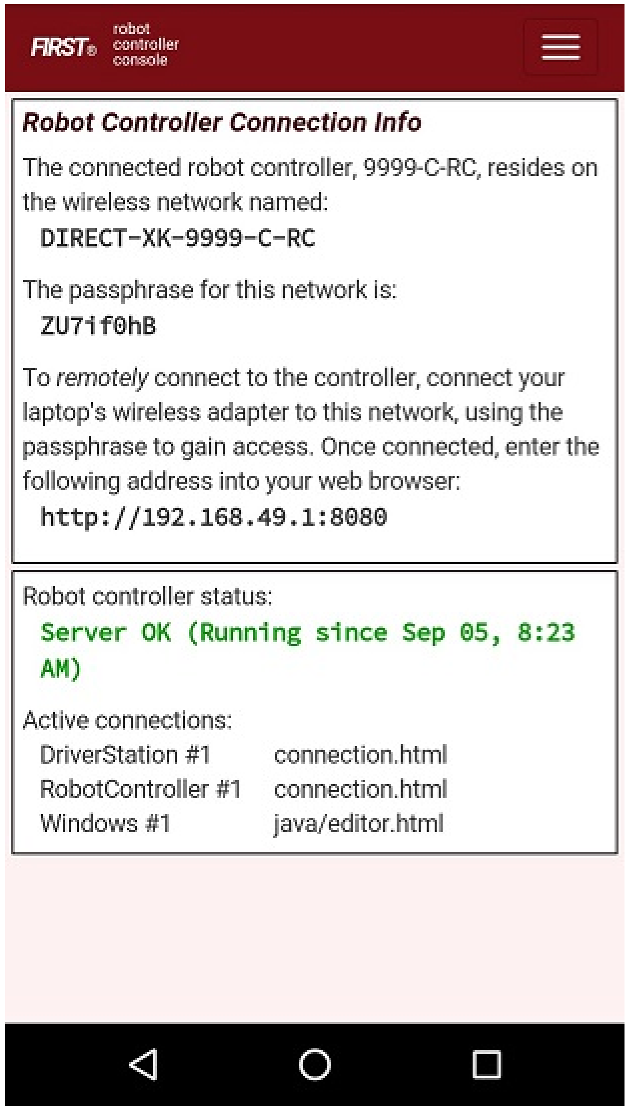
Note that this example assumes the user has a Windows 10 laptop. If you are not using a Windows 10 laptop, the procedure to connect to the Programming & Manage Wi-Fi network will differ. Refer to your device's documentation for details on how to connect to a Wi-Fi network.

Connecting Your Laptop to the Program & Manage Network Instructions

1. On the DRIVER STATION, touch the three dots in the upper right hand corner of the screen to launch the pop-up menu. Select **Program & Manage** from the pop-up menu to display the **Program & Manage** access information.

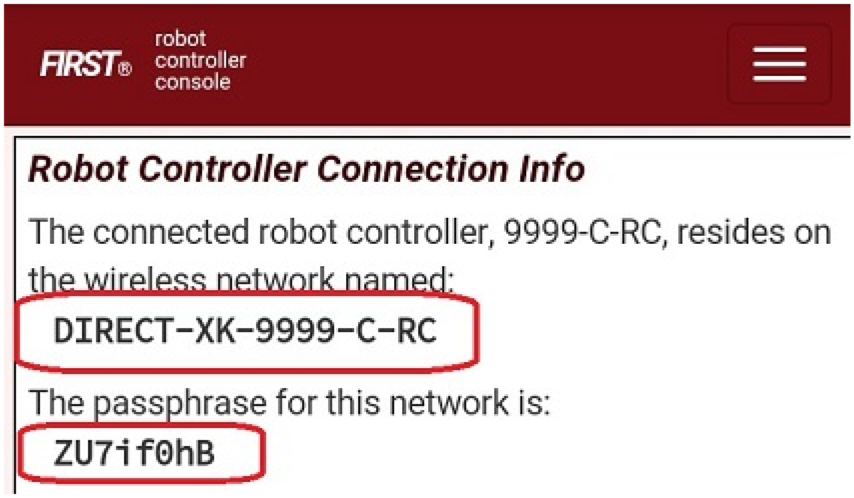


2. The Program & Manage screen displays important information that you can use to connect your laptop to the Blocks or OnBot Java Programming Mode server.



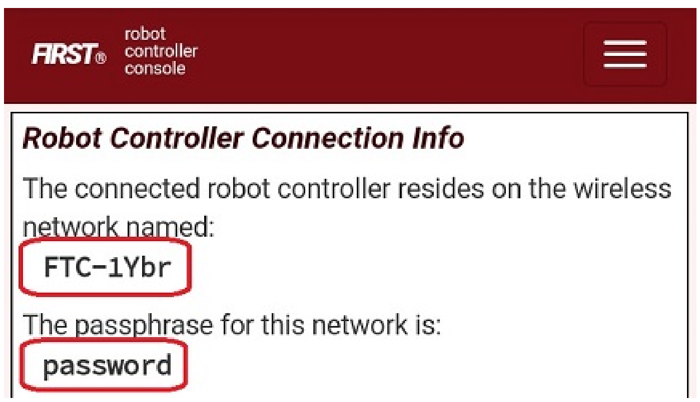
3. Verify the network name and passphrase for the Program & Manage wireless network. Towards the top of the screen, the name of the Program & Manage wireless network is displayed. If you are using an Android smartphone as your Robot Controller, then the wireless network name will begin with the phrase “DIRECT-“.

In this example, the name of the Wi-Fi network is “DIRECT-XK-9999-C-RC” and the secure passphrase is “ZU7if0hB”

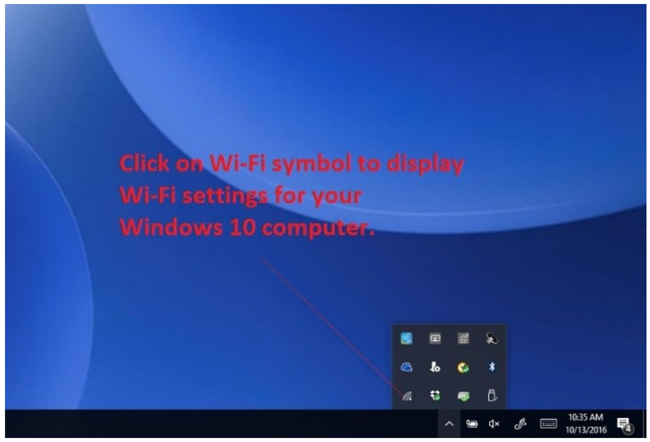


If you are using a Control Hub, then the wireless network name will be whatever you specified when you configured your Control Hub. If you haven't changed the Control Hub's name yet, then by default the wireless network's name will begin with "FTC-". If you haven't changed its password yet, then by default the wireless network's passphrase will be "password".

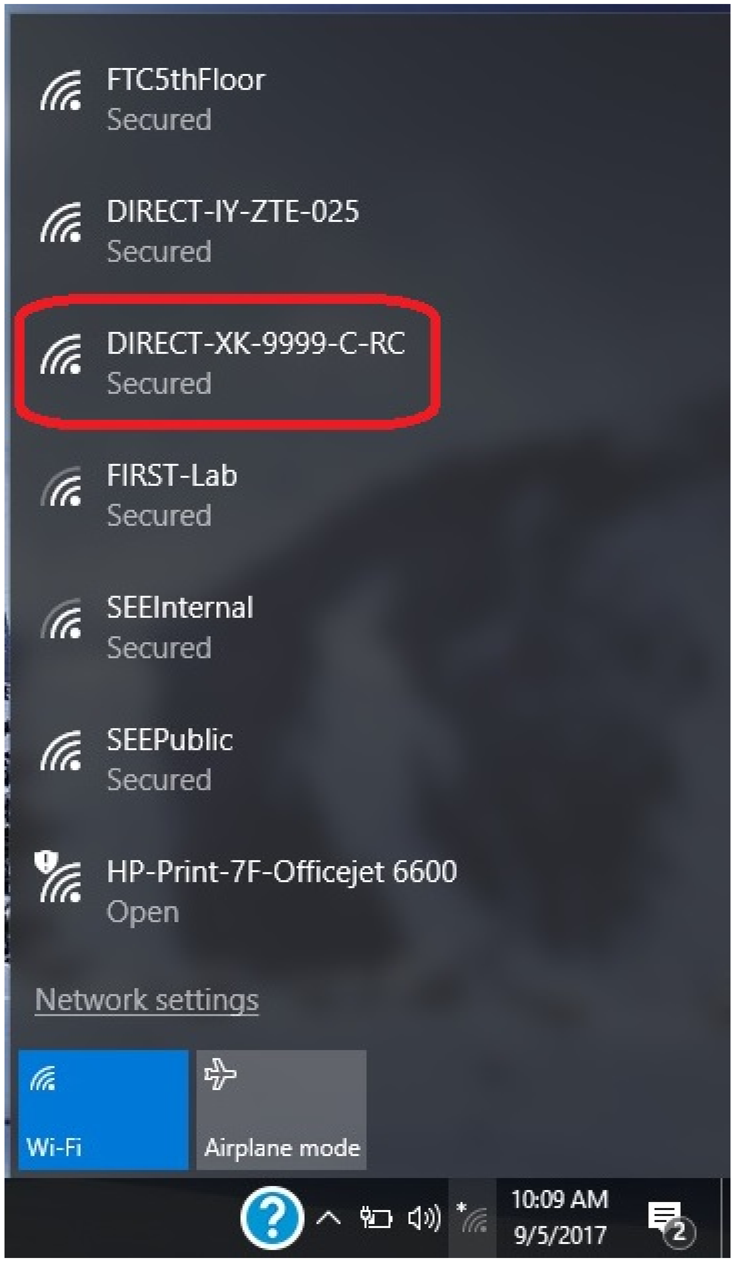
In the screenshot below, the Control Hub's wireless network name is "FTC-1Ybr" and the secure passphrase is "password".



4. On your Windows 10 computer, look in the lower right hand corner of your desktop for a Wi-Fi symbol. Click on the Wi-Fi symbol to display a list of available Wi-Fi Networks in your vicinity.

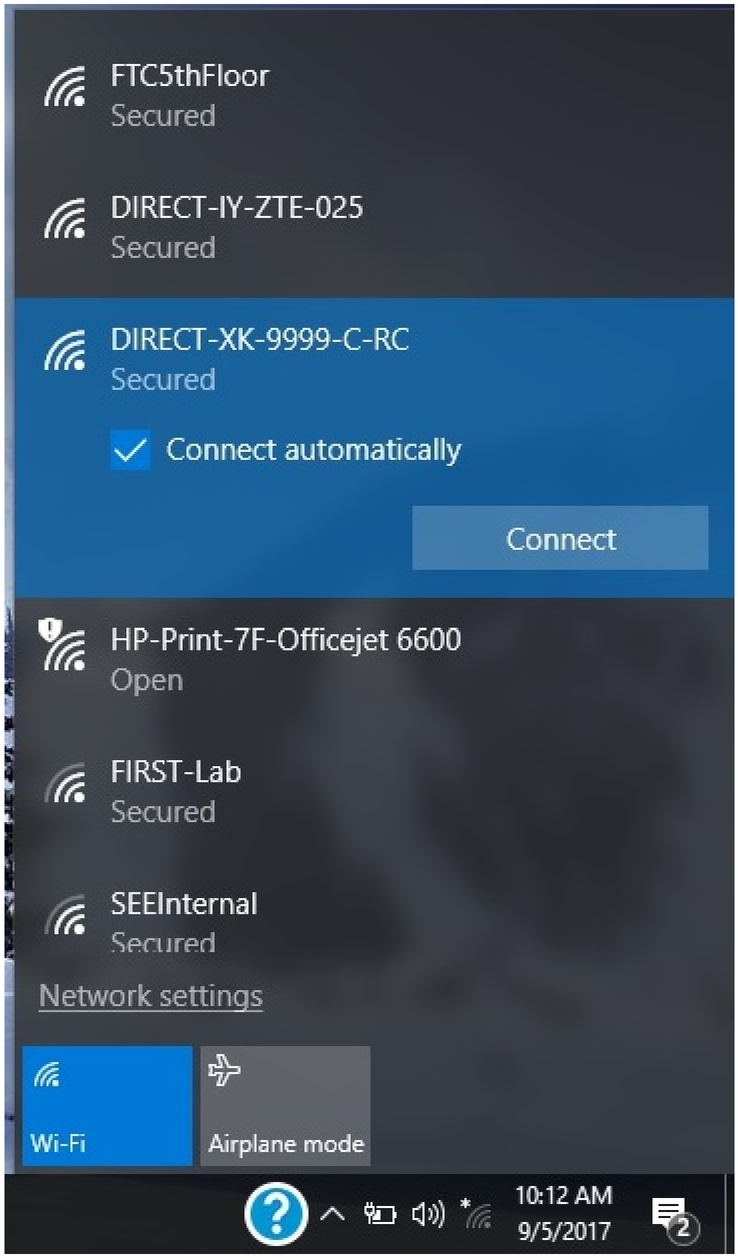


5. Look for the wireless network that matches the name displayed on the Program & Manage screen.



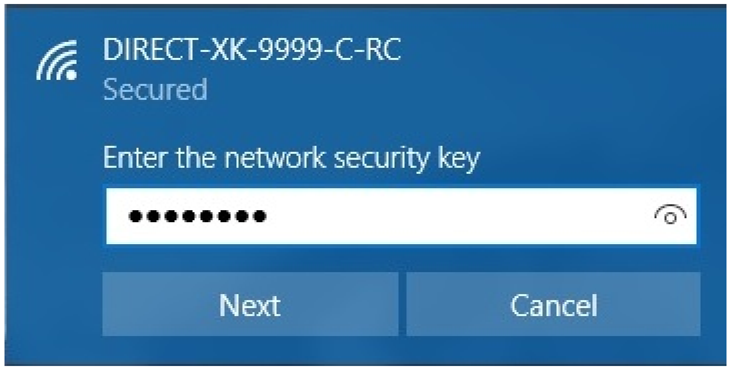
In this example, the name of the wireless network for the Android Robot Controller is "DIRECT-XK-9999-C-RC" and the network is visible in the list displayed on the Windows 10 computer.

6. Once you have found the target network in the list, click on it to select it.



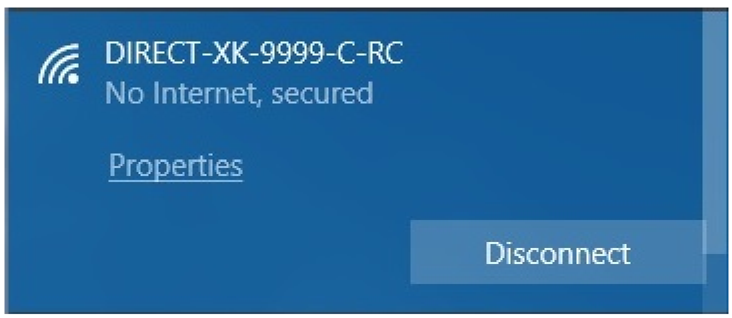
Press the Connect button to connect to the network.

7. When prompted, provide the network passphrase (in this example “ZU7if0hB”) and press “Next” to continue.



Note that the passphrase is case sensitive. Make sure that your spelling and capitalization matches the original spelling and capitalization shown on the Program & Manage screen.

8. Once you have successfully established a wireless connection between your Windows 10 laptop and your Robot Controller Android device, the status should be displayed in the wireless settings for your laptop.



If the display is not updated as shown after a few seconds, try clicking on Network Connections at the bottom of the blue box showing the Wi-Fi connections. This will bring up a Setting dialog box that includes a link to "Show available networks", which can be used to force the list of Wi-Fi connections to be updated.

> **Attention:** Note that when you are connected to the blocks programming mode server on your Robot Controller, your laptop **will not have access to the Internet**. It only has direct access to the Robot Controller.

Troubleshooting Your Wireless Connection

If you cannot see your Programming Mode wireless network in the list of available networks, or, if you are having problems connecting your laptop to the Program & Manage wireless network, make sure you answer the following questions:

1. Is the Robot Controller running and connected to the DRIVER STATION?
2. Is your Windows laptop updated with the most current system updates and service packs? Older versions of Windows 8 and 10, for example, had issues that could prevent the laptop from displaying the Program & Manage wireless network in the list of available networks.

Writing an Op Mode Blocks

Creating Op Modes Blocks

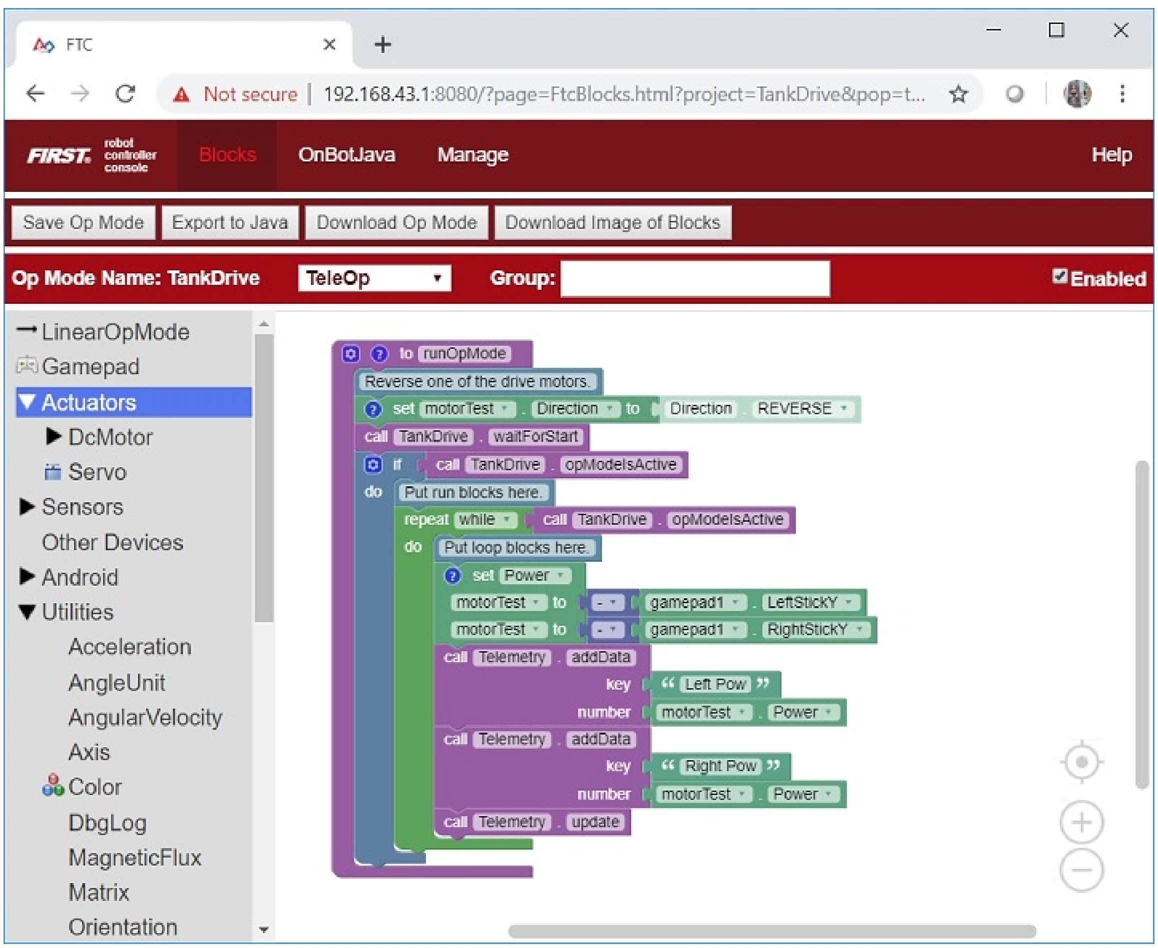
What's an Op Mode?

During a typical *FIRST* Tech Challenge match, a team's robot must perform a variety of tasks to score points. For example, a team might want their robot to follow a white line on the competition floor and then score a game element into a goal autonomously during a match. Teams write programs called *op modes* (which stands for "operational modes") to specify the behavior for their robot. These op modes run on the Robot Controller after being selected on the DRIVER STATION.

Teams who are participating in the *FIRST* Tech Challenge have a variety of programming tools that they can use to create their own op modes. This section of the wiki explains how to use the Blocks Programming Tool to write an op mode for a robot.

The Blocks Programming Tool

The Blocks Programming Tool is a user-friendly programming tool that is served up by the Robot Controller. A user can create custom op modes for their robot using this tool and then save these op modes directly onto the Robot Controller. Users drag and drop jigsaw-shaped programming blocks onto a design "canvas" and arrange these blocks to create the program logic for their op mode. The Blocks Programming Tool is powered by Google's Blockly software and was developed with support from Google.



The examples in this section use a Windows laptop computer to connect to the Robot Controller. This Windows laptop computer has a Javascript-enabled web browser installed that is used to access the Blocks Programming Tool.

Note that the process used to create and edit an op mode is identical if you are using a Control Hub as your Robot Controller.

Note that if you prefer, you can use an alternate device, such as an Apple Mac laptop, an Apple iPad, an Android tablet, or a Chromebook, instead of a Windows computer to access the Blocks Programming Tool. The instructions included in this document, however, assume that you are using a Windows laptop.

Also note that this section of the wiki assumes that you have already setup and configured your Android devices and robot hardware. It also assumes that you have successfully connected your laptop to the Robot Controller's Progam & Manage wireless network.

Creating Your First Op Mode

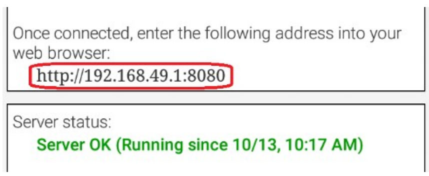
If you connected your laptop successfully to the Program & Manage wireless network of the Robot Controller, then you are ready to create your first op mode. In this section, you will use the Blocks Programming Tool to create the program logic for your first op mode.

Note that it will take an estimated 10 minutes to create your first op mode.

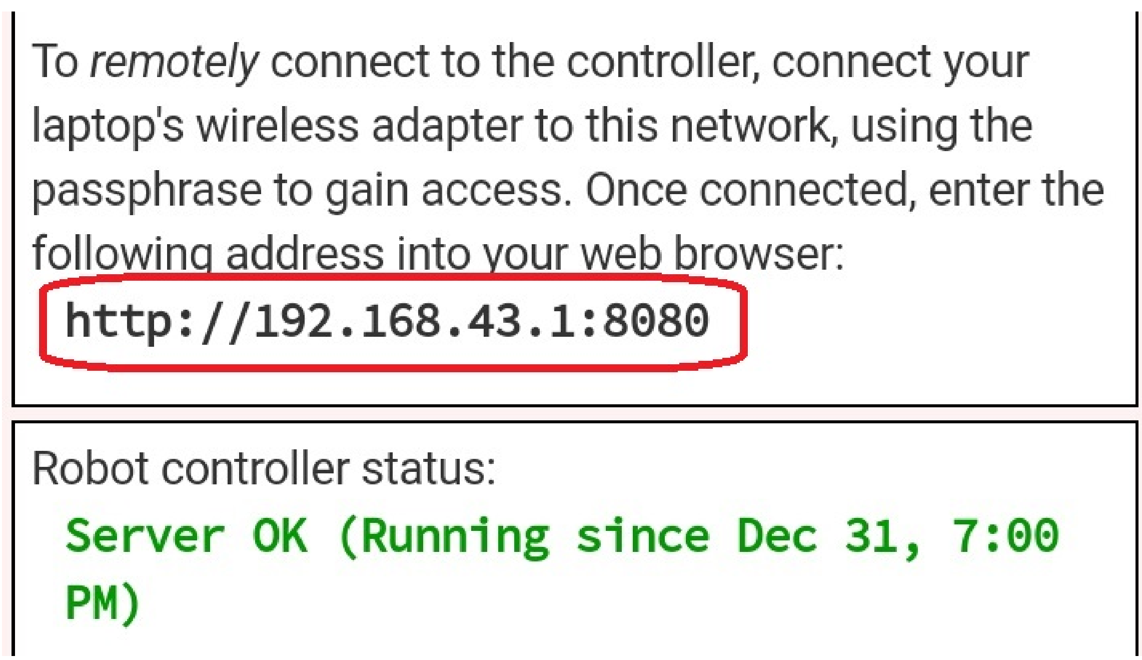
Creating Your First Op Mode Instructions

1. Launch the web browser on your laptop (FIRST recommends using Google Chrome) and find the web address that is displayed on the Program & Manage screen of the Robot Controller.

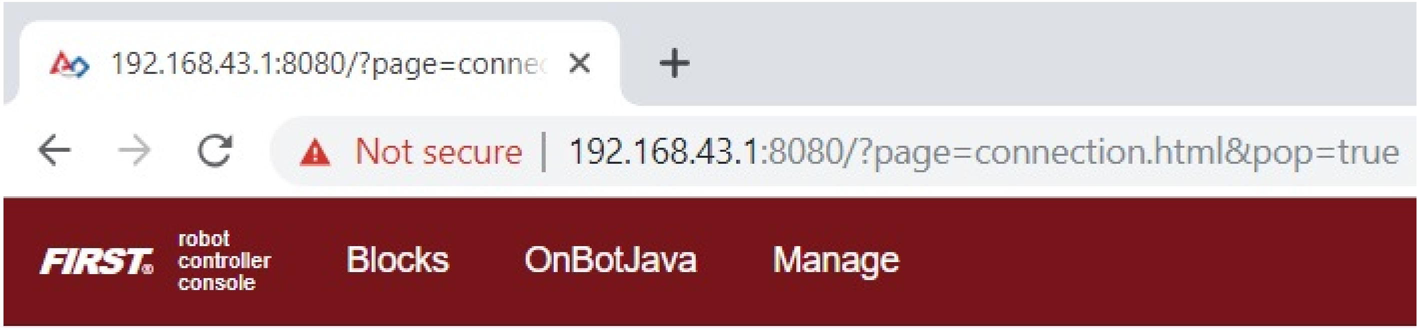
Important: If your Robot Controller is an Android smartphone, then the address to access the Program & Manage server is "192.168.49.1:8080".



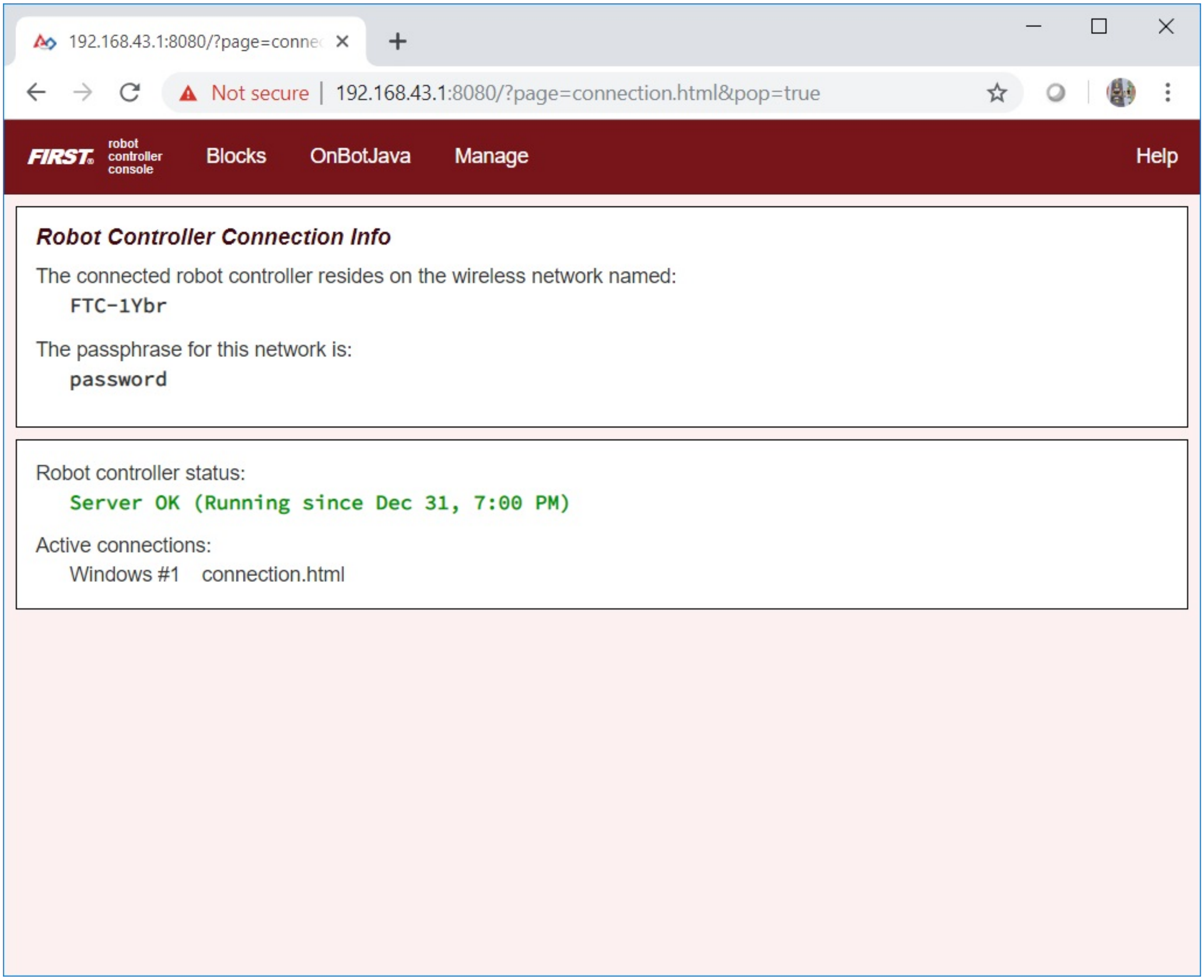
Important: If your Robot Controller is a Control Hub, then the address to access the Program & Manage server is "192.168.43.1:8080". Notice the difference in the third octet of the IP addresses (the Control Hub has a "43" instead of a "49").



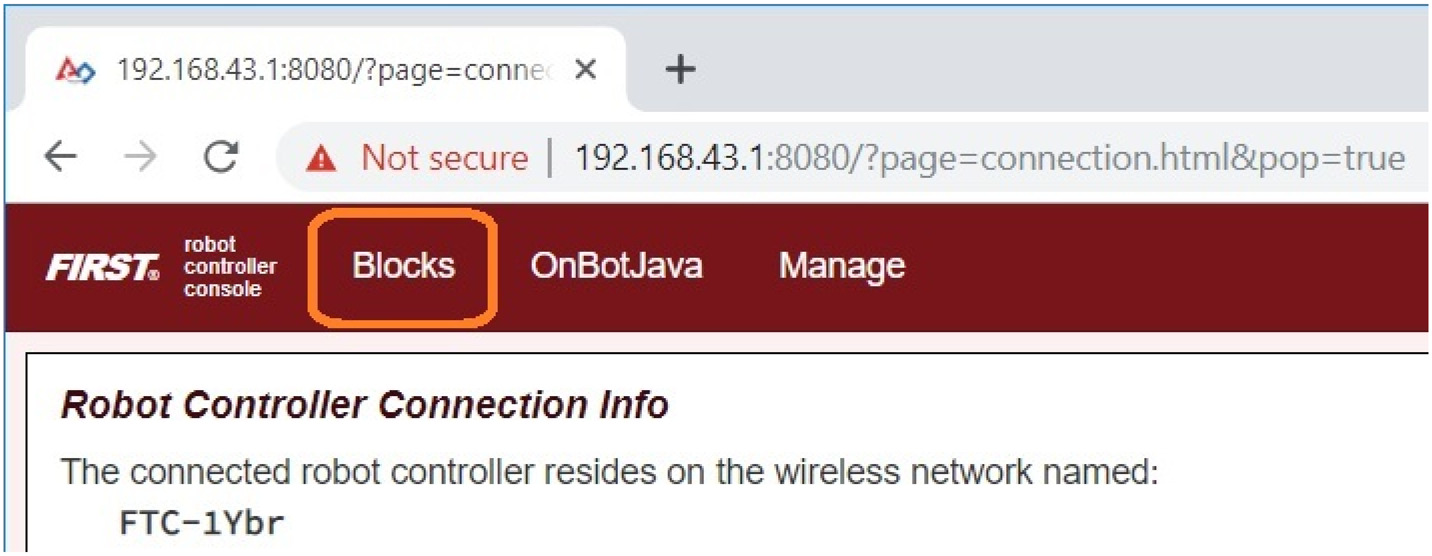
Type this web address into the address field of your browser and press RETURN to navigate to the Program & Manage web server.



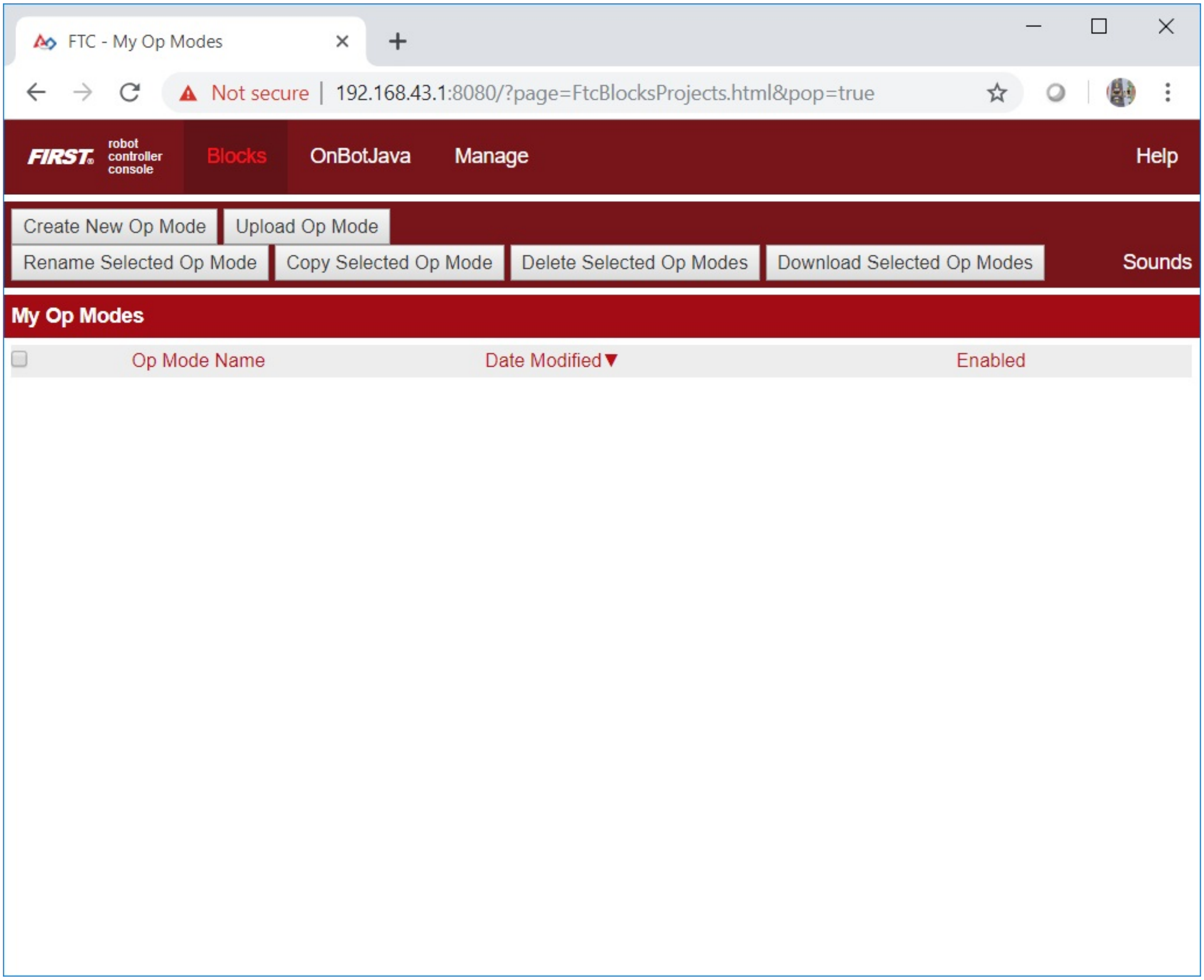
2. Verify that your web browser is connected to the programming mode server. If it is connected to the programming mode server successfully, the Robot Controller Console should be displayed.



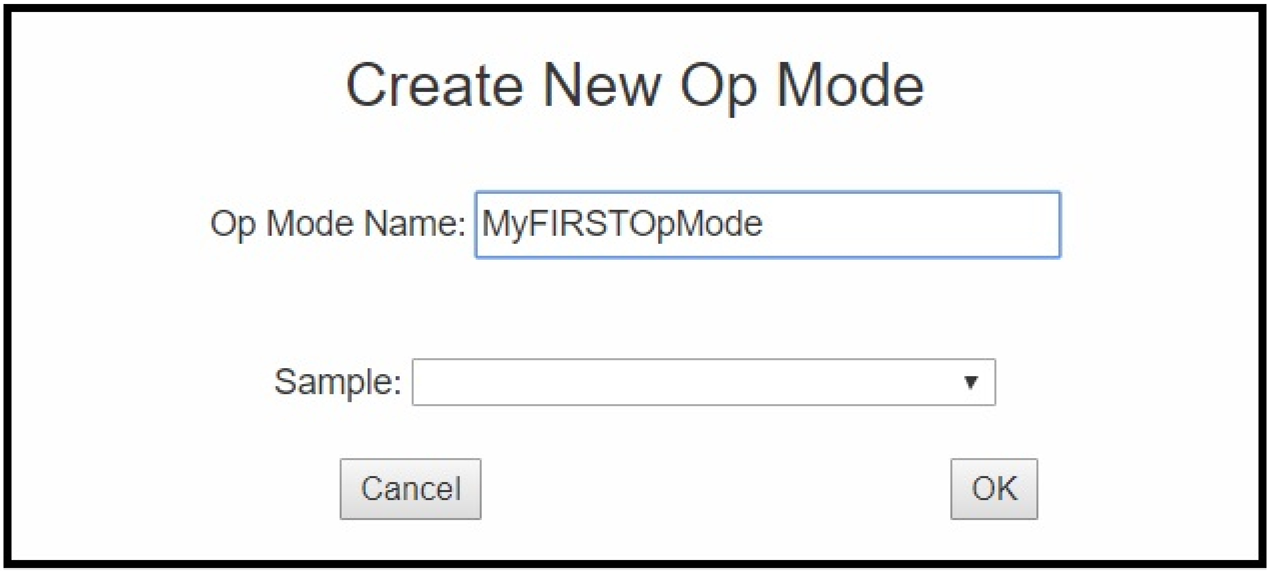
3. Press the **Blocks** link towards the top of the Console to navigate to the main Blocks Programming screen.



The main Blocks Programming screen is where you create new op modes. It is also the screen where you can see a list of existing Blocks Op Modes on a Robot Controller. Initially this list will be empty until you create and save your first op mode.

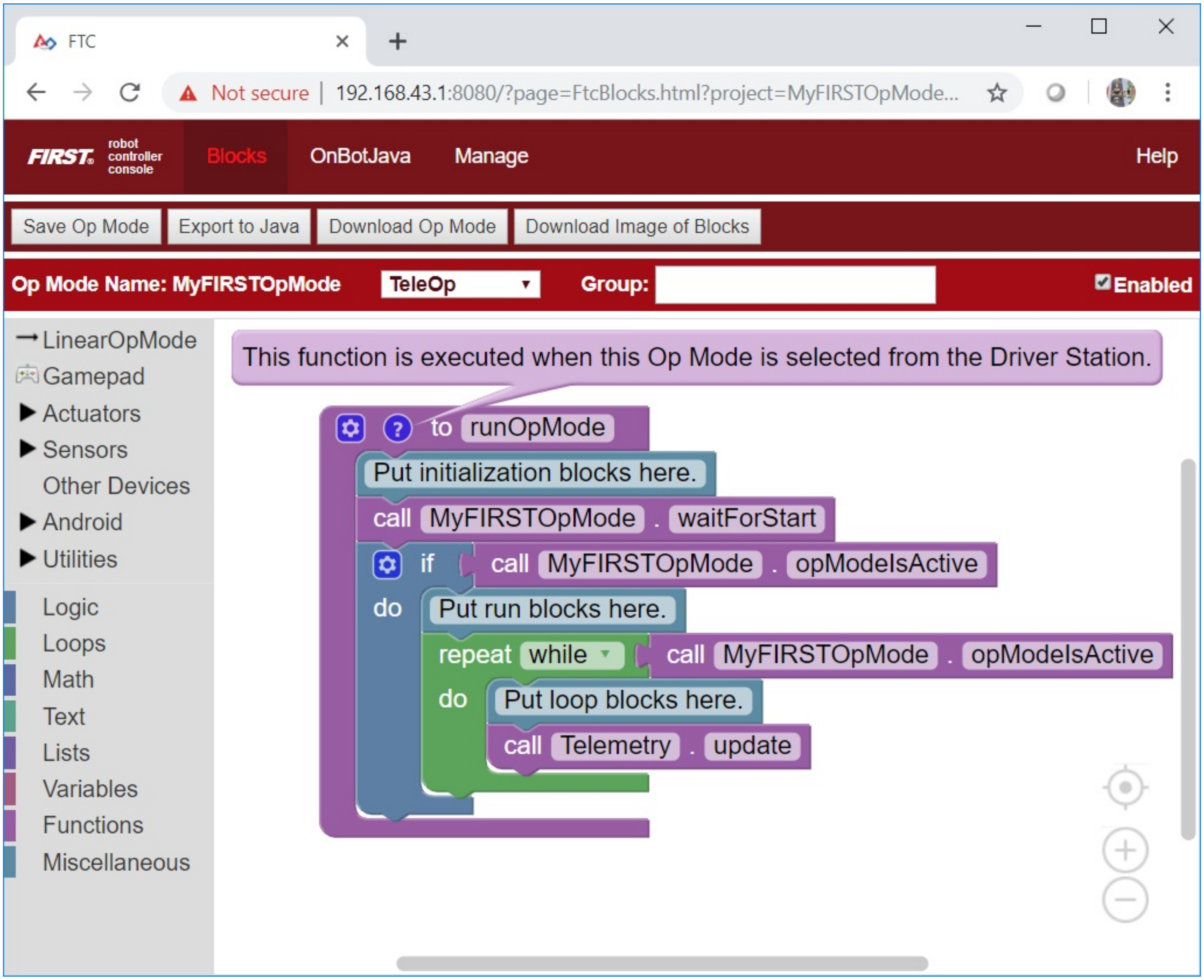


4. Press the “Create New Op Mode” button which should be visible towards the upper left hand corner of the browser window.



When prompted, specify a name for the op mode and hit “OK” to continue.

5. Verify that you created the new op mode. You should see your newly created op mode opened for editing in your web browser’s main screen.

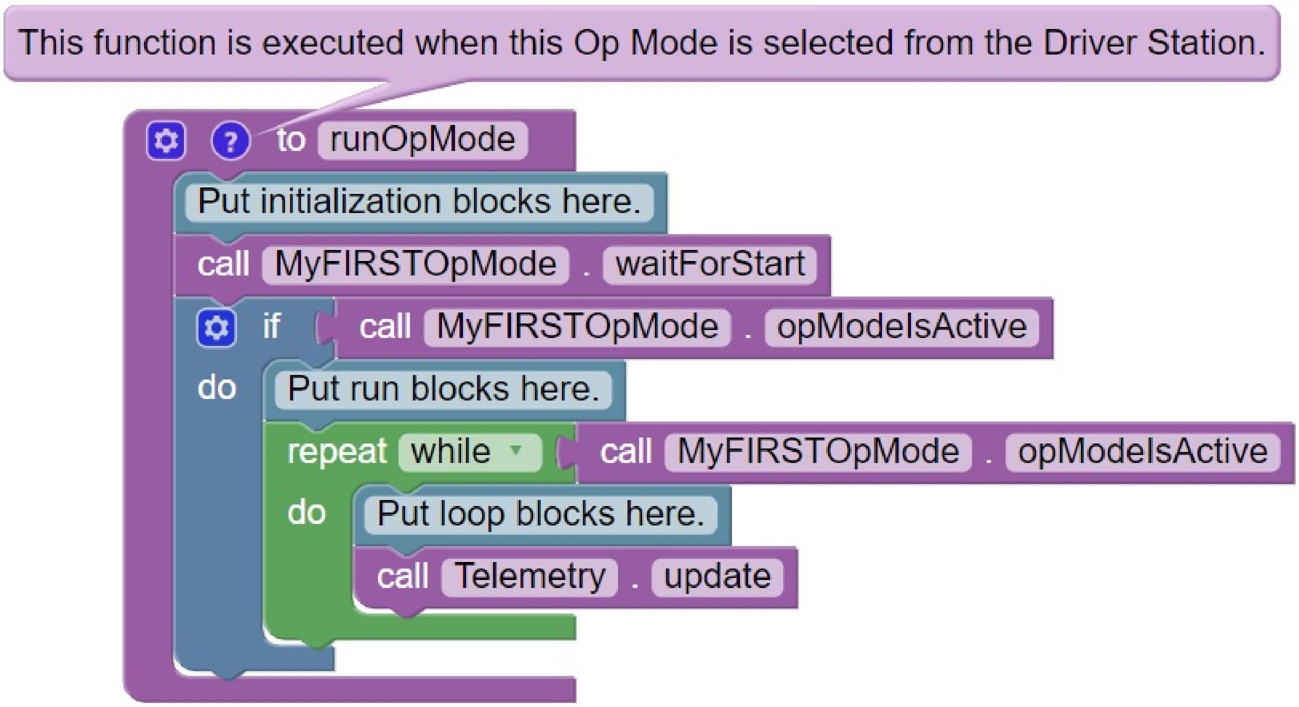


Notice that the left-hand side of the browsers screen contains a list of categorized programming blocks. If you click on a category, the browser will display a list of available related programming blocks.

The right-hand side of the screen is where you arrange your programming blocks to create the logic for your op mode.

Examining the Structure of Your Op Mode

When you create a new op mode, there should already be a set of programming blocks that are placed on the design canvas for your op mode. These blocks are automatically included with each new op mode that you create. They create the basic structure for your op mode.

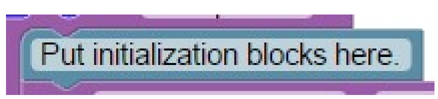


In the figure shown above, the main body of the op mode is defined by the outer purple bracket that has the words "to runOpMode" at the top. As the help tip indicates, this function is executed when this op mode ("MyFIRSTOpMode" in this example) is selected from the DRIVER STATION.

It can be helpful to think of an op mode as a list of tasks for the Robot Controller to perform. The Robot Controller will process this list of tasks sequentially. Users can also use control loops (such as a while loop) to have the Robot Controller repeat (or iterate) certain tasks within an op mode.

If you think about an op mode as a list of instructions for the robot, this set of instructions will be executed by the robot whenever a team member selects the op mode called “MyFIRSTOpMode” from the list of available op modes for this Robot Controller.

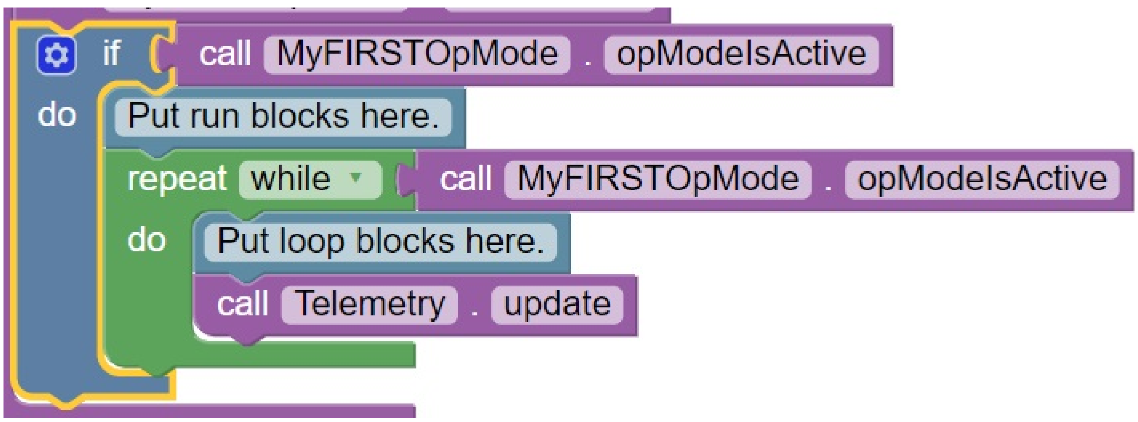
You can hide the help text by clicking on the blue button with the question mark (“?”) on it. Let’s look at the flow of this basic op mode. The blue colored block with the words “Put initialization blocks here” is a comment. Comments are placed in an op mode for the benefit of the human user. The robot will ignore any comments in an op mode.



Any programming blocks that are placed after the “Put initialization blocks here” comment (and before the “call MyFIRSTOp-Mode.waitForStart” block) will be executed when the op mode is first selected by a user at the DRIVER STATION.

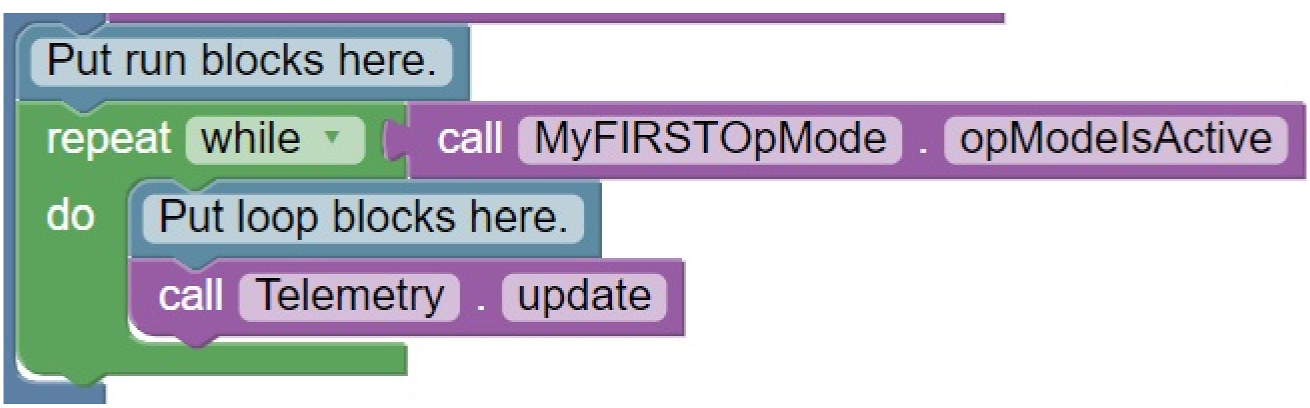
When the Robot Controller reaches the block labeled “call MyFIRSTOpMode.waitForStart” it will stop and wait until it receives a Start command from the DRIVER STATION. A Start command will not be sent until the user pushes the Start button on the DRIVER STATION. Any code after the “call MyFIRSTOpMode.waitForStart” block will get executed after the Start button has been pressed.

After the “call MyFIRSTOpMode.waitForStart”, there is a conditional “if” block (“if call MyFIRSTOpMode.isActive”) that only gets executed if the op mode is still active (i.e., a stop command hasn’t been received).



Any blocks that are placed after the “Put run blocks here” comment and before the green block labeled “repeat while call My-FirstOpMode.opModeIsActive” will be executed sequentially by the Robot Controller after the Start button has been pressed.

The green block labeled “repeat while call MyFirstOpMode.opModeIsActive” is an iterative or looping control structure.



This green control block will perform the steps listed under the "do" portion of the block as long as the condition "call MyFIRSTOpMode.opModeIsActive" is true. What this means is that the statements included in the "do" portion of the block will repeatedly be executed as long as the op mode "MyFIRSTOpMode" is running. Once the user presses the Stop button, the "call MyFIRSTOpMode.opModeIsActive" clause is no longer true and the "repeat while" loop will stop repeating itself.

Controlling a DC Motor

In this section, you will add some blocks to your op mode that will allow you to control a DC motor with a gamepad.

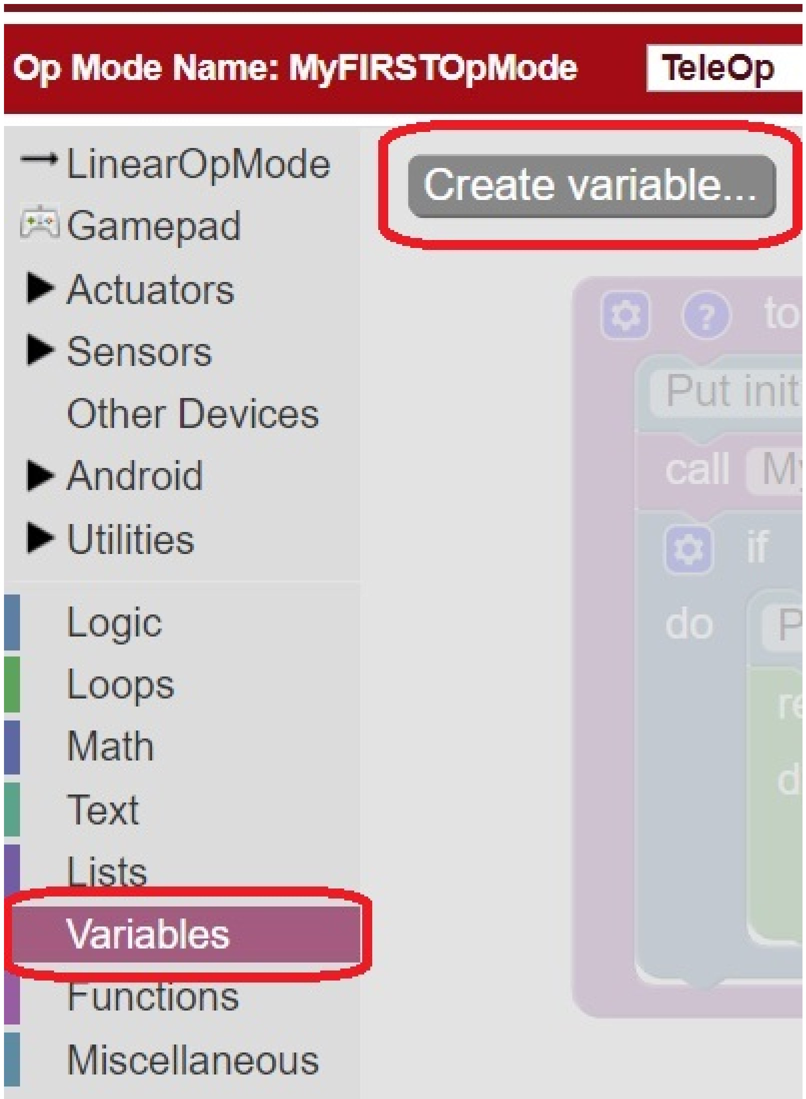
Note that you will need an estimated 15 minutes to complete this task.

Important: The programming blocks for user configured devices (motors, servos and sensors) will only be visible in the Blocks tool if there is an active configuration file with the configured devices included in the file. If a type of device is not included in the active configuration file, then its programming blocks will be missing from the palette of blocks.

If you did not *create and activate a configuration file yet* please follow *this link* to do so. After you created and activated your configuration file, you can close and then reopen your op mode so that the programming blocks for the newly configured devices will be visible.

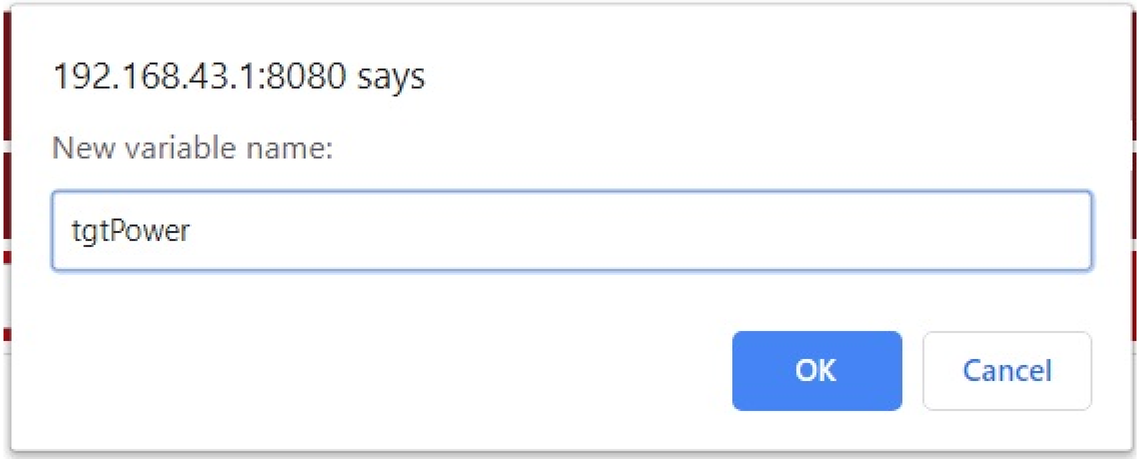
Modifying Your Op Mode to Control a DC Motor Instructions

1. On the left-hand side of the screen click on the category called "Variables" to display the list of block commands that are used to create and modify variables within your op mode.

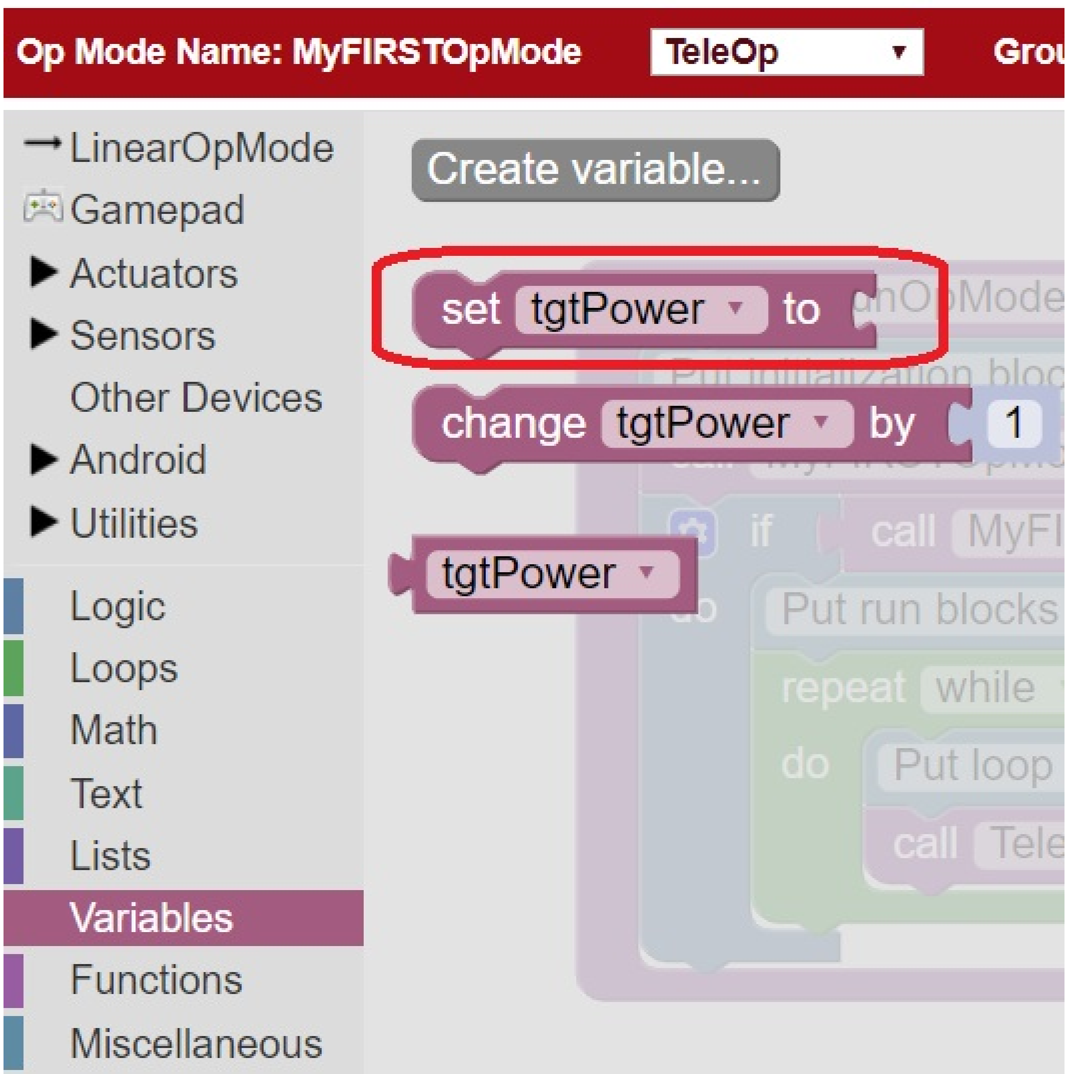


Click on "Create variable…" to create a new variable that will represent the target motor power for our op mode.

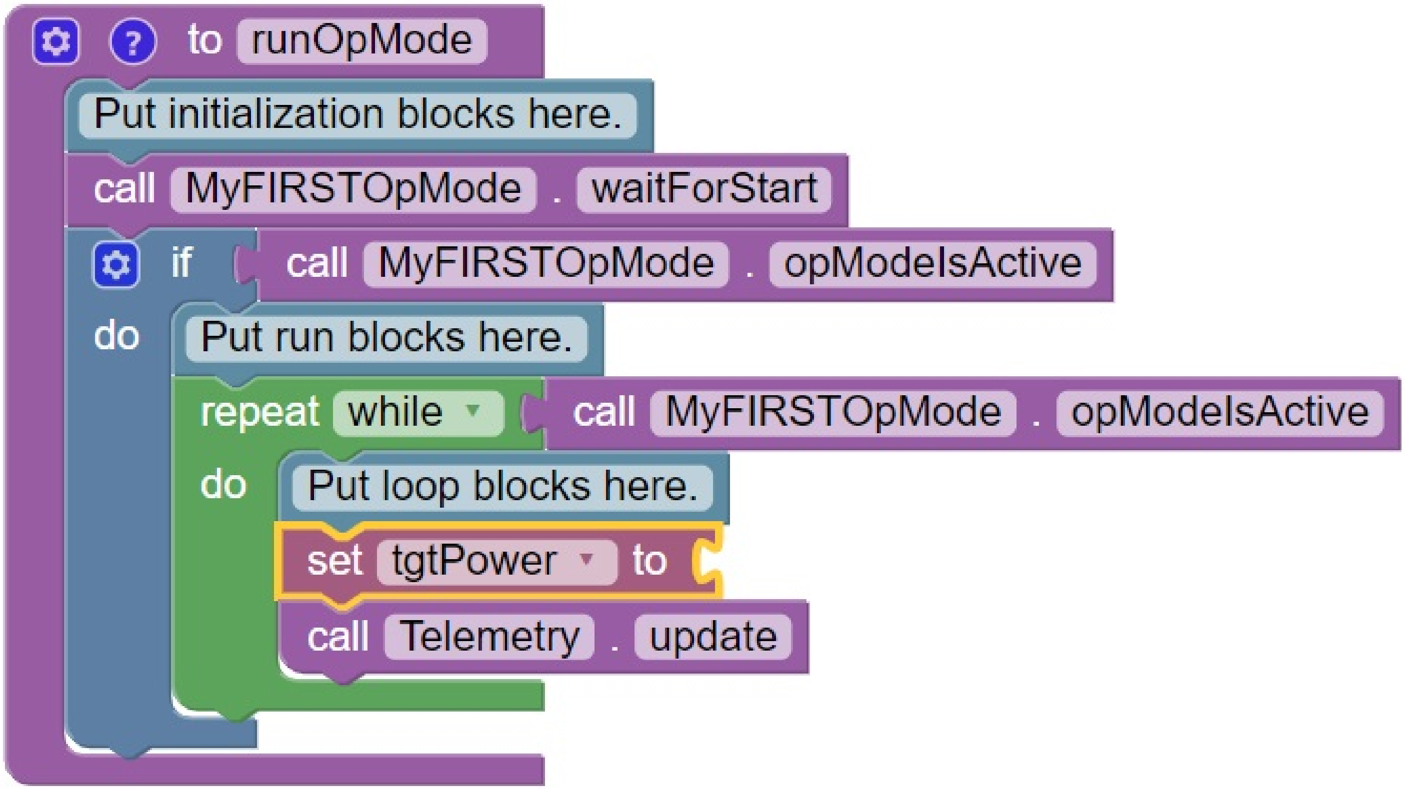
2. When prompted, type in a name ("tgtPower") for your new variable.



3. Once you have created your new variable, some additional programming blocks should appear under the "Variables" block category.

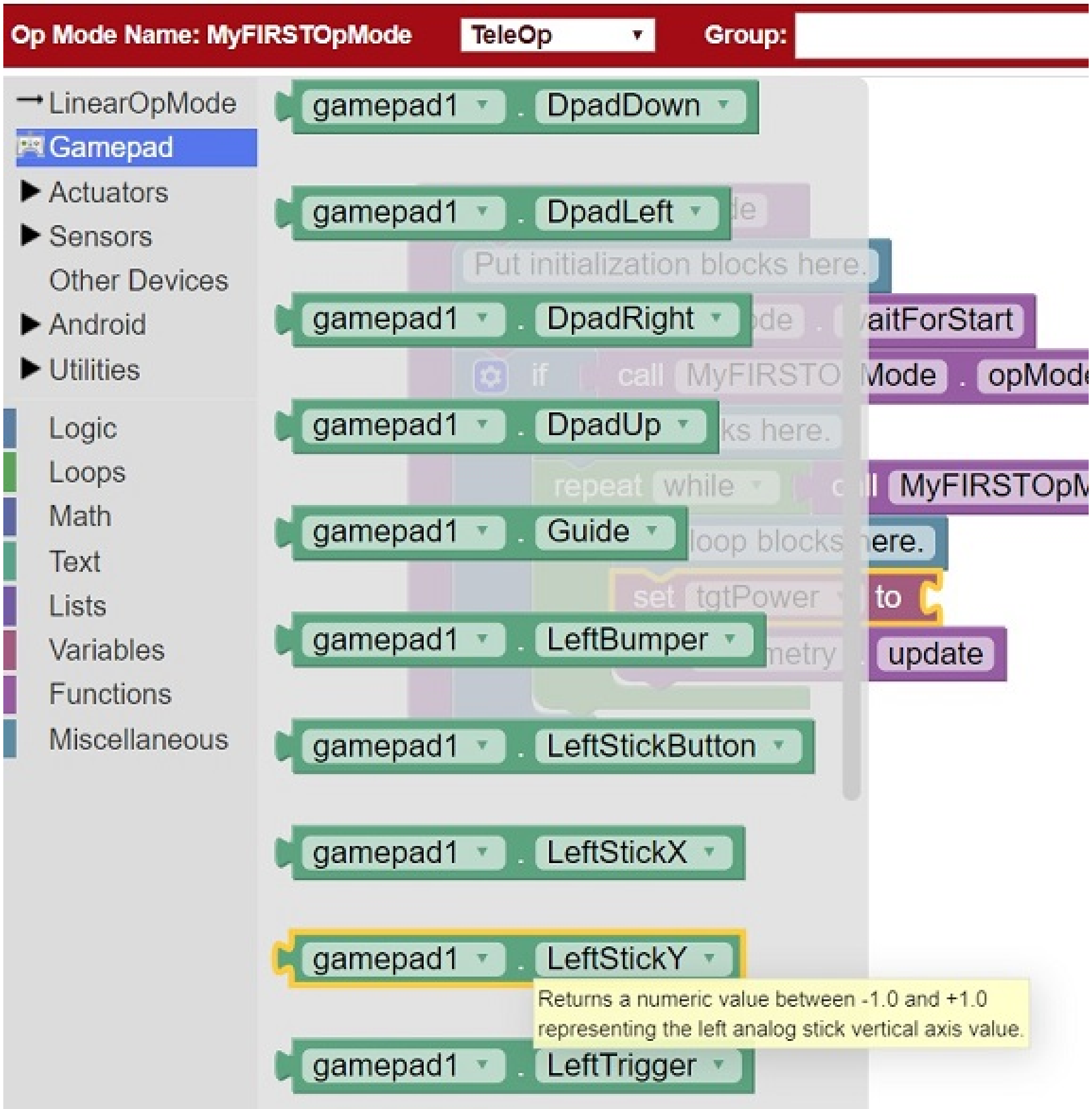


4. Click on the “set tgtPower to” programming block and then use the mouse to drag the block to the spot just after the “Put loop blocks here” comment block.



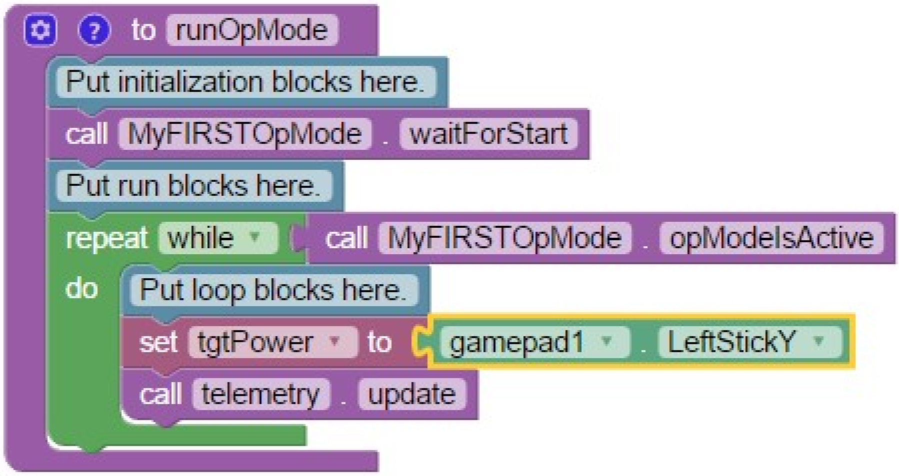
The “set tgtPower to” block should snap right into position.

5. Click on the “Gamepad” category of the programming blocks and select the “gamepad1.LeftStickY” block from the list of available blocks.

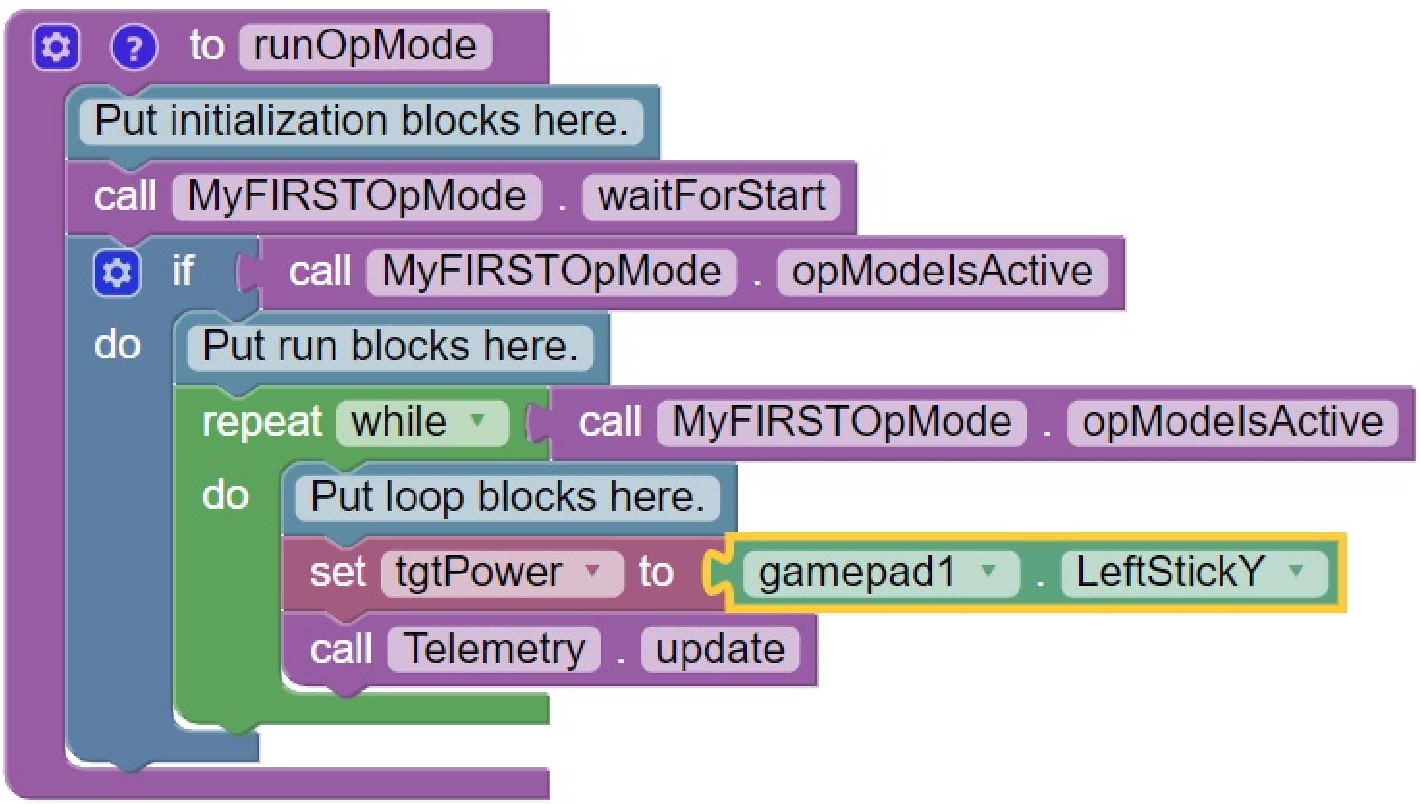


Note that the control system lets you have up to two gamepads controlling a robot. By selecting "gamepad1" you are telling the op mode to use the control input from the gamepad that is designated as driver #1.

6. Drag the "gamepad1.LeftStickY" block so it snaps in place onto the right side of the "set tgtPower to" block. This set of blocks will continually loop and read the value of gamepad #1's left joystick (the y position) and set the variable tgtPower to the Y value of the left joystick.

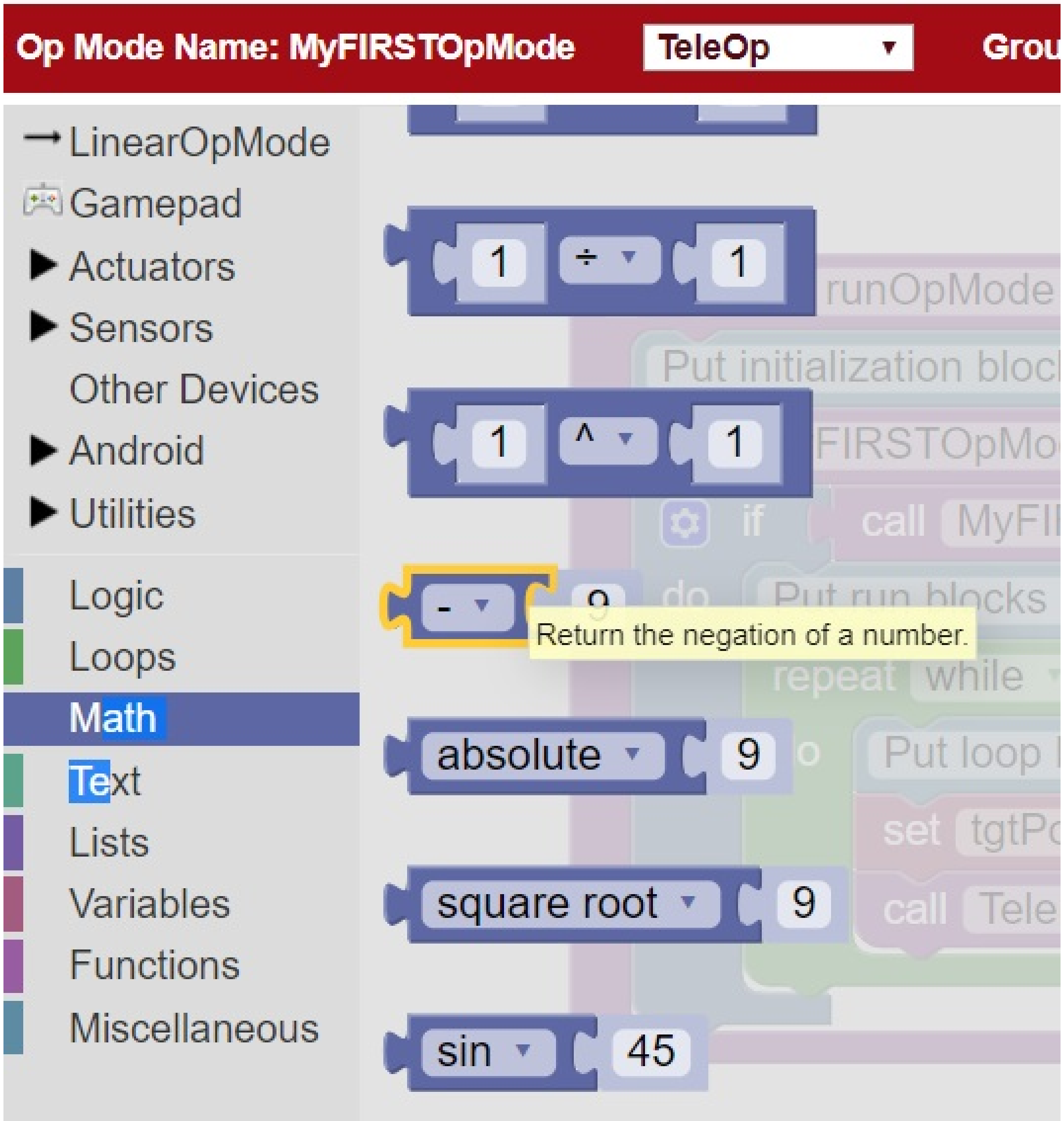


Note that for the F310 gamepads, the Y value of a joystick ranges from -1, when a joystick is in its topmost position, to +1, when a joystick is in its bottommost position.

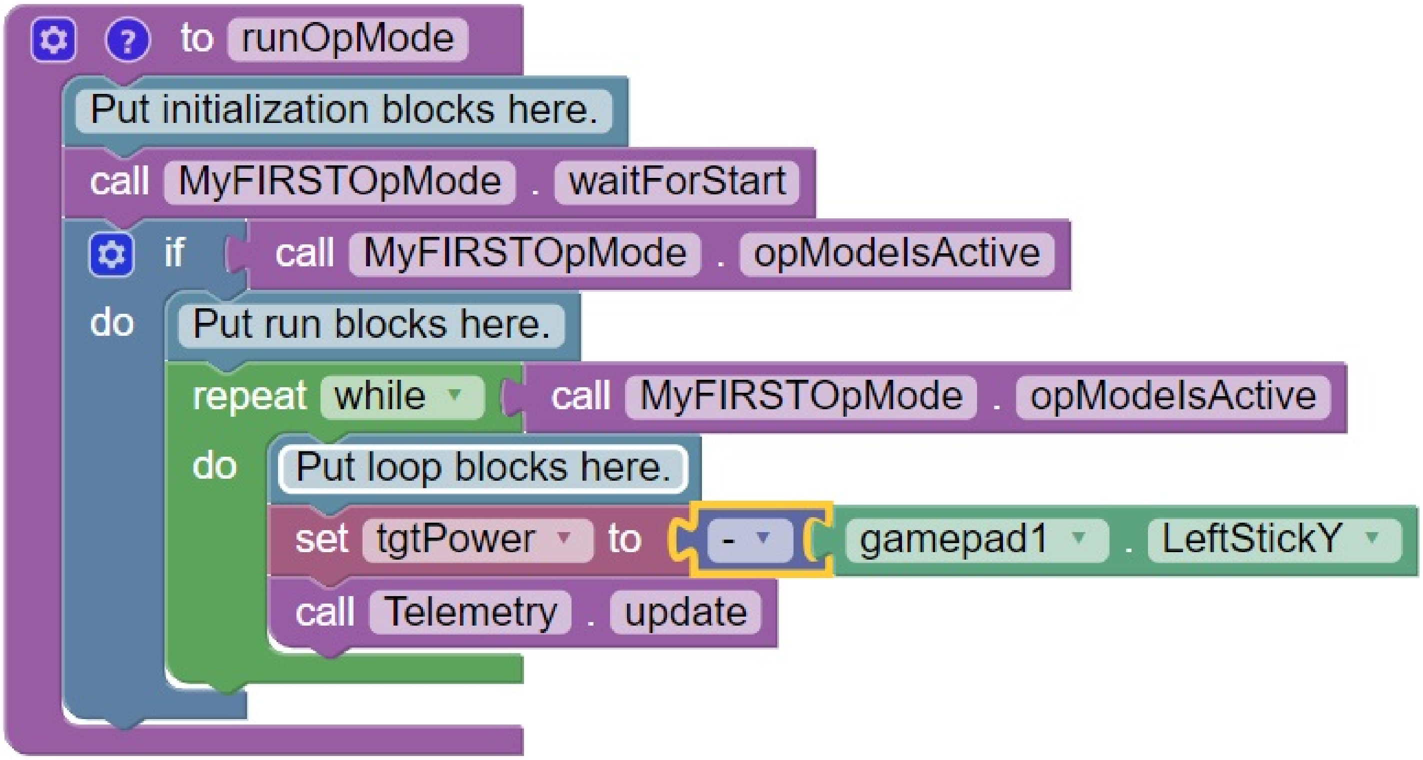


This means that for the blocks shown in our example, if the left joystick is pushed to the top, the variable tgtPower will have a value of -1.

7. Click on the “Math” category for the programming blocks and select the negative symbol (“-“).

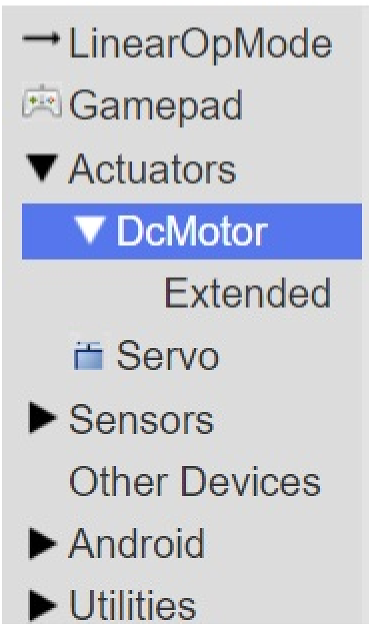


8. Drag the negative symbol (also known as a “negation operator”) to the left of the “gamepad1.LeftStickY” block. It should click in place after the “set tgtPower to” block and before the “gamepad1.LeftStickY” block.

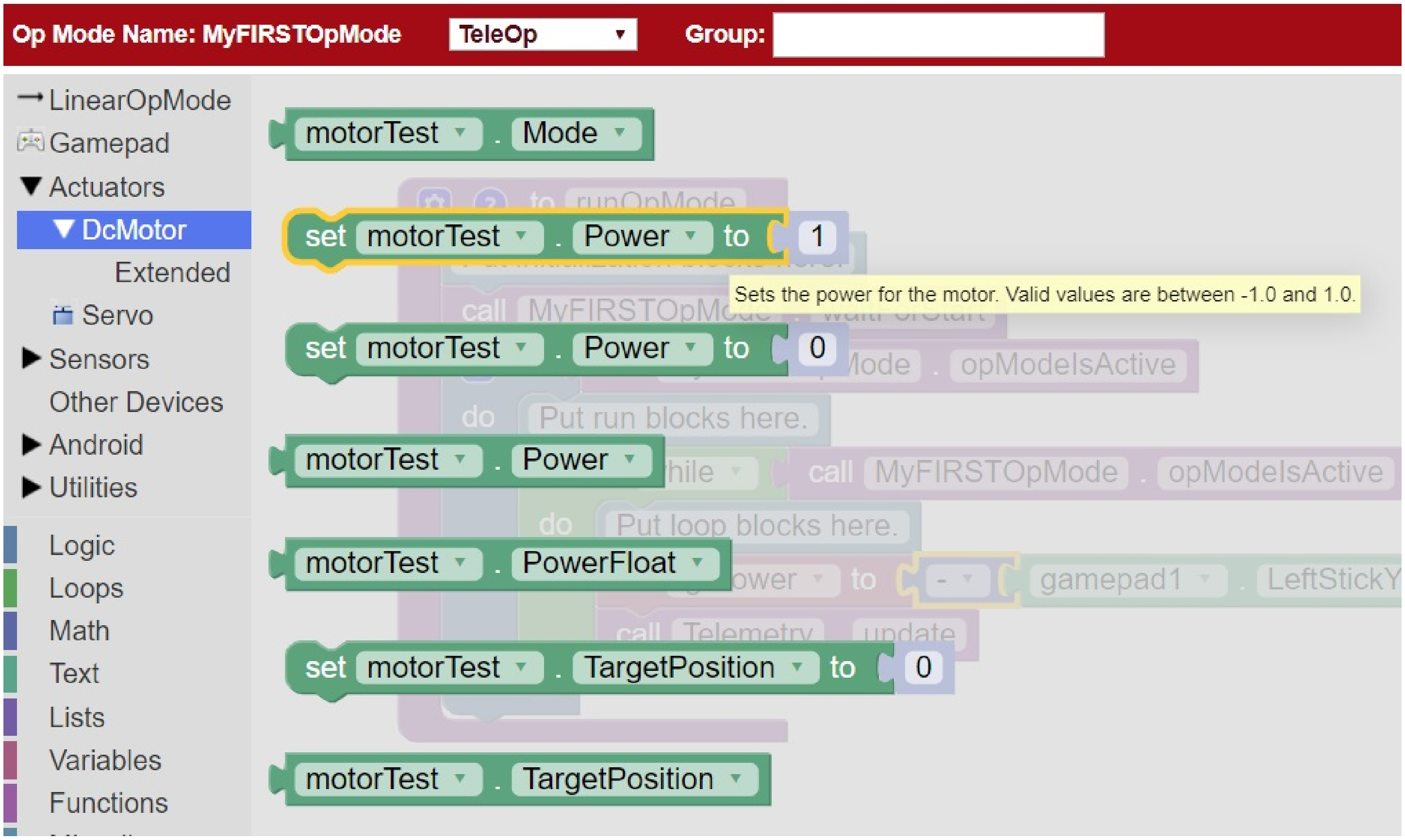


With this change, the variable tgtPower will be set to +1 if the left joystick is in its topmost position and will be set to -1 if the joystick is in its bottommost position.

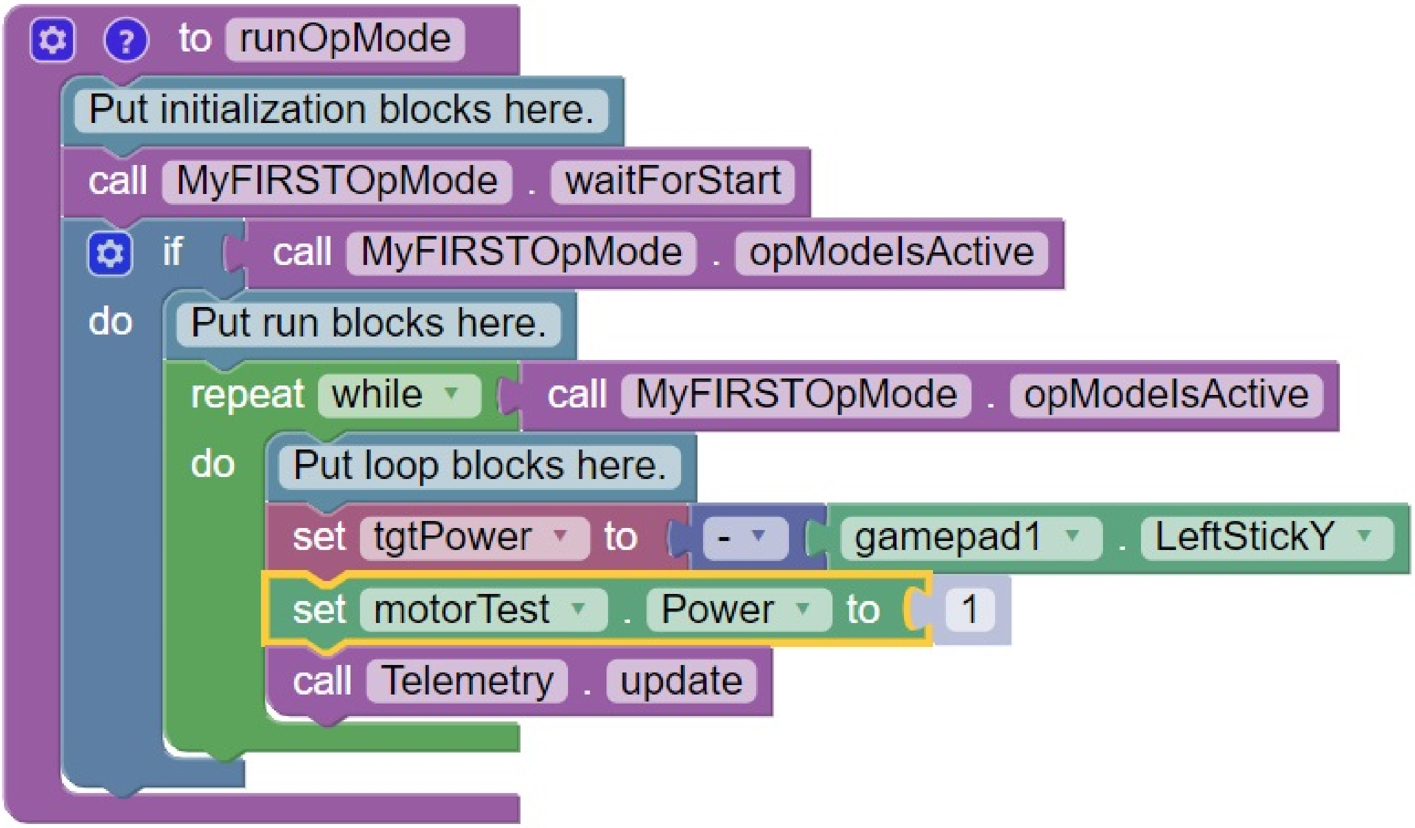
9. Click on the “Actuators” category of blocks. Then click on the “DcMotor” category of blocks.



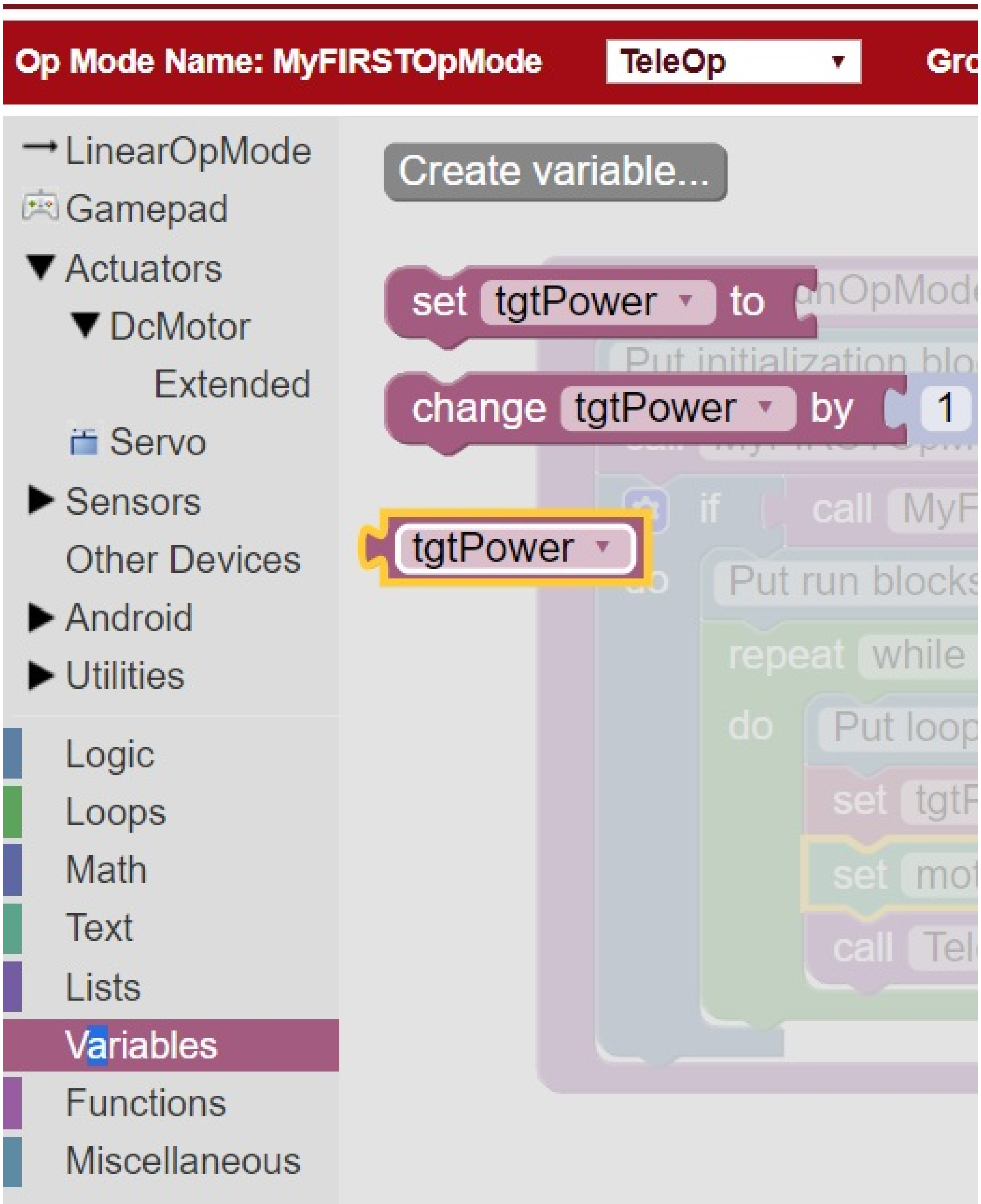
10. Select the “set motorTest.Power to 1” programming block.



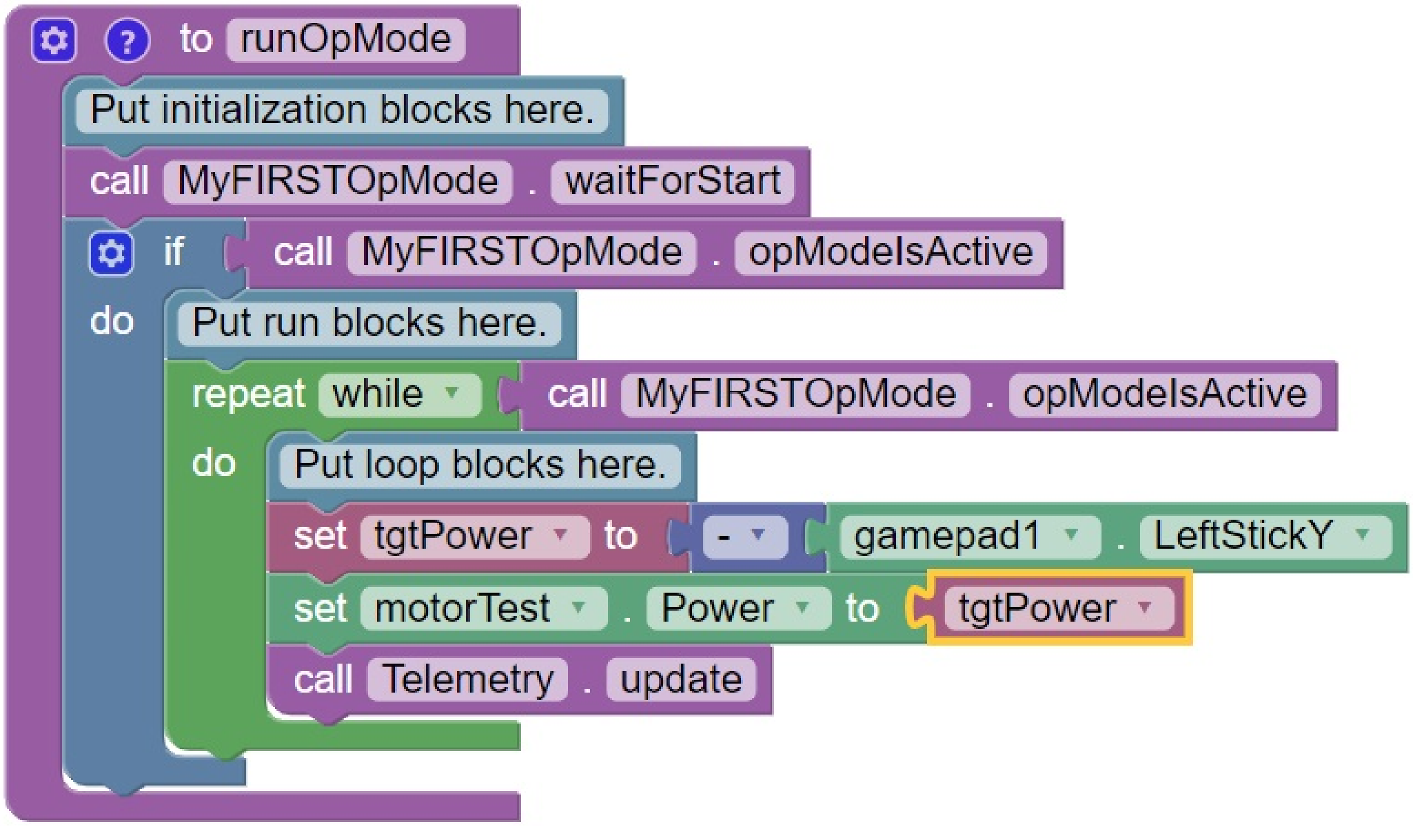
11. Drag and place the “set motorTest.Power to 1” block so that it snaps in place right below the “set tgtPower to” block.



12. Click on the “Variables” block category and select the “tgtPower” block.



13. Drag the “tgtPower” block so it snaps in place just to the right of the “set motor1.Power to” block.



The “tgtPower” block should automatically replace the default value of “1” block.

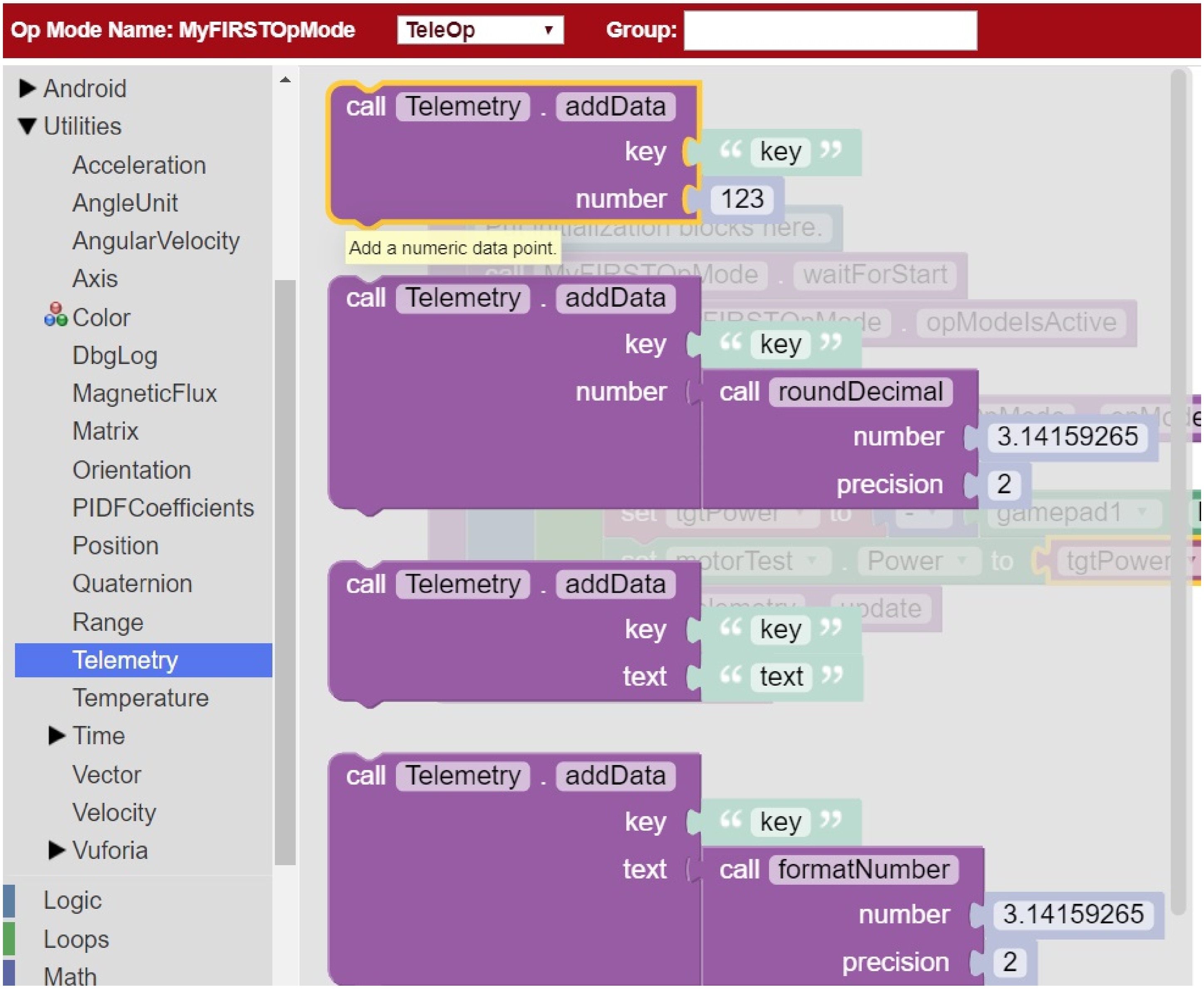
Inserting Telemetry Statements

Your op mode is just about ready to run. However, before continuing, you will add a couple of telemetry statements that will send information from the Robot Controller to the DRIVER STATION for display on the DRIVER STATION user interface. This telemetry mechanism is a useful way to display status information from the robot on the DRIVER STATION. You can use this mechanism to display sensor data, motor status, gamepad state, etc. from the Robot Controller to the DRIVER STATION.

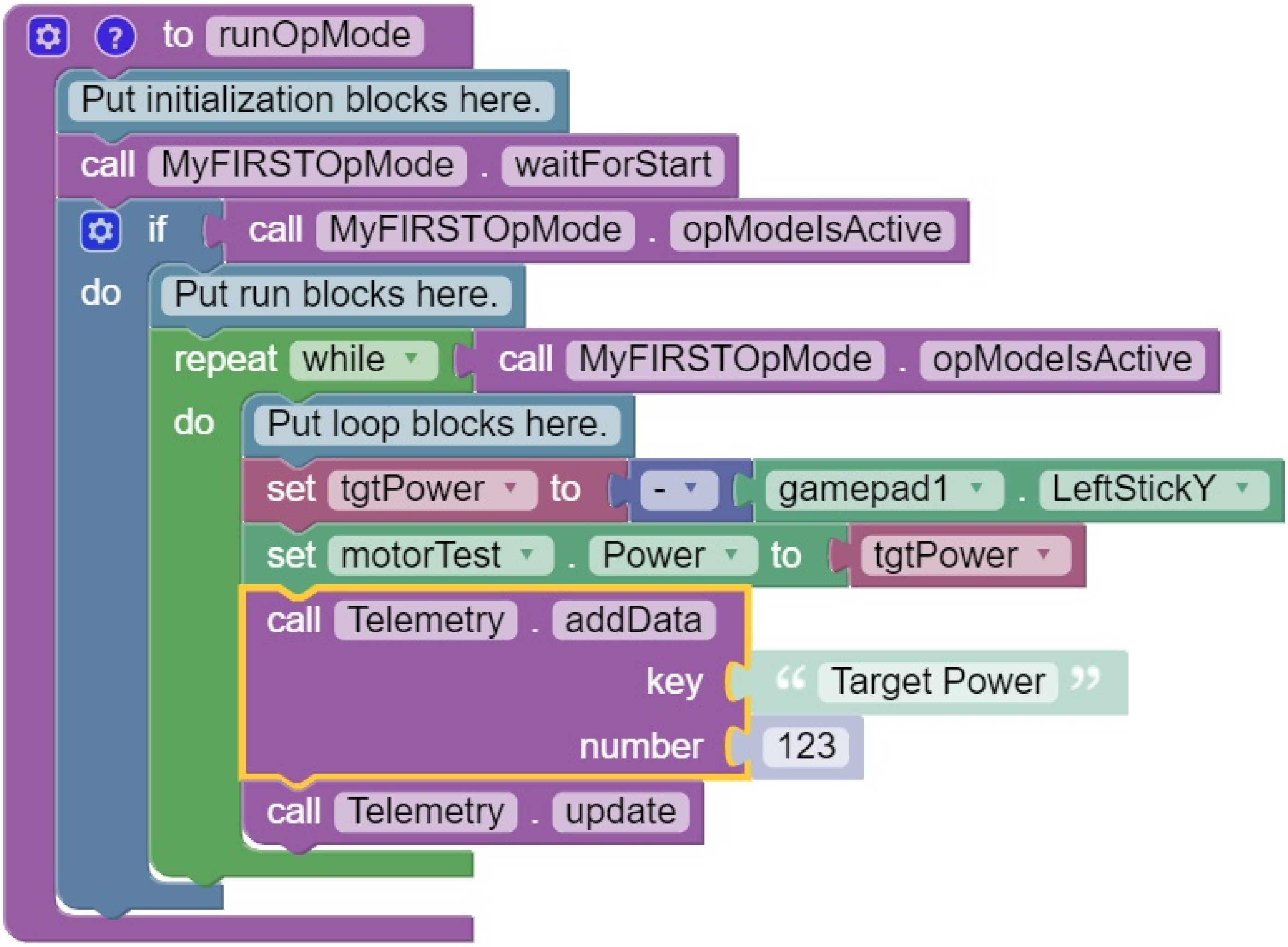
Note that you will need an estimated 15 minutes to complete this task.

Inserting Telemetry Statements Instructions

1. Click on the “Utilities” category on the left-hand side of the browser window. Select the “Telemetry” subcategory and select the “call telemetry.addData(key, number)” block.

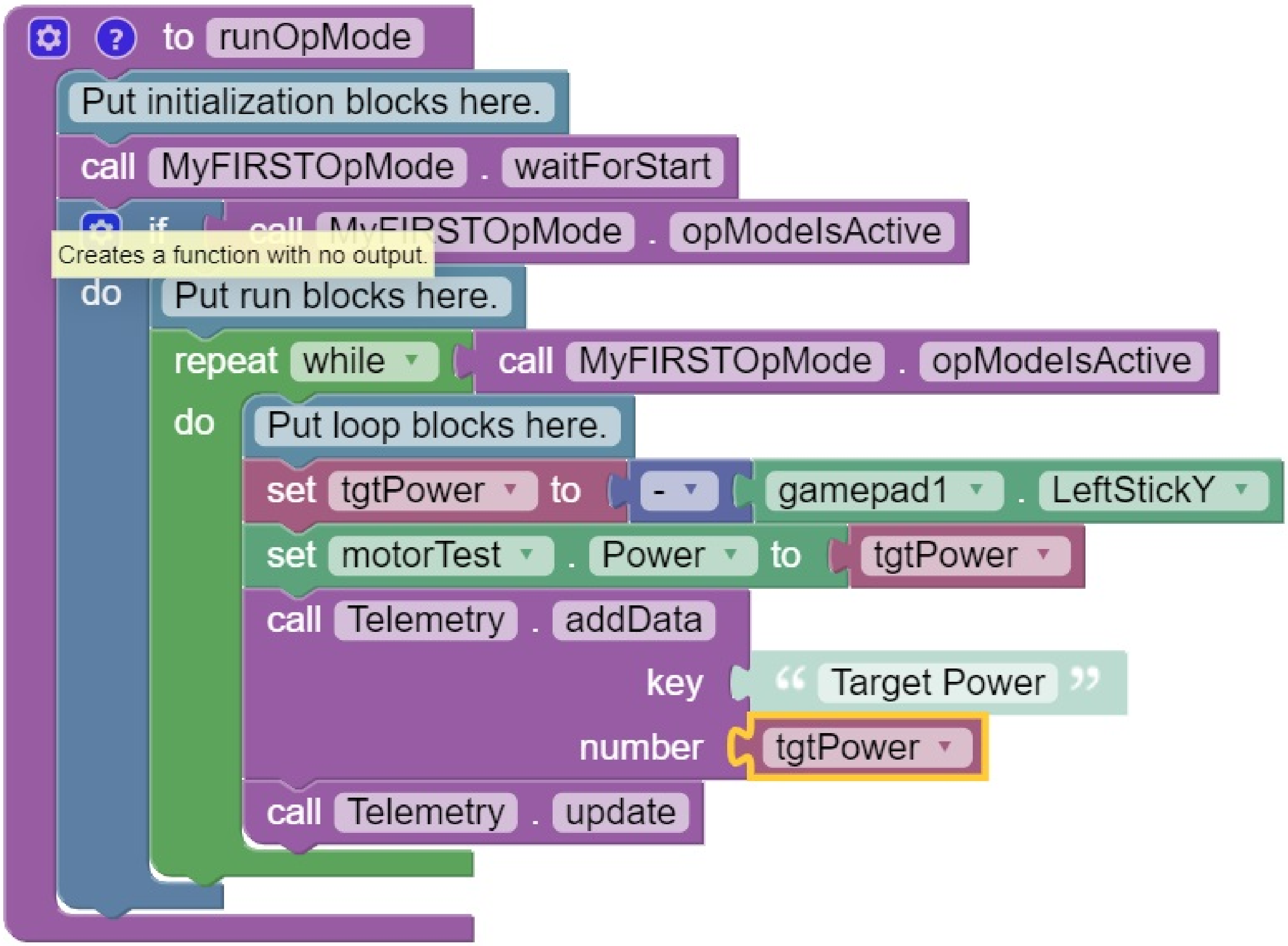


2. Drag the “call telemetry.addData(key, number)” block and place it below the “set motor1.Power to” block. Click on the green text block “key” and highlight the text and change it to read “Target Power”.



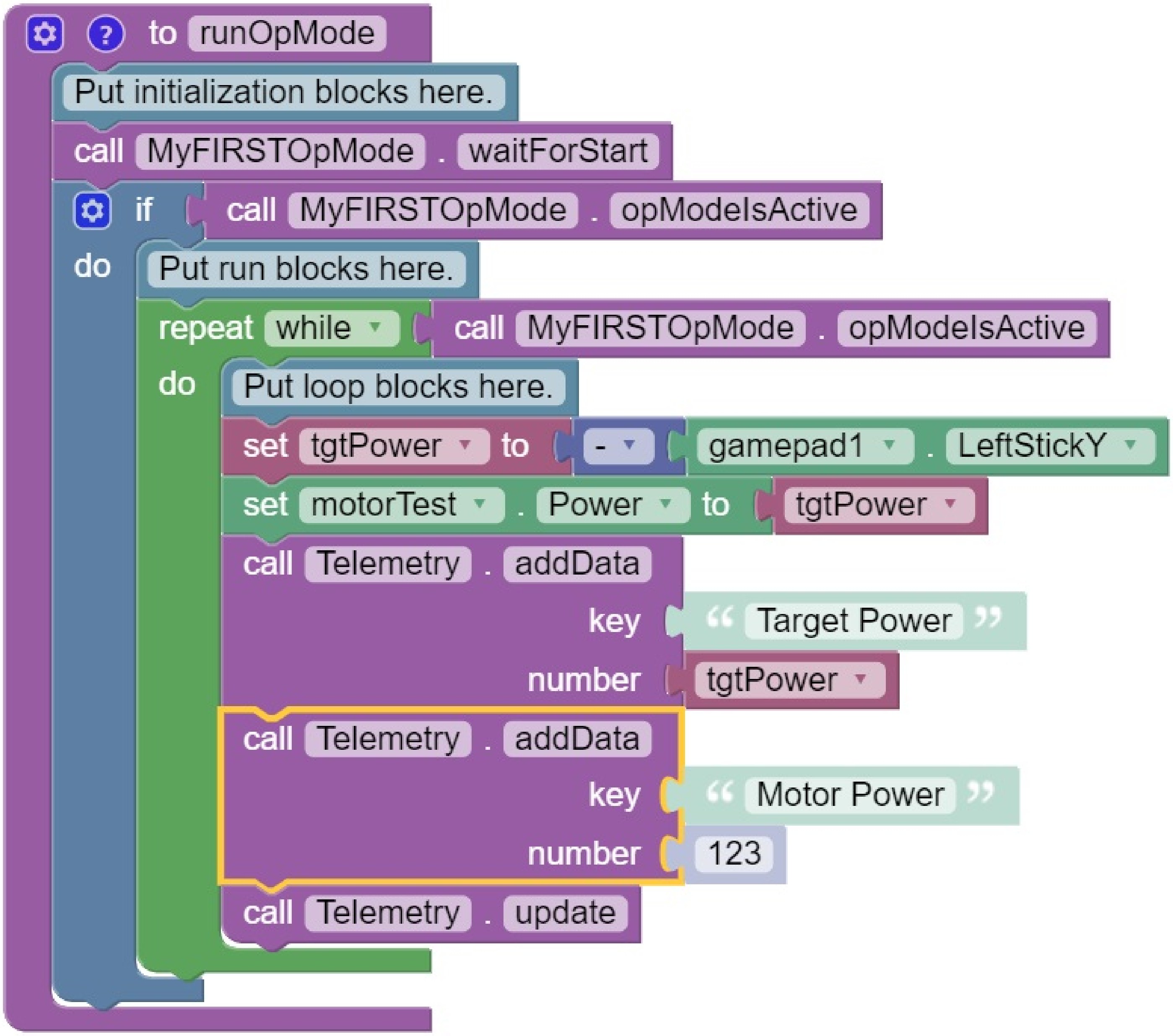
Note that the “call telemetry.update” block is an important block. Data that is added to the telemetry buffer will not be sent to the DRIVER STATION until the “telemetry.update” method is called.

3. Click on the “Variables” block category and select the “tgtPower” block. Drag the block so it clicks into place next to the “number” parameter on the telemetry programming block.

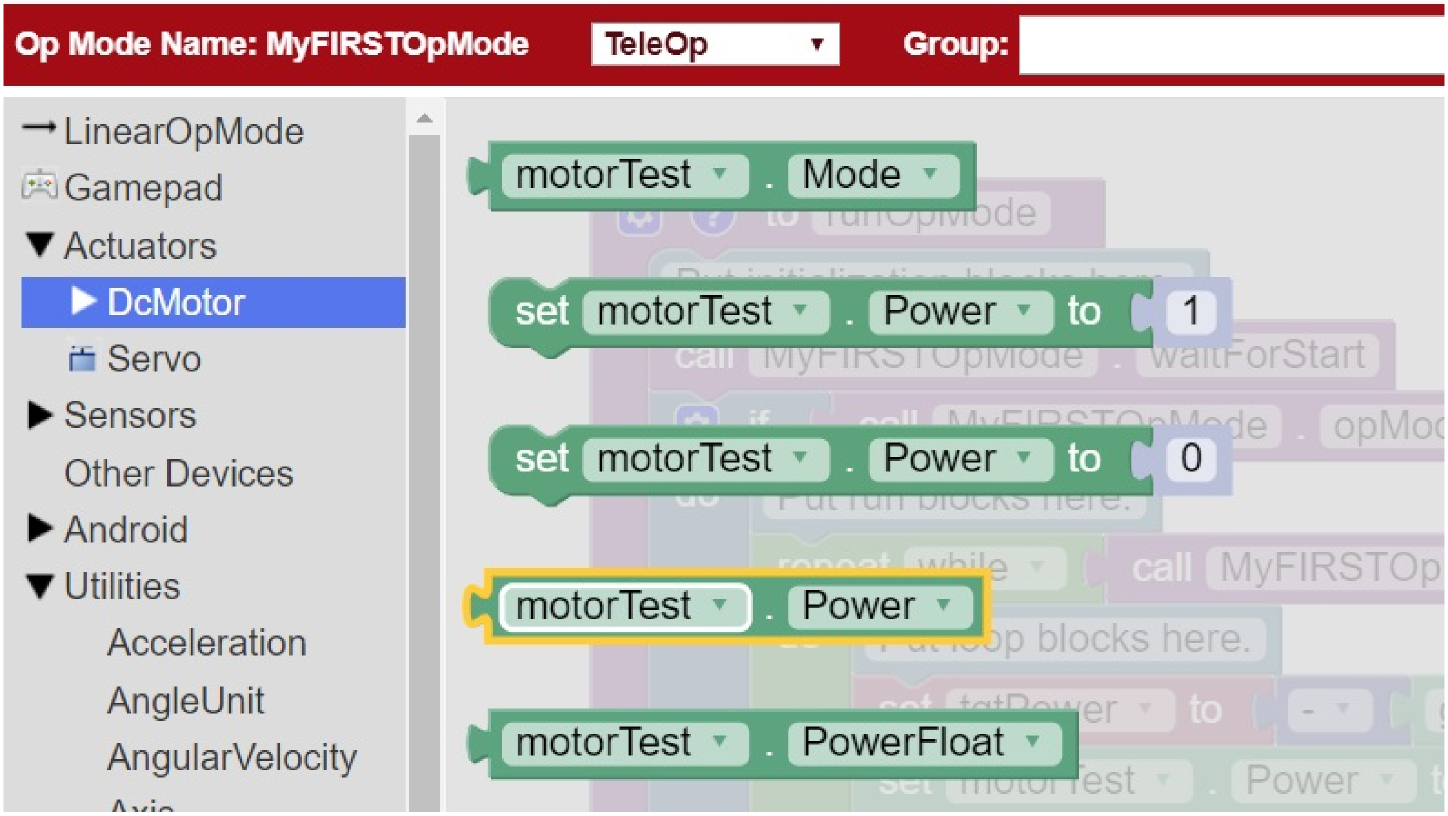


The Robot Controller will send the value of the variable tgtPower to the DRIVER STATION with a key or label of "Target Power". The key will be displayed to the left of the value on the DRIVER STATION.

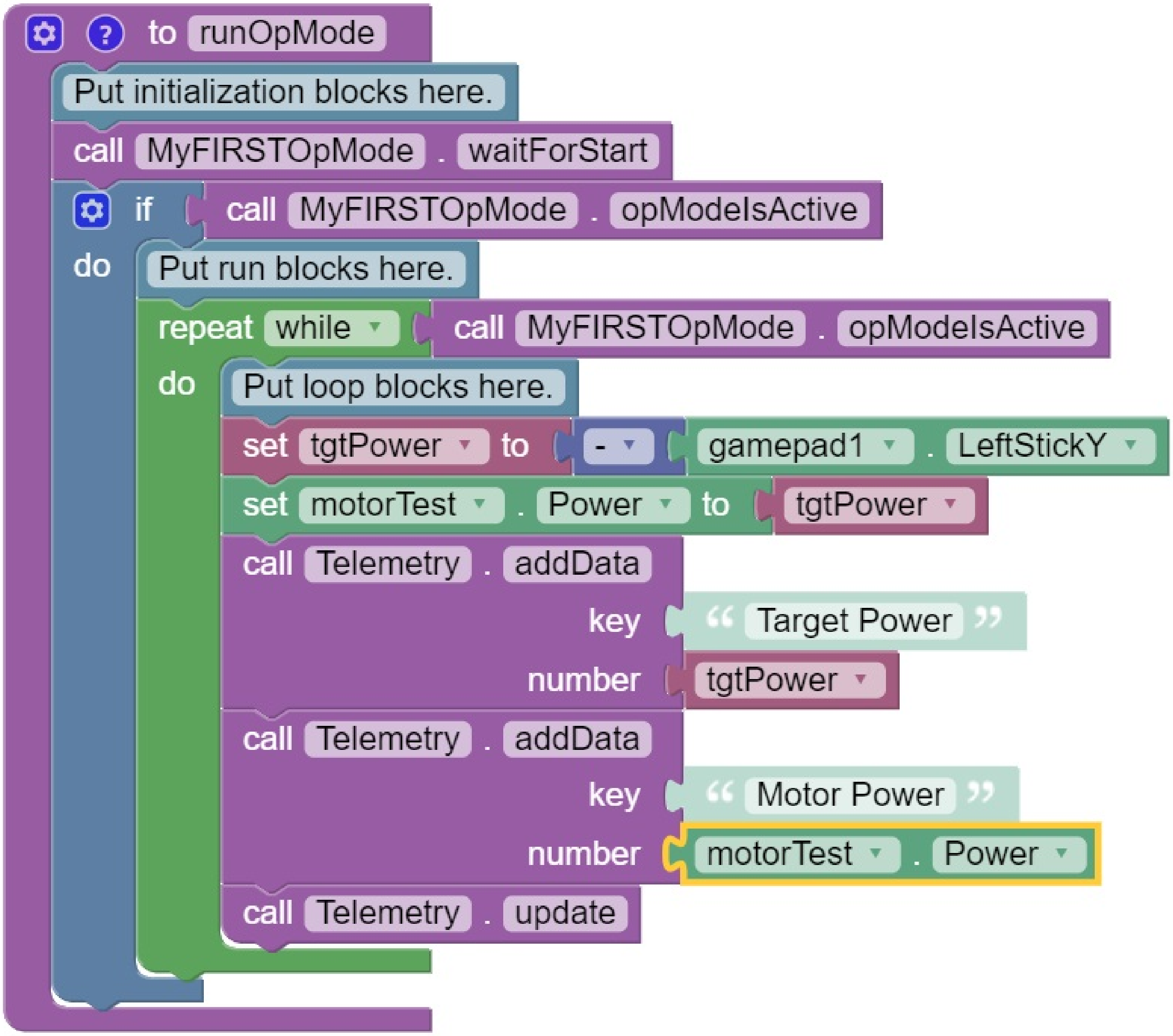
4. Repeat this process and name the new key "Motor Power".



5. Find and click on the “DcMotor” subcategory. Look for the green programming block labeled “motorTest.Power”.



6. Drag the "motorTest.Power" block to the "number" parameter of the second telemetry block.



Your op mode will now also send the motor power information from the Robot Controller to be displayed on the DRIVER STATION.

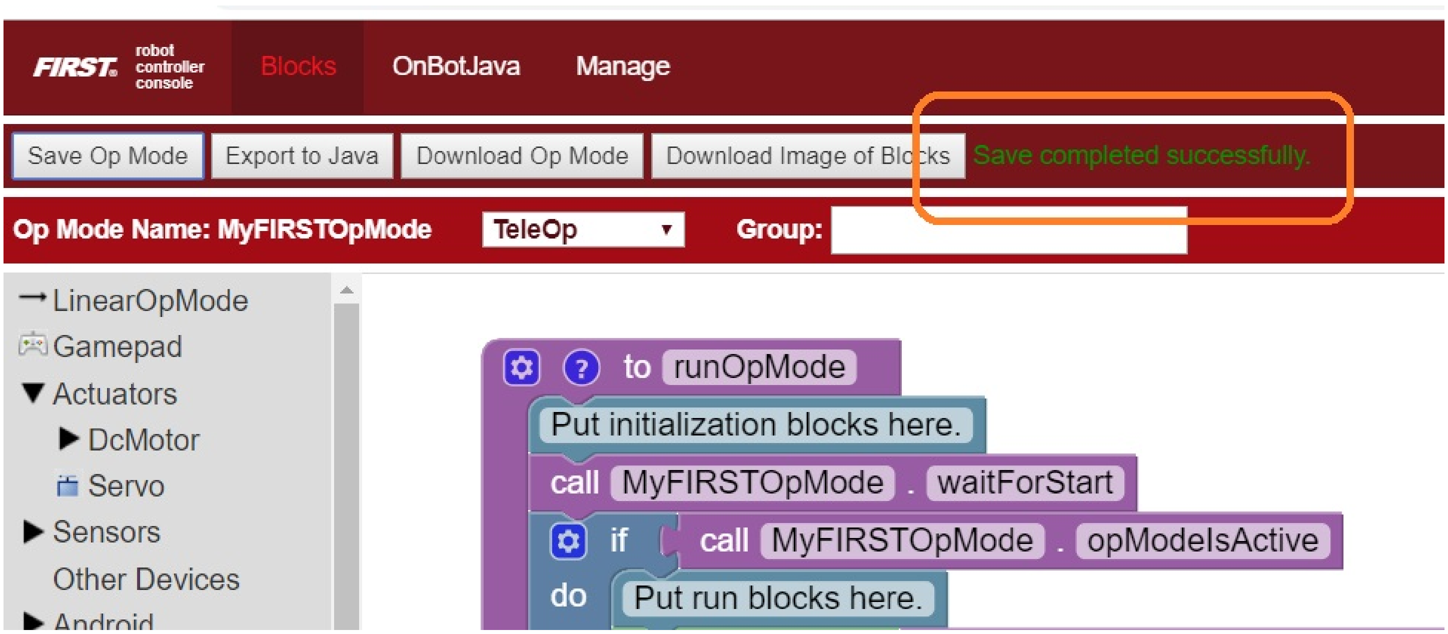
Saving Your Op Mode

After you have modified your op mode, it is very important to save the op mode to the Robot Controller.

Note it will take an estimated 1 minute to complete this task.

Saving Your Op Mode Instructions

1. Press the "Save Op Mode" button to save the op mode to the Robot Controller. If your save was successful, you should see the words "Save completed successfully" to the right of the buttons.



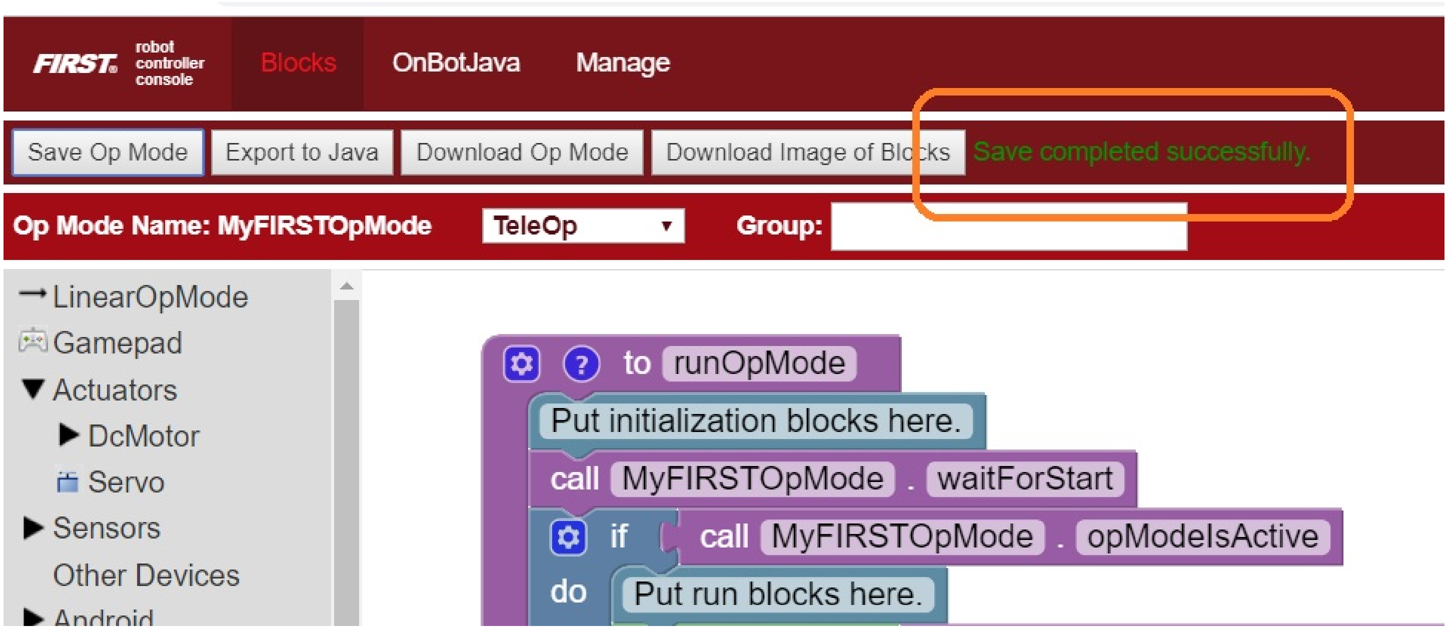
Exiting Program & Manage Screen

After you have modified and saved your op mode, if your DRIVER STATION is still in the Program & Manage screen, then you should exit this screen and return to the Main DRIVER STATION screen.

Note it will take an estimated 1 minute to complete this task.

Exiting Programming Mode Instructions

1. Press the Android back arrow to exit the Program & Manage screen. You need to exit the Program & Manage screen before you can run your op mode.



Congratulations! You wrote your first op mode using the Blocks Programming Tool! You will learn how to run your op mode in the the section entitled *Running Your Op Mode*.

Running Your OpMode (All Languages)

If your OpMode requires input from a gamepad, then you will need to connect a Logitech F310 or other approved gamepad to the DRIVER STATION. Note that you can have up to two gamepads connected to a DRIVER STATION. If using a phone, this will require a USB hub. However, in this example, we will only have a solitary gamepad connected.

Note that you will need an estimated 10 minutes to complete this task.

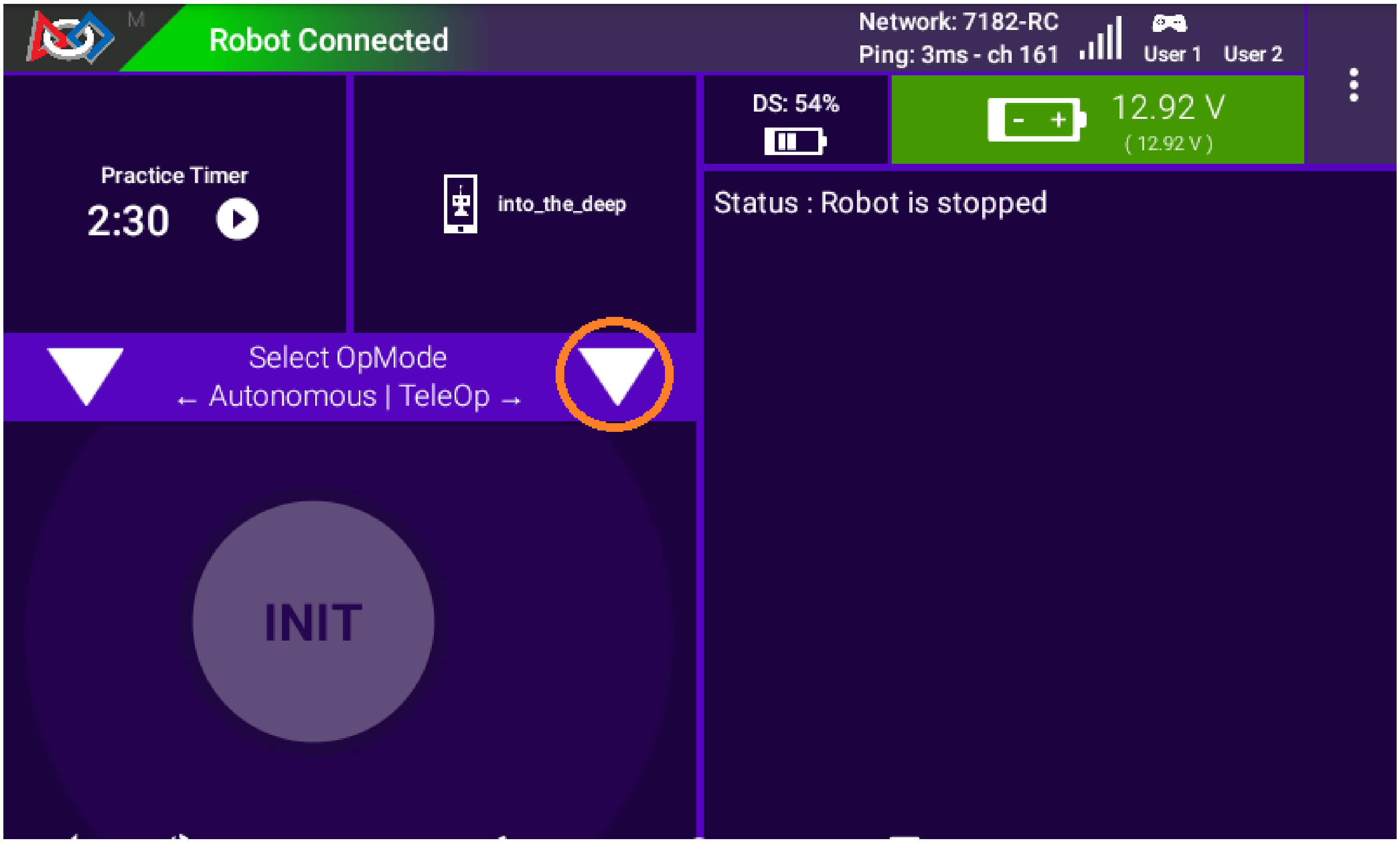
Running Your OpMode Instructions

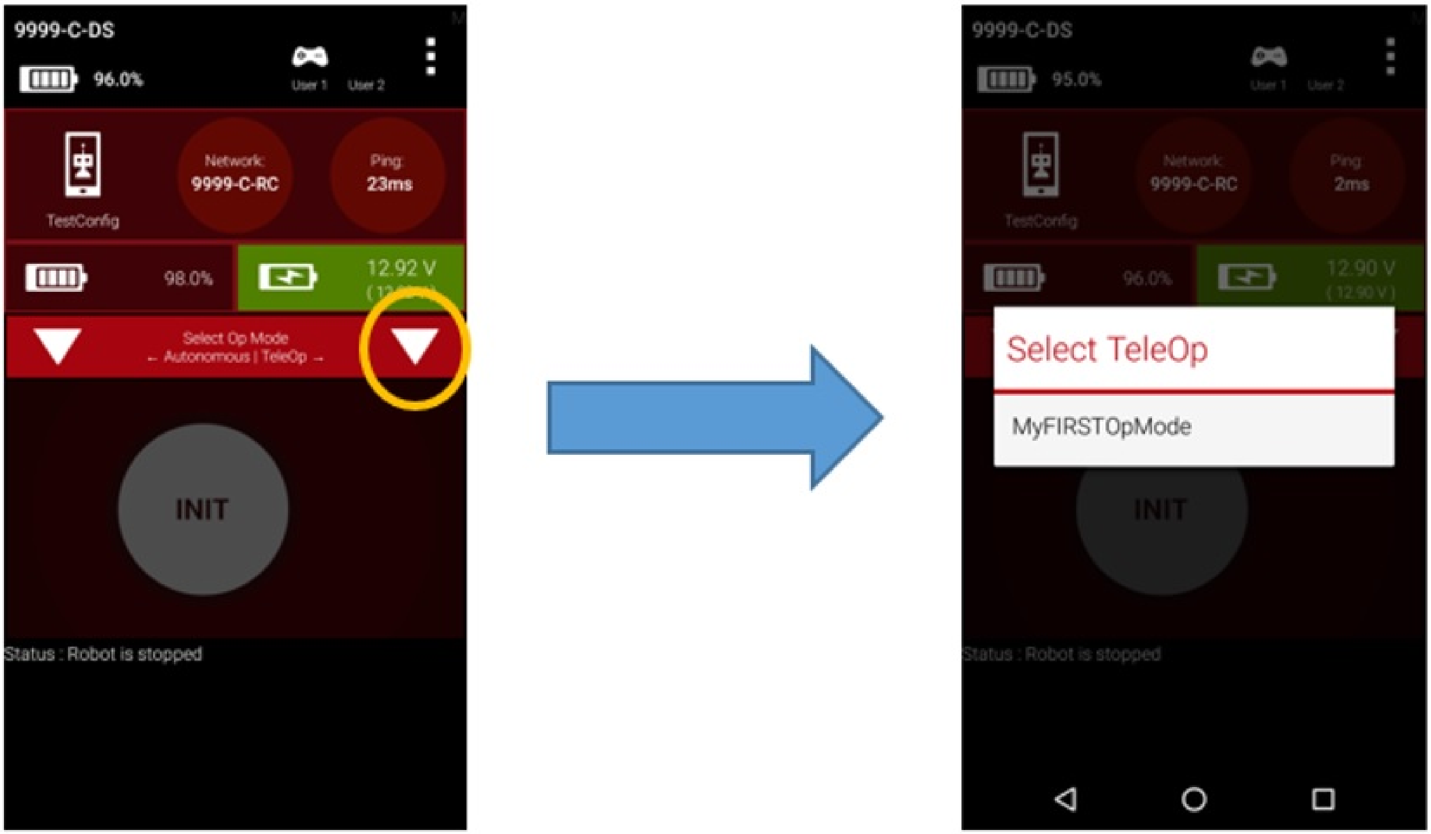
1. Connect the gamepad to the DRIVER STATION. If using a phone, you will need a Micro USB OTG adapter cable.

2. For the examples in this wiki, the OpModes are looking for input from the gamepad designated as the user or driver #1. Press the Start button and the A button simultaneously on the Logitech F310 controller to designate your gamepad as user #1. If using a PS4-style gamepad, use the Options and Cross buttons to designate your gamepad as user #1.

Note that pushing the Start button and the B button simultaneously would designate the gamepad as user #2.

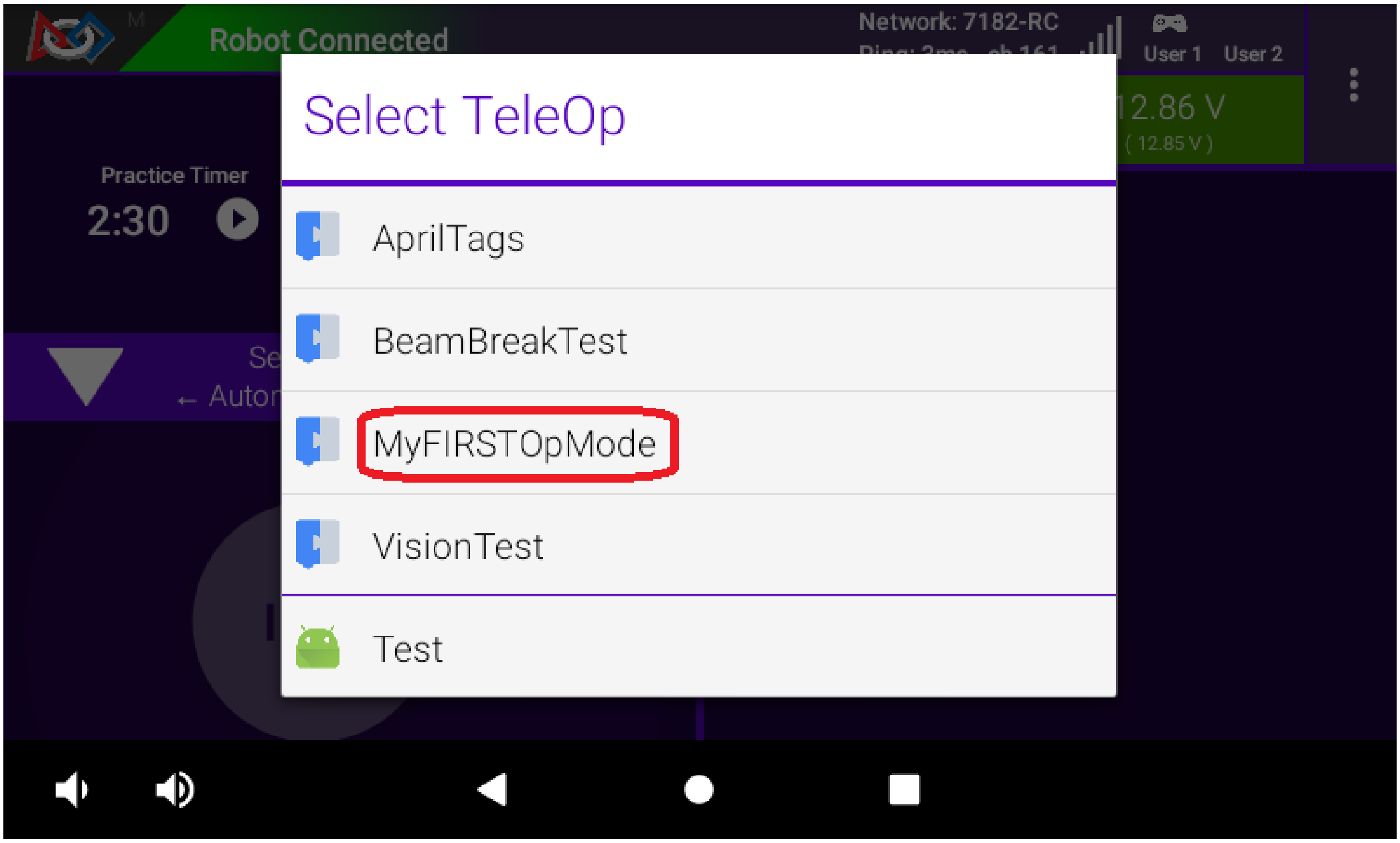
3. On the DRIVER STATION screen, touch the triangular-shaped, "TeleOp" dropdown list button to display a list available OpModes. You should see your recently saved OpMode among the list of available OpModes that reside on your Robot Controller.

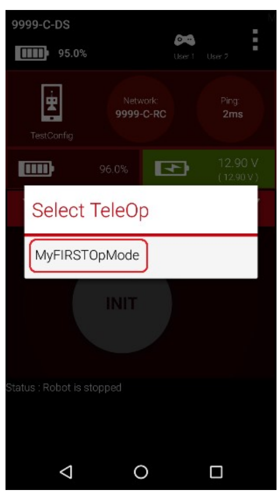




Note that the word “TeleOp” is short for “Tele-Operated” and it implies a driver controlled OpMode (i.e. an OpMode that gets input from a human driver).

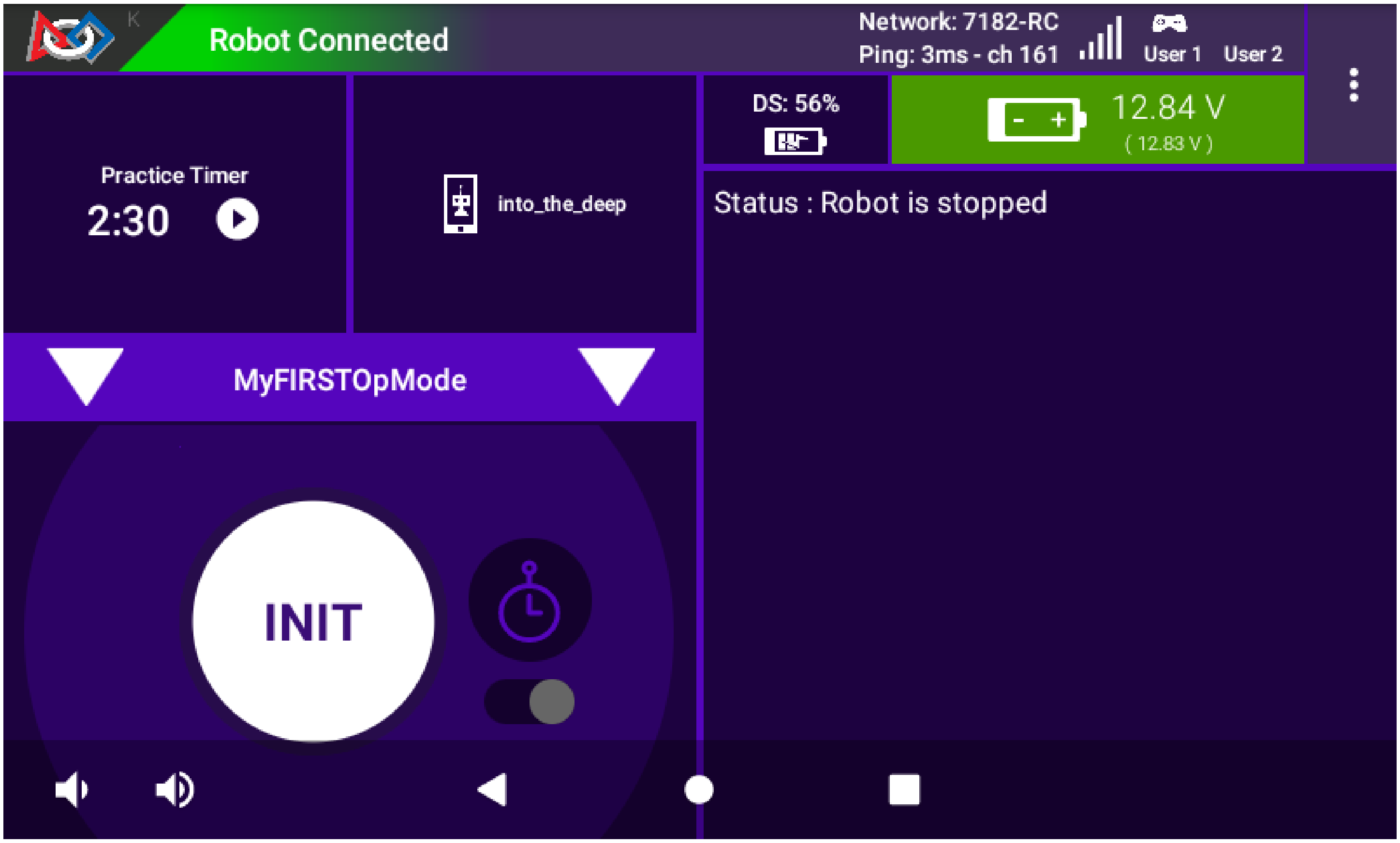
4. Select “MyFIRSTOpMode” to load your OpMode on the Robot Controller.



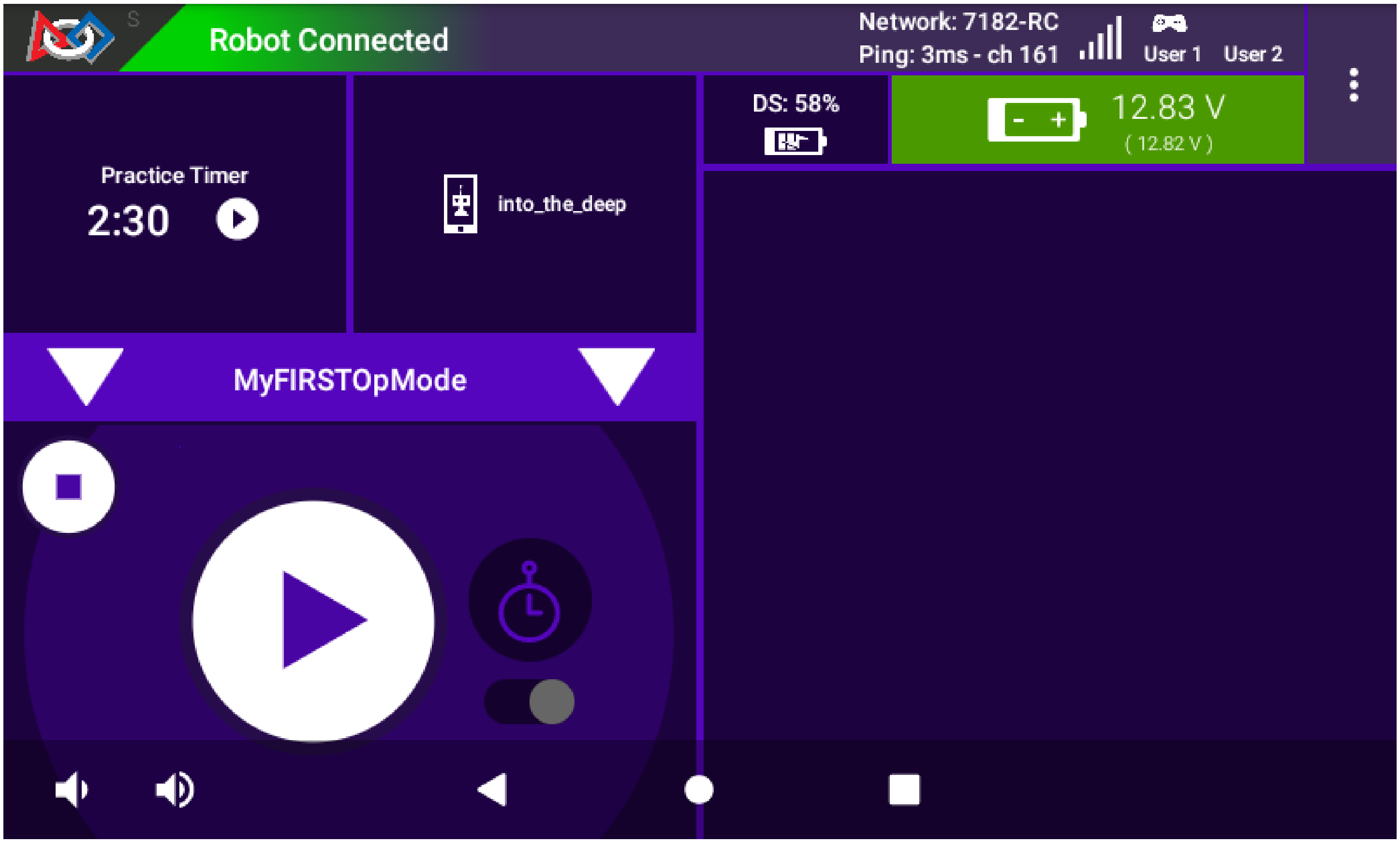


Note that even though you are using the DRIVER STATION to select the OpMode, the actual OpMode instructions will be executed on the Robot Controller.

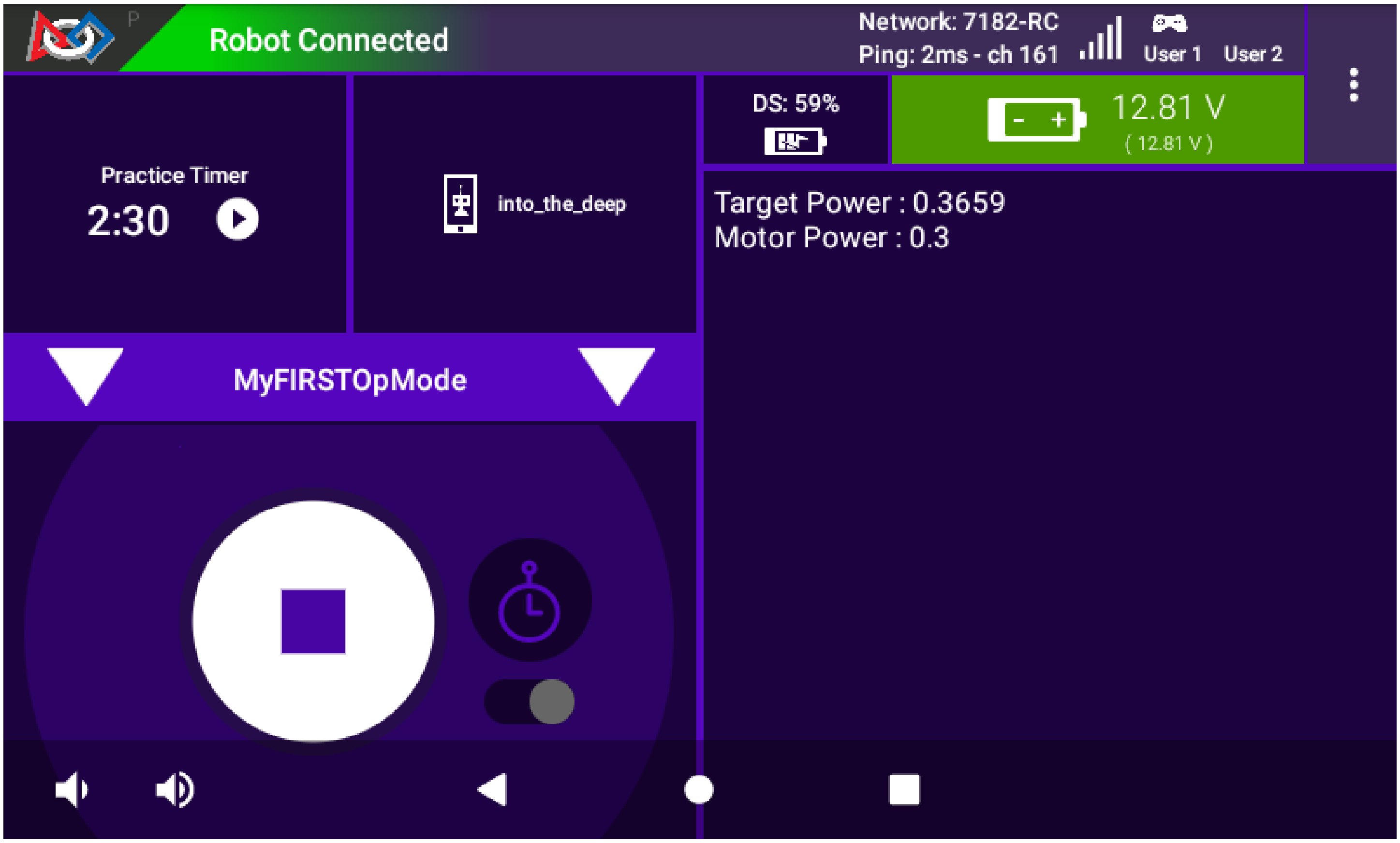
1. Press the INIT button to initialize your OpMode.



6. Push the Start button (designated by the triangular-shaped symbol) to start the OpMode run.



7. Use the left joystick of the gamepad to control the operation of the DC motor. As you manipulate the left joystick up and down, the target power and the motor power should be displayed in the upper right hand corner of the screen.



If you want to stop your OpMode, press the square-shaped Stop button on the DRIVER STATION.

Managing OpModes in Blocks Blocks

Blocks is a programming language that uses graphical programming elements to create programs. As such its file format is different than, say, a JAVA or other text-based programming language file. Blocks programs are saved with a **.blk** extension, but its contents are actually formatted as XML (Extensible Markup Language). The actual XML format in a Blocks program is beyond the scope of this document, except to say that it's not intended to be read/viewed/interpreted by any other program than Blocks. There is not a general program on a MAC or a PC that can view or edit the Blocks program, it must always be done through the Blocks interface within the Robot Controller App (running on a REV Control Hub or legal Android SmartPhone) - that is, to say, you cannot simply double-click on the file to open it up in an editor program that lives on your computer.

Creating an OpMode

There is a *great tutorial for creating OpModes* that also explains a lot about the Blocks interface and helps you to understand what a Blocks program does. It is recommended to check out this document for learning how to work with Blocks OpModes.

Saving an OpMode

It's important to understand what is meant by **"Saving"** an OpMode. When programming/editing an OpMode, you're using either a web browser (Chrome, etc.) or you are using a program *acting* as a web browser (REV Hardware Client, etc.). The program that you are creating/editing only *ephemerally* exists within the web browser; there is no auto-save or feature to ensure that the program is ultimately saved back onto the device (REV Control Hub or approved SmartPhone) for use by a robot. Only the *SAVE* operation will actually save the OpMode to a **.blk** file onto the device. Therefore, it's imperative that Blocks programmers *SAVE* their work often, and especially once they have completed their work. The mechanism by which you can *SAVE* an OpMode is via the "**Save Op Mode**" button within the editing window of the software.

Once a program is saved, a message will appear on the right-hand side of the same row to indicate that the program has been saved.

Downloading an OpMode

Once an OpMode has been saved to a device, the OpMode can be selected via the DRIVER STATION or edited again via the programming interfaces. However, that Blocks program only exists as a Blocks File (**.blk**) on the device. Often it is desirable to save a copy of the program on your laptop (or on another device, or in some other safe location) or provide the program for use by others (teammates, another robot, other teams, provide online, etc.).

In order to get a copy of the Blocks program from the device, you need to *download* the program from the device. You can do this in one of two ways, either through the editing interface or the main Blocks management interface.

Downloading an OpMode through the Editing Interface

While editing an OpMode, an OpMode can be *saved* and it can also be *downloaded* (there are other options, but we're just going to focus on these two for the time being). When an OpMode is saved, the program is saved **onto the device** into a Blocks file (**.blk**). In order to save a copy of the program to your local computer (for safe storage or for sharing) you need to *download* the program. Downloading the program *does* issue a Save action on the current program, but this should not be relied upon - programmers should always save their program before downloading. Downloading an OpMode is performed via the "**Download Op Mode**" button within the Editing Interface.

Pressing the "Download Op Mode" button makes the file available to the web browser, so the web browser will manage the file in its usual way (e.g. with Chrome the file is saved into the computer's "Downloads" folder).

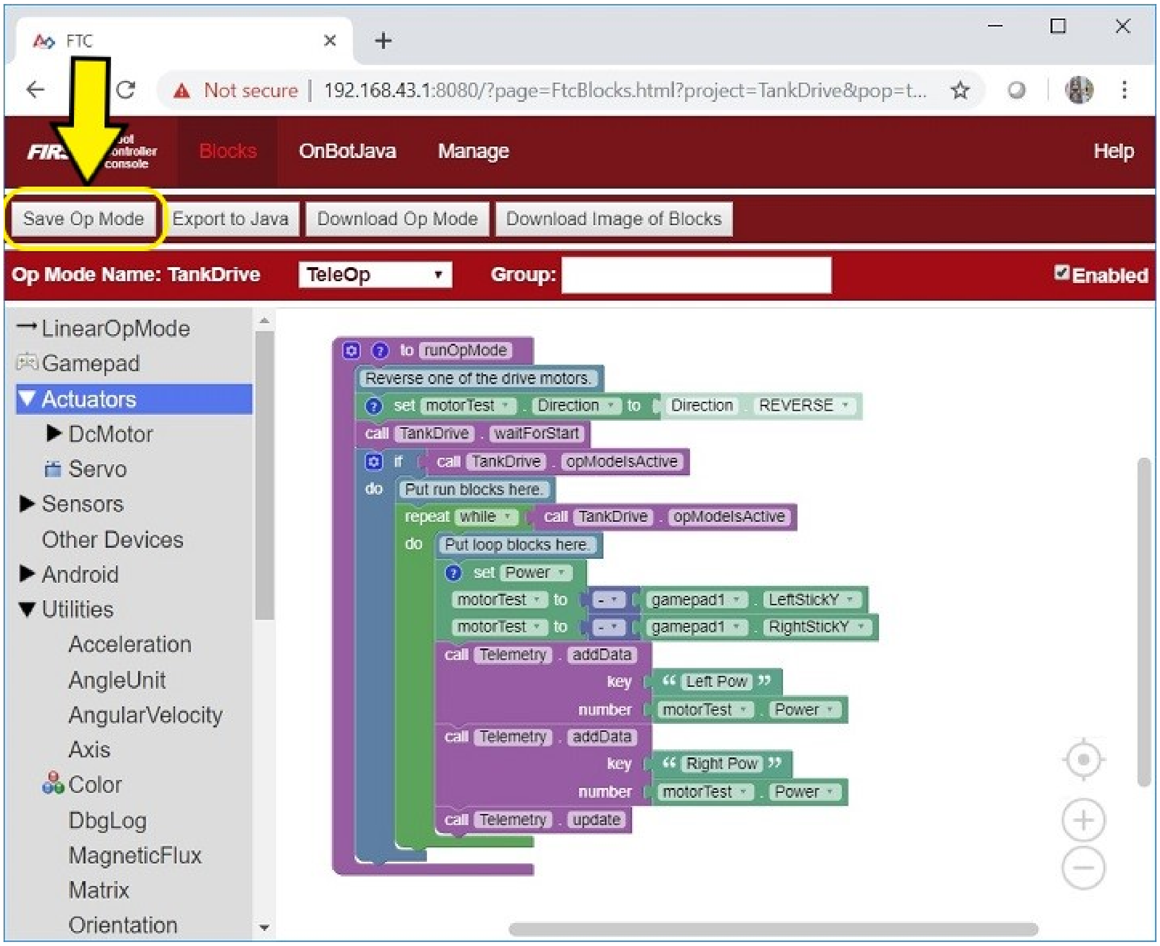


Fig. 1: Saving the OpMode within the Blocks Editor

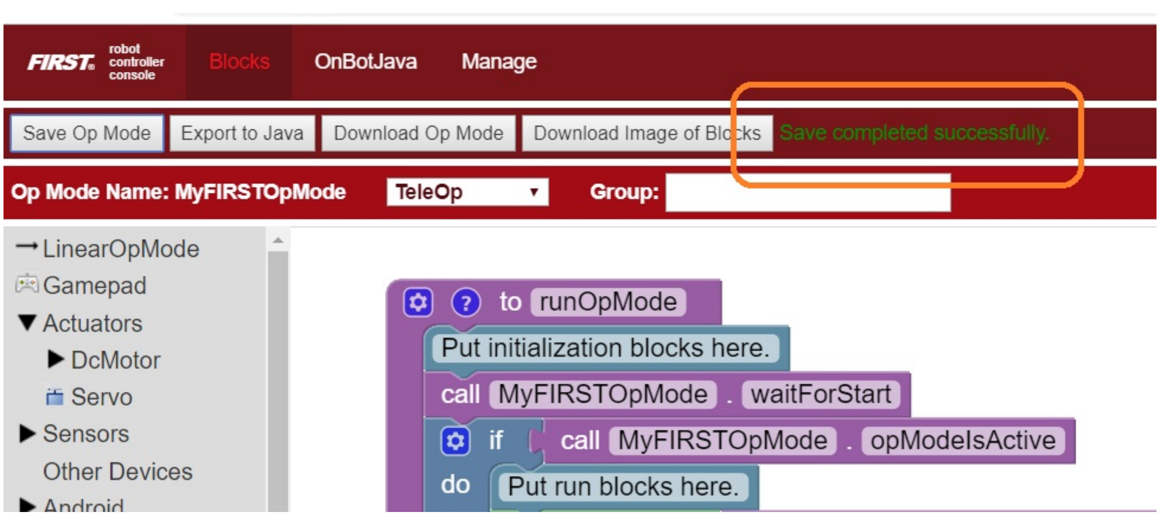


Fig. 2: Message indicating OpMode has been Saved

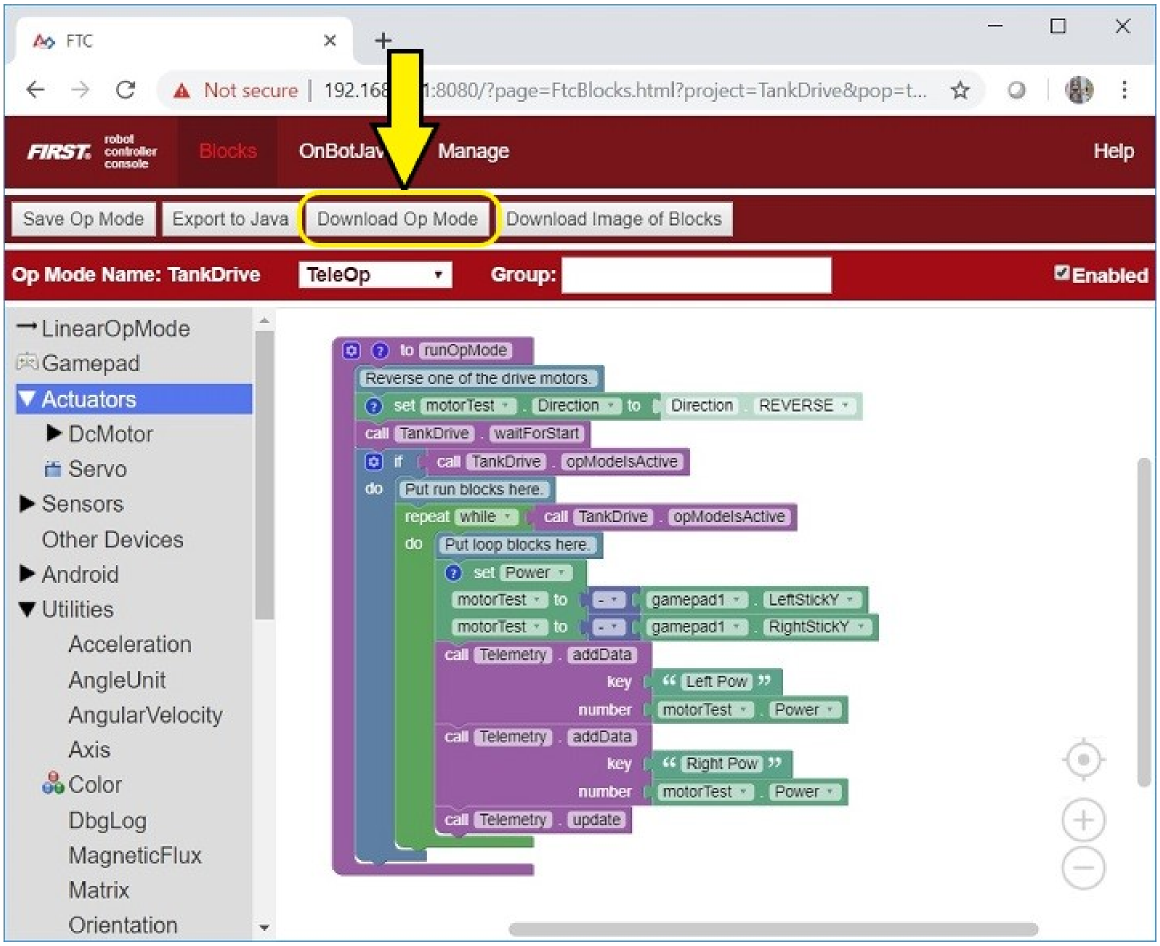


Fig. 3: Downloading a Blocks program

Downloading an OpMode through the Management Interface

By clicking on the "Blocks" menu item, you will be taken to the Blocks management interface. This interface shows you all of the Blocks OpModes currently on the device and provides you with options for managing those OpModes.

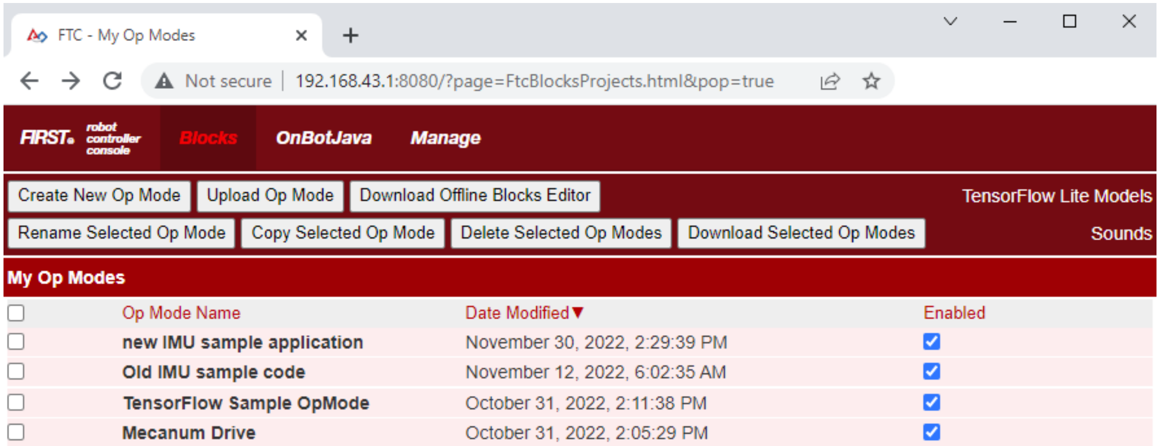


Fig. 4: Blocks Management Interface

OpModes can be downloaded through this interface. Initially, the "**Download Selected Op Modes**" button on this interface is grayed out. One or more Op Modes can be selected in this interface, and then they can all be downloaded at once. In the example below, the "Mecanum Drive" opmode is selected and then downloaded via the "**Download Selected Op Modes**" button.

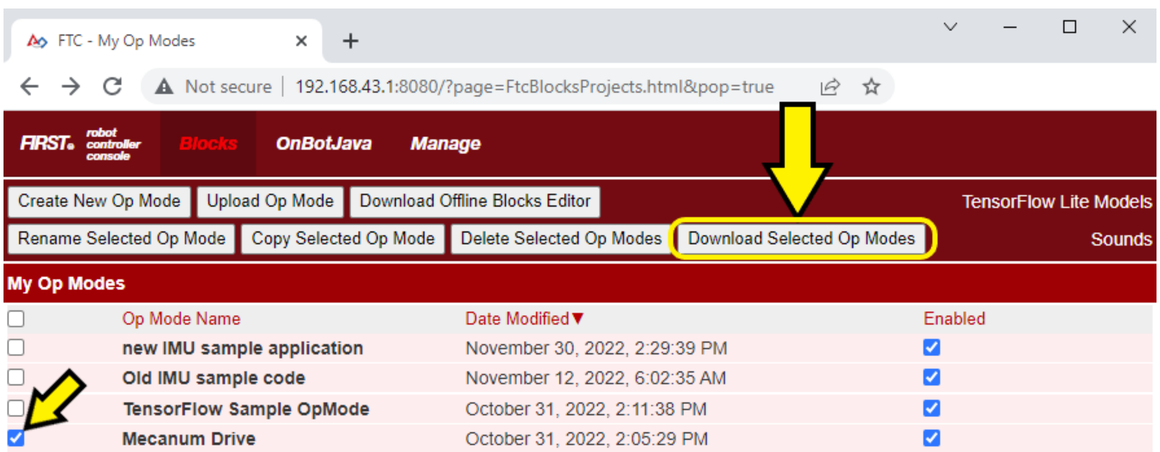


Fig. 5: Downloading Blocks via the Management Interface

Uploading Blocks

If you have a previously downloaded Blocks file, or you receive a Blocks file from another source (like sample Blocks from REV, for example) you will want to *upload* the Blocks file (**.blk**) to the device (REV Control Hub or Android Smartphone). Within the Blocks Management interface, there is a button on the top menu marked, "**Upload Op Mode**".

Once you press "**Upload Op Mode**" a pop-up window will appear to allow you to choose the file you want to upload. Click the "**Choose File**" button to open a file browser for your local computer to select the **.blk** Blocks file to upload. Once uploaded, the Blocks program will open within the Blocks interface.

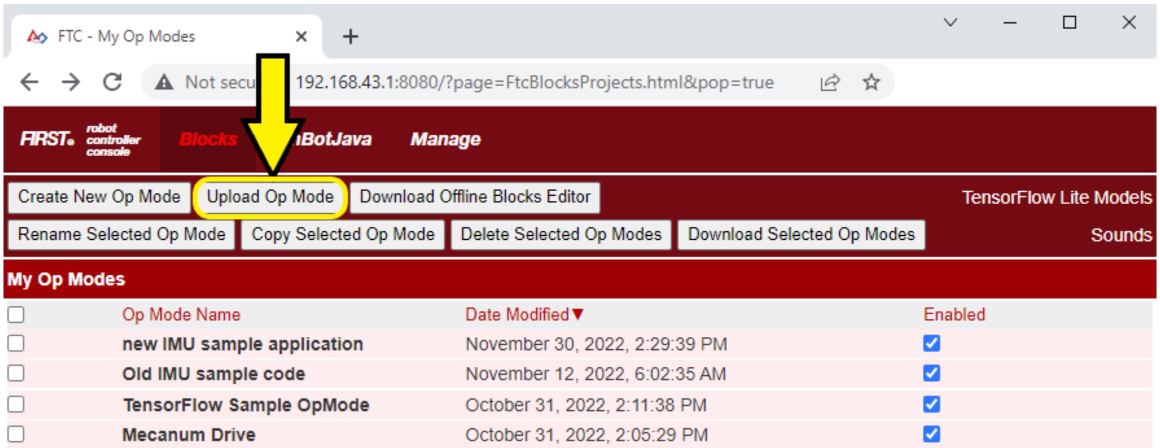


Fig. 6: Uploading Blocks Files via the Management Interface

Once a block is uploaded, it can be edited and modified like any other OpMode!

Controlling a Servo Blocks

In the section titled *Creating an Op Mode with Blocks* you learned how to use the Blocks Programming Tool to write an op mode that controls a 12V DC motor. In this section, you will learn how to write an op mode that controls a servo motor.

What is a Servo Motor?

A servo motor is a special type of motor that is designed for precise motion. A typical servo motor has a limited range of motion.

In the figure below, a "standard scale" 180-degree servo is shown. This type of servo is popular with hobbyists and with FIRST Tech Challenge teams. This servo motor can rotate its shaft through a range of 180 degrees. Using an electronic module known as a servo controller you can write an op mode that will move a servo motor to a specific position. Once the motor reaches this target position, it will hold the position, even if external forces are applied to the shaft of the servo.

Servo motors are useful when you want to do precise movements (for example, sweep an area with a sensor to look for a target or move the control surfaces on a remotely controlled airplane).

Modifying Your Op Mode to Control a Servo

Let's modify your op mode to add the logic required to control a servo motor. For this example, you will use the buttons on the Logitech F310 gamepad to control the position of the servo motor.

With a typical servo, you can specify a target position for the servo. The servo will turn its motor shaft to move to the target position, and then maintain that position, even if moderate forces are applied to try and disturb its position.

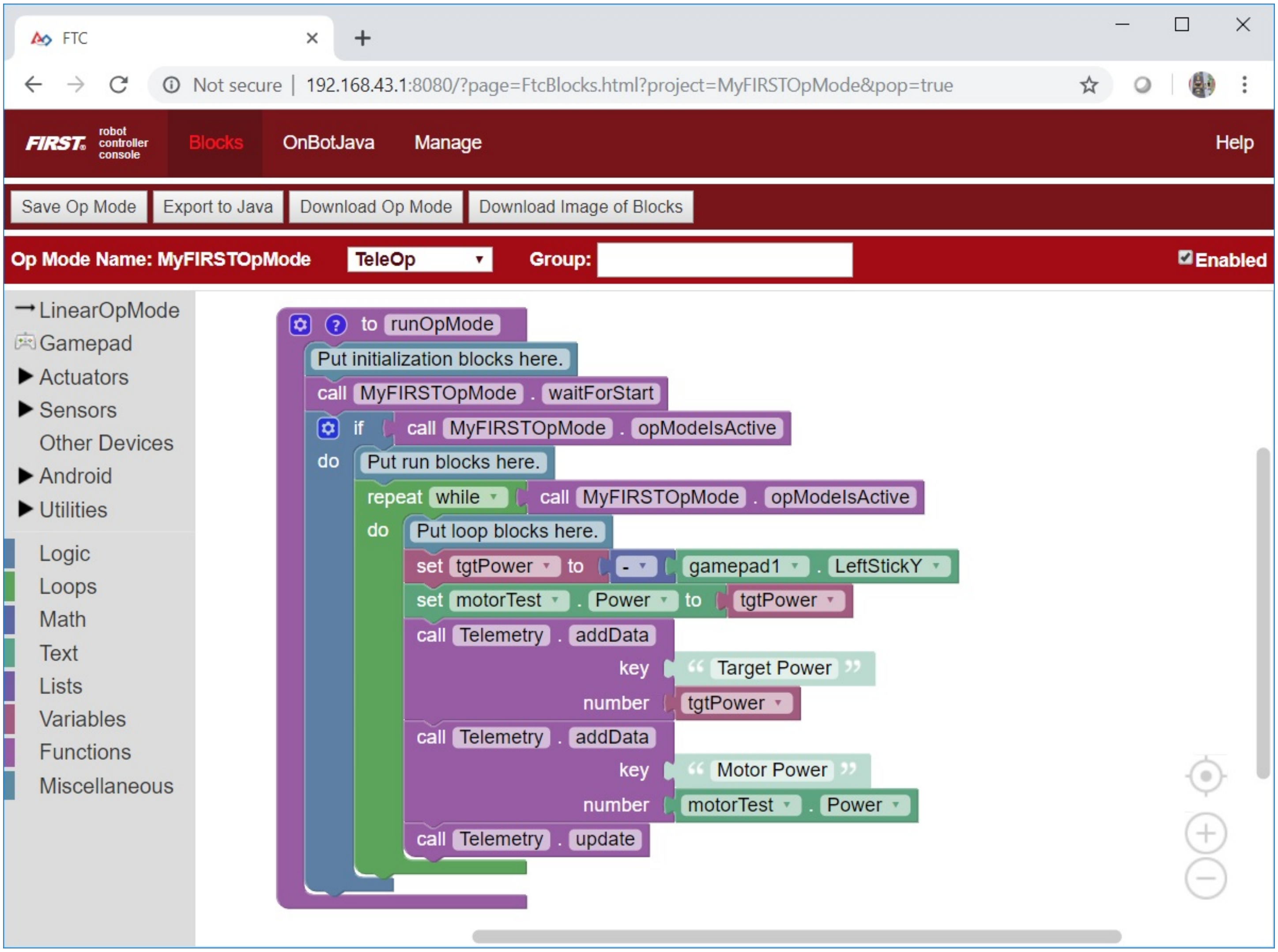
For the blocks Program & Manage server, you can specify a target position that ranges from 0 to 1 for a servo. A target position of 0 corresponds to zero degrees of rotation and a target position of 1 corresponds to 180 degrees of rotation for a typical servo motor.

In this example, you will use the colored buttons on the right side of the F310 controller to control the position of the servo. Initially, the op mode will move the servo to the midway position (90 degrees of its 180-degree range). Pushing the yellow "Y" button will move the servo to the zero-degree position. Pushing the blue "X" button or the red "B" button will move the servo to the 90-degree position. Pushing the green "A" button will move the servo to the 180-degree position.

Modifying the Op Mode to Control a Servo Motor Instructions

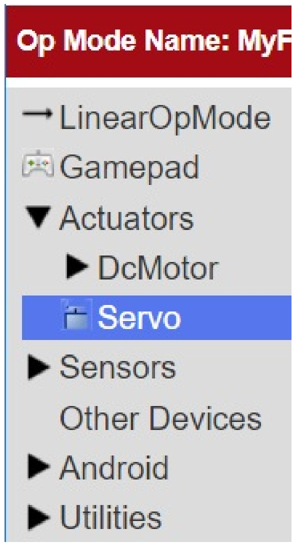
1. Verify that your laptop is still connected to the Robot Controller's Program & Manage Wi-Fi network.

2. Verify that "MyFIRSTOpMode" is opened for editing. If it is not, you can click on the FIRST logo in the upper left-hand corner of the browser window on the laptop. This should take you to the main Blocks Development Tool project screen.

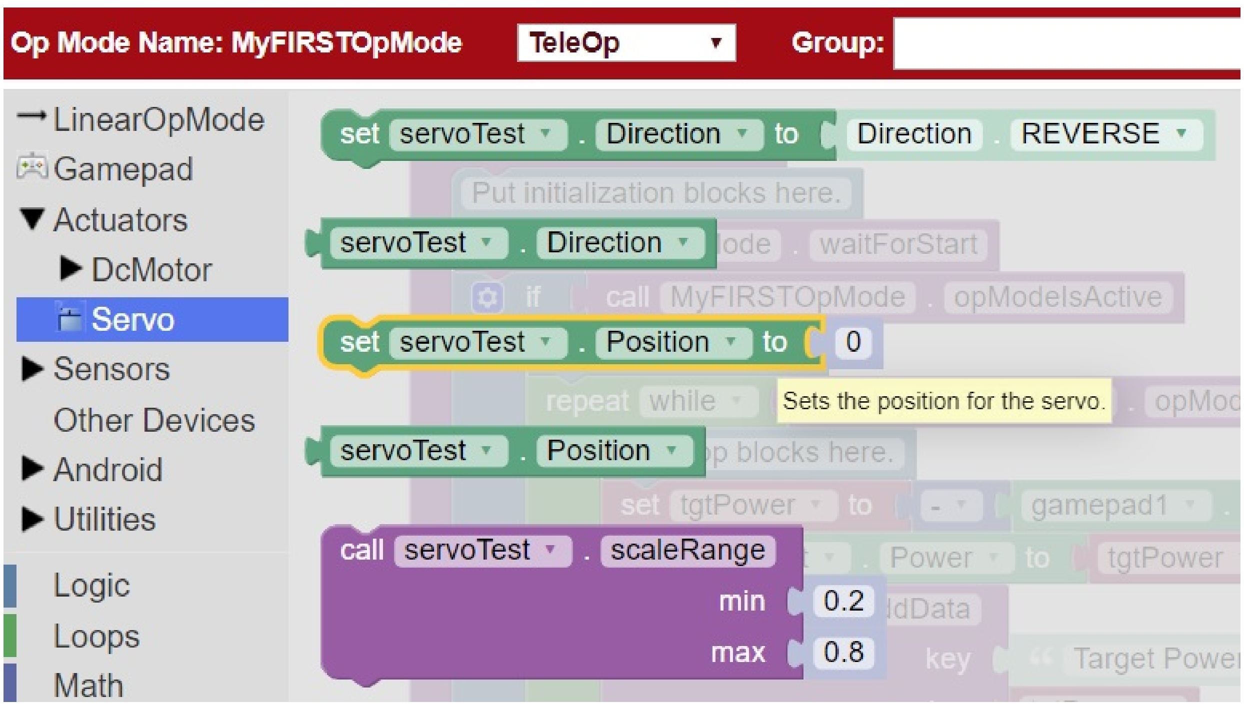


Click on the “MyFIRSTOpMode” project to open it for editing if it is not already opened.

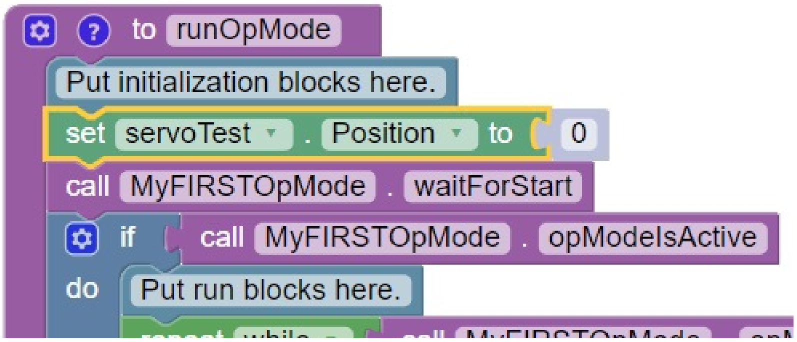
3. On the left-hand side of the screen click on the category called “Actuators” and look for the subcategory called “Servos”.



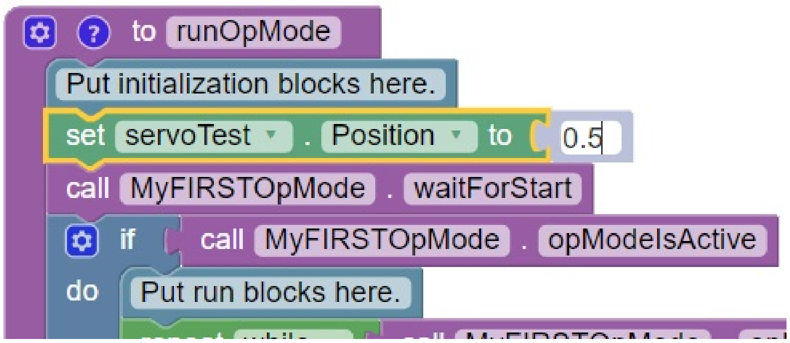
4. Select the "set servoTest.Position to" block from the list of available Servo blocks.



5. Drag the "set servoTest.Position to" block to the spot just under the comment block that reads "Put initialization blocks here." The block should click into place.

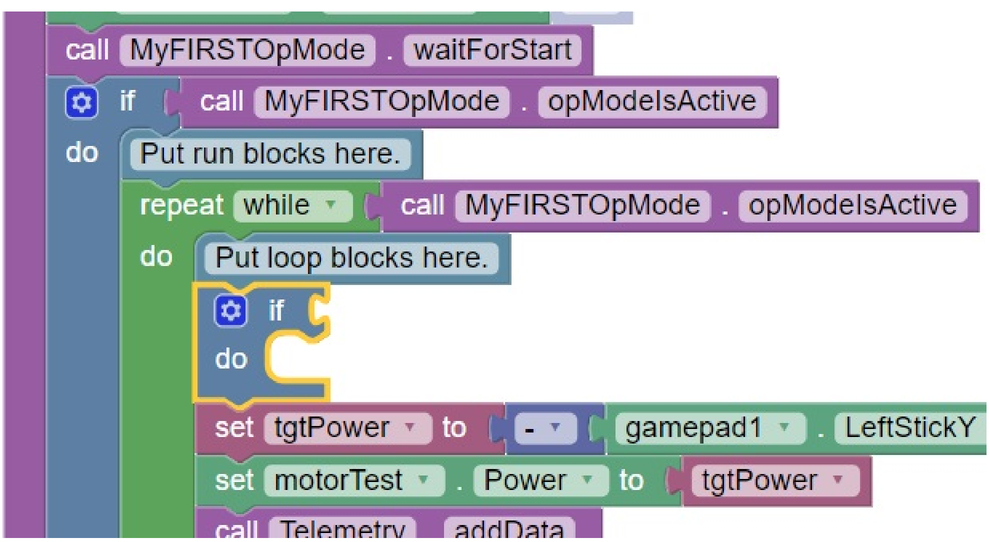


6. Click on the number block "0" and change the block's value to "0.5".



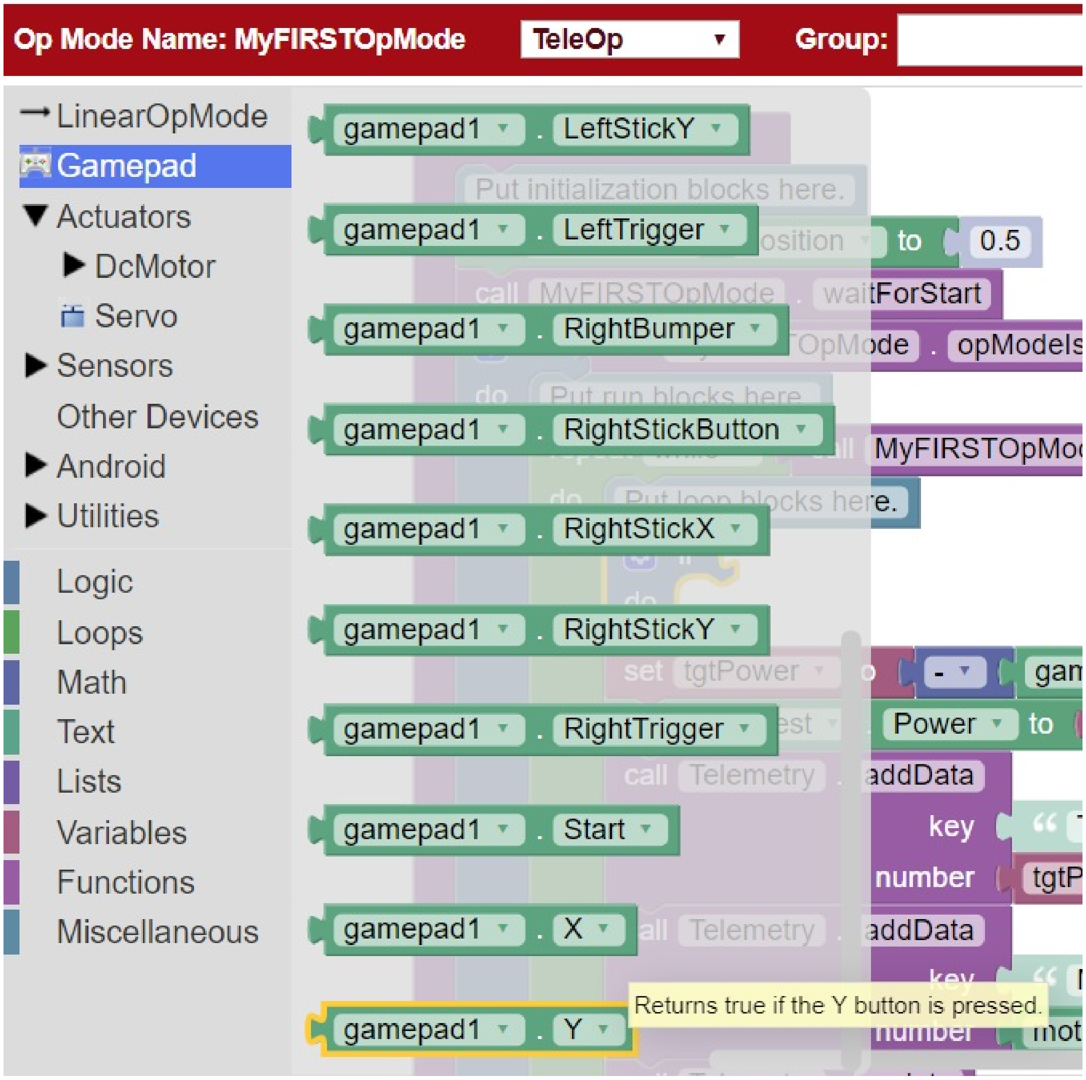
When a user selects this op mode, the servo position will initially be set to the midway point (90-degree position).

7. Click on the "Logic" category of the programming blocks and select the "if do" block from the list of available blocks. Drag the block to the position immediately after the comment block that reads "Put loop blocks here."



The block should click into place.

8. Click on the “Gamepad” category of the programming blocks and select the “gamepad1.Y” block from the list of available blocks.

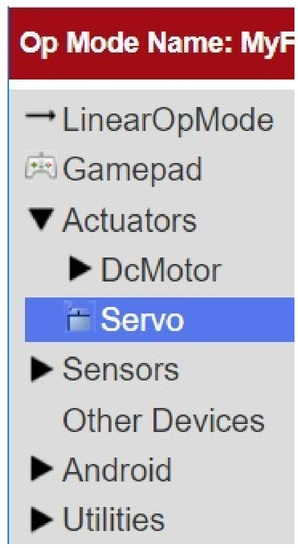


Note that this block is towards the bottom of the list of blocks. You might have to scroll down to the bottom of the list before you can select this block.

9. Drag the “gamepad1.Y” block to the right side of the “if do” block. The block should click into place.

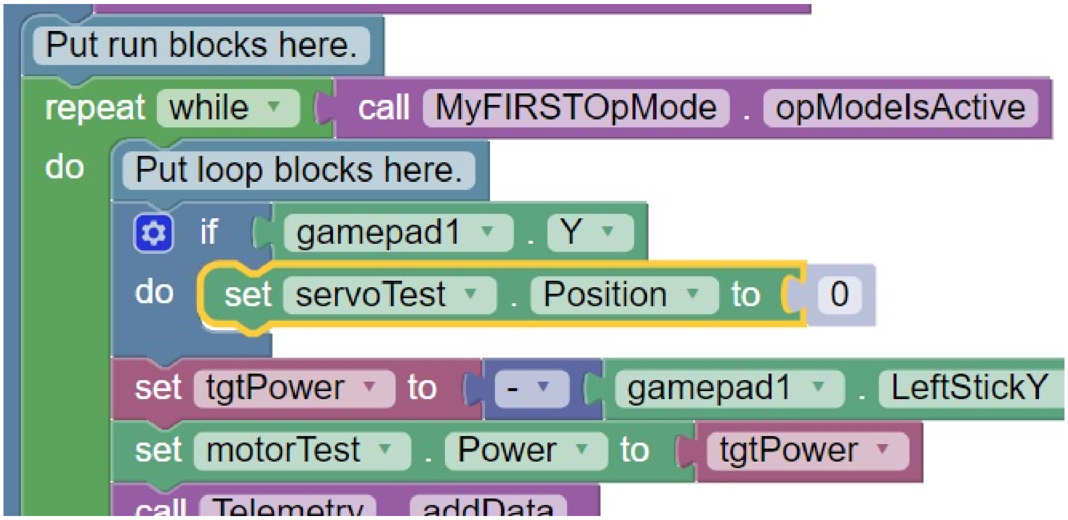
The “if do” block will use the state of the gamepad1.Y value its test condition. If the “Y” button is pressed, the statements within the “do” portion of the block will be executed.

10. On the left-hand side of the screen click on the category called “Actuators” and look for the subcategory called “Servos”.



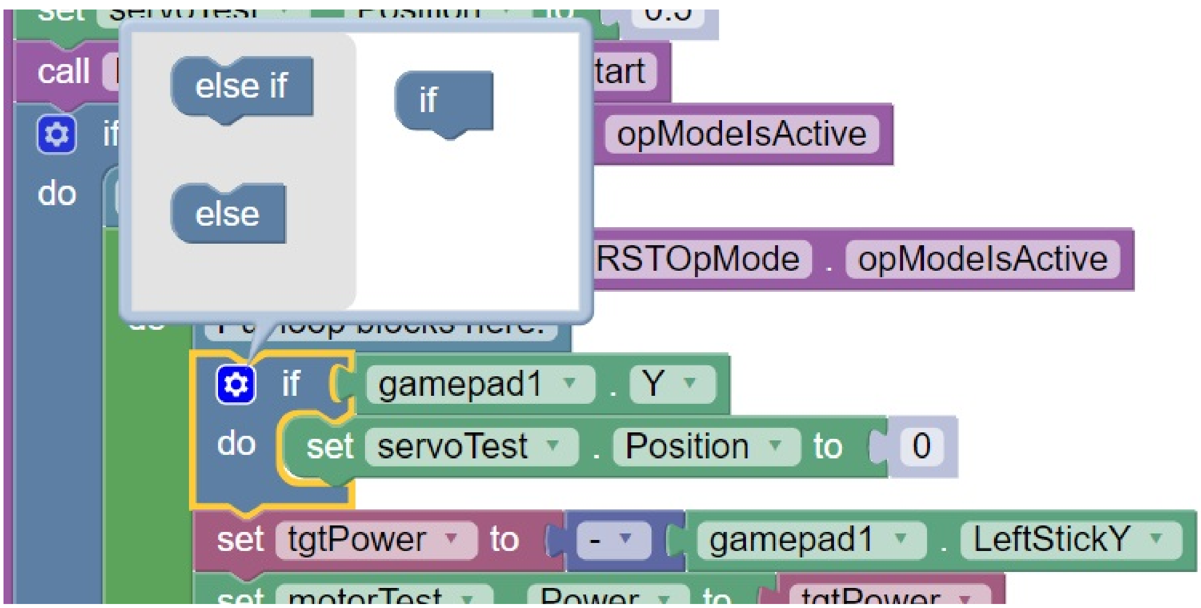
11. Select the “set servoTest.Position to” block from the list of available Servo blocks.

12. Drag the “set servoTest.Position to” block so that it snaps in place in the do portion of the “if do” block.

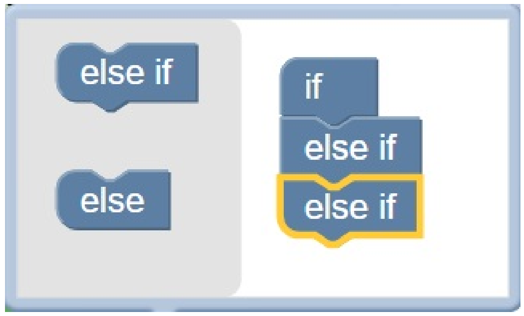


If the "Y" button is pressed on gamepad #1, the op mode will move the servo's position to the 0-degree position.

13. Click on the blue and white Settings icon for the "if do" block. This will display a pop-up menu that lets you modify the "if do" block.

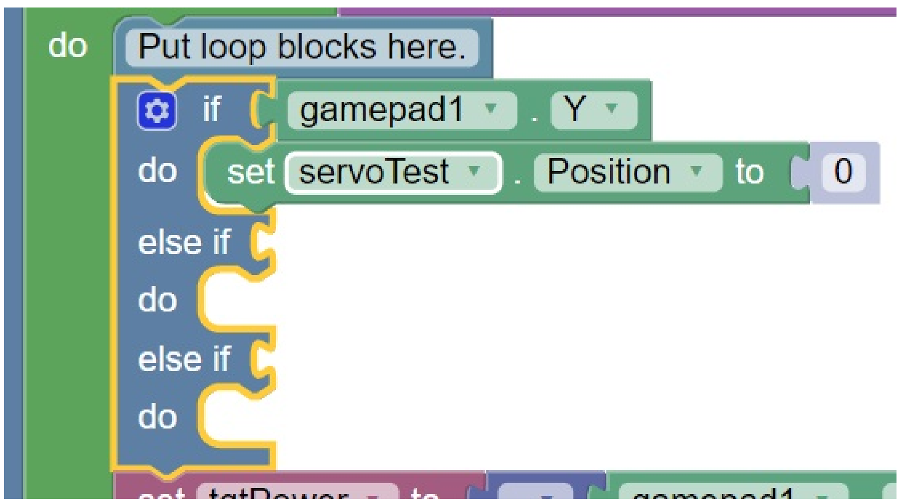


14. Drag an "else if" block from the left side of the pop-up menu and snap it into place under the "if" block.

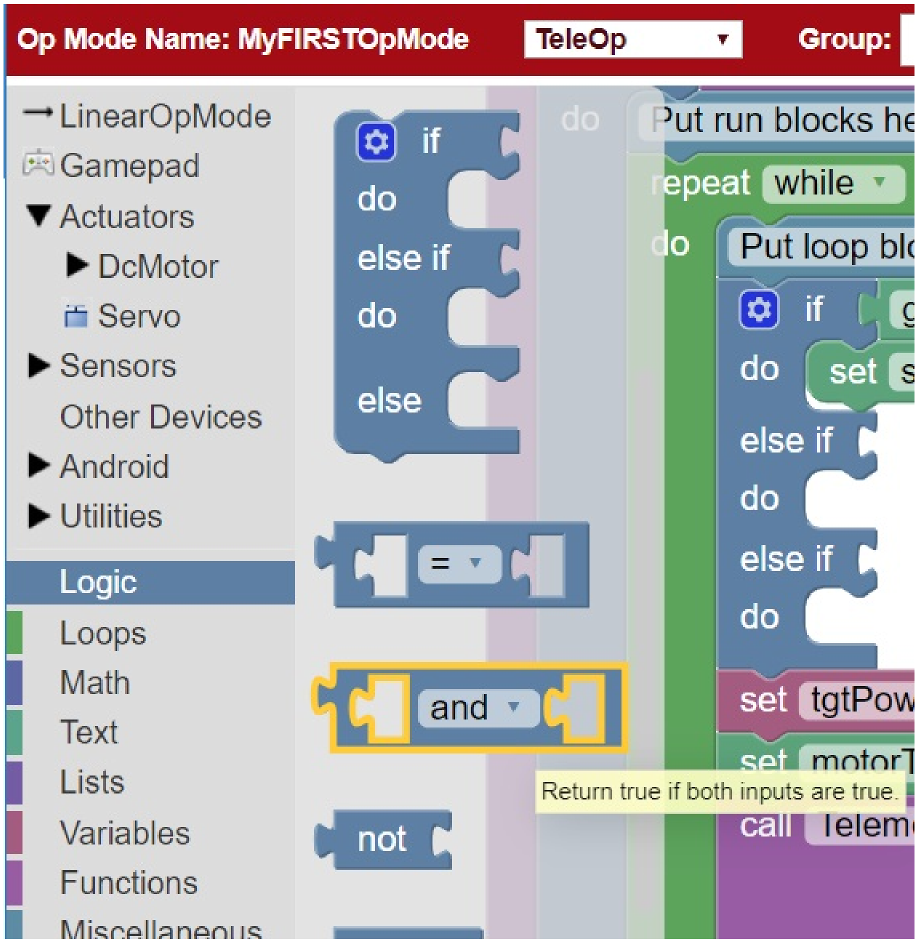


Drag a second “else if” block from the left side and snap it into place on the right side under the first “else if” block.

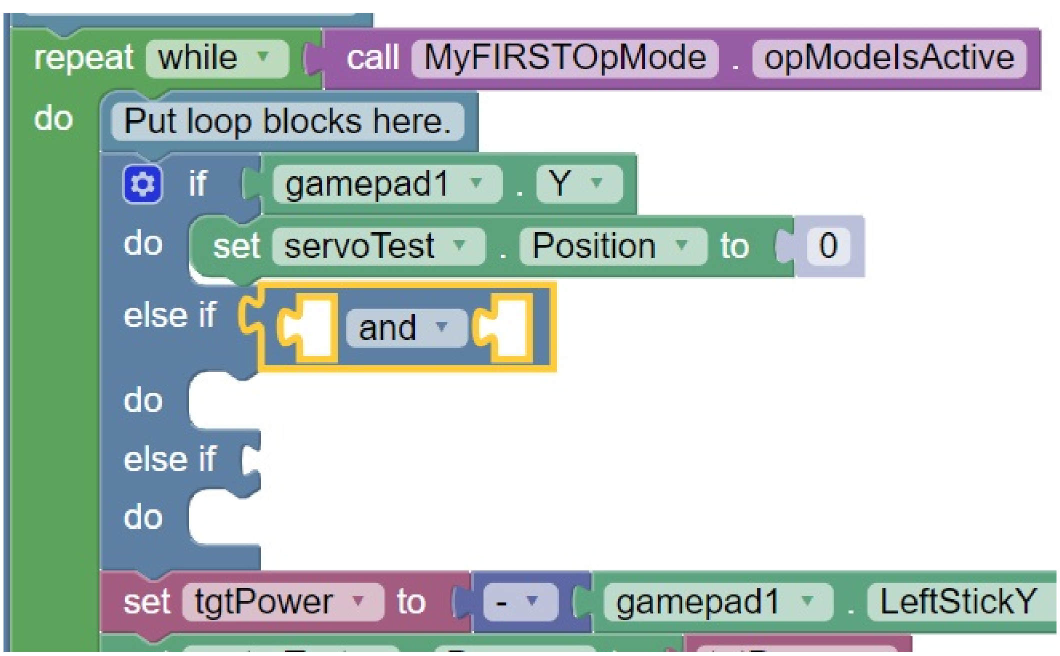
15. Click on the Settings icon to hide the pop-up menu for the “if do” block. The “if do” block should now have two “else if” test conditions added.



16. Click on the “Logic” category and select the logical “and” block.

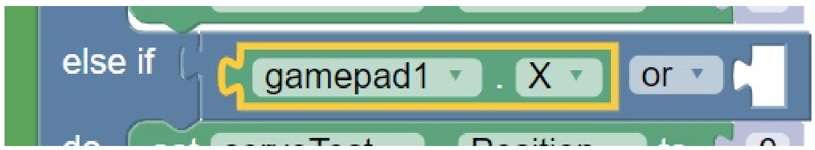


17. Drag the "and" block so it clicks in place as the test condition for the first "else if" block.

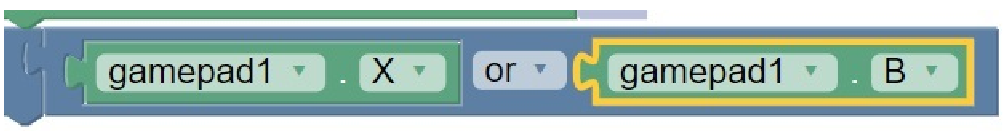


18. Click on the word "and" and select "or" from the pop-up menu to change the block to a logical "or" block.

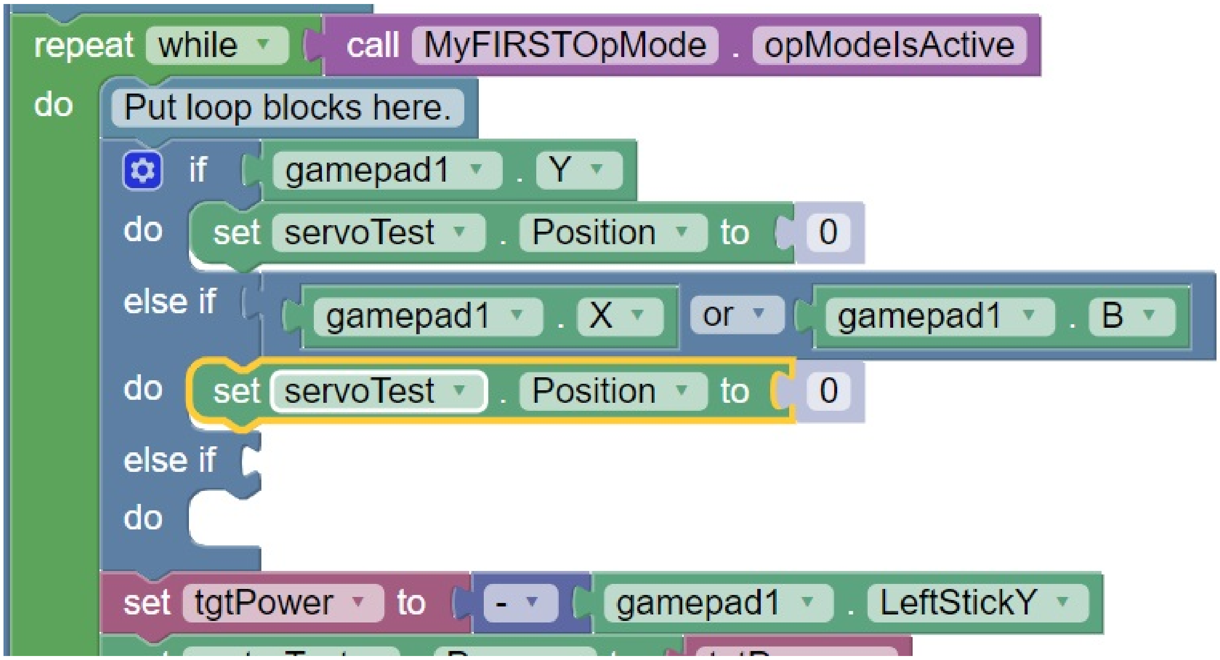
19. Click on the "Gamepad" category and select the "gamepad1.X" block. Drag the block so that it clicks in place as the first test condition of the logical "or" block.



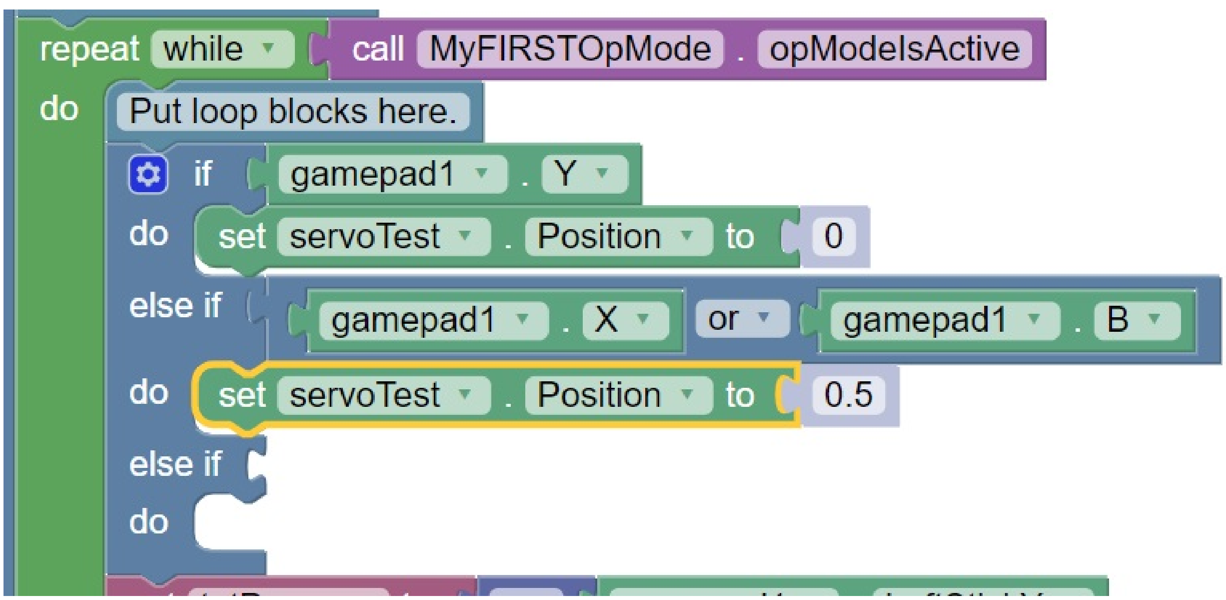
20. Click on the "Gamepad" category and select the "gamepad1.B" block. Drag the block so that it clicks in place as the second test condition of the logical "or" block.



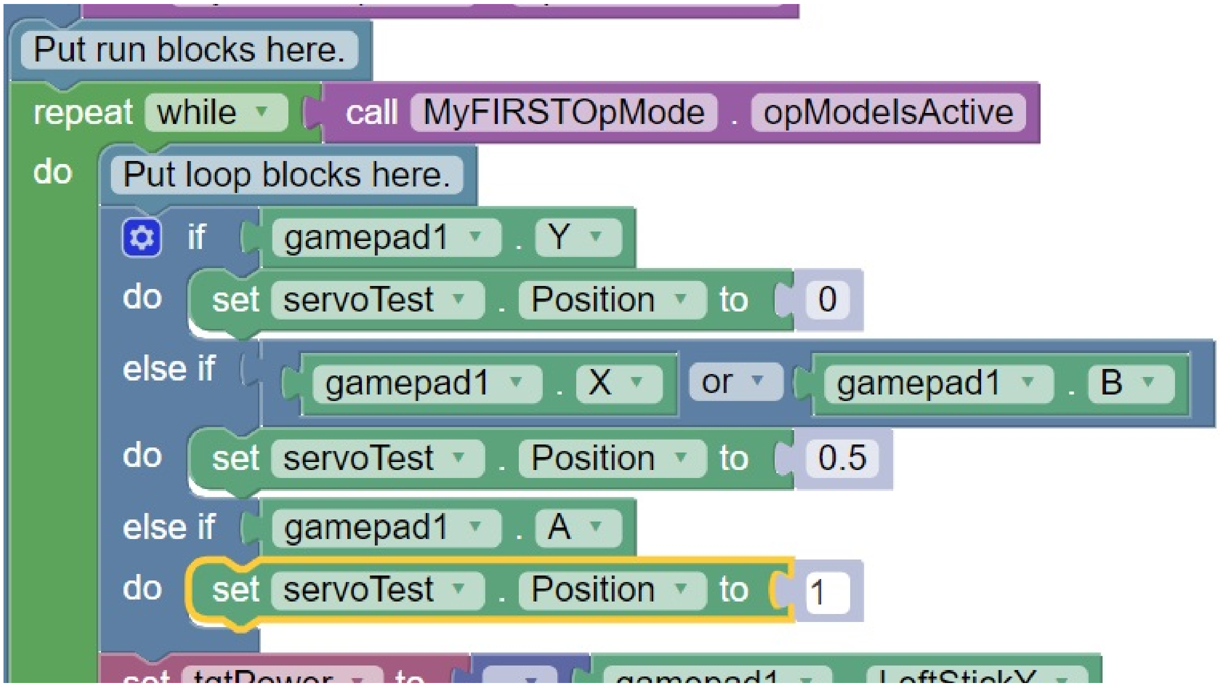
21. Select a "set servoTest.Position to" block and place it into "do" clause of the first else-if block.



22. Highlight the number “0” and change it to “0.5”. With this change, if the user presses the “X” button or “B” button on gamepad #1, the op mode will move the servo to the midway (90-degree) position.

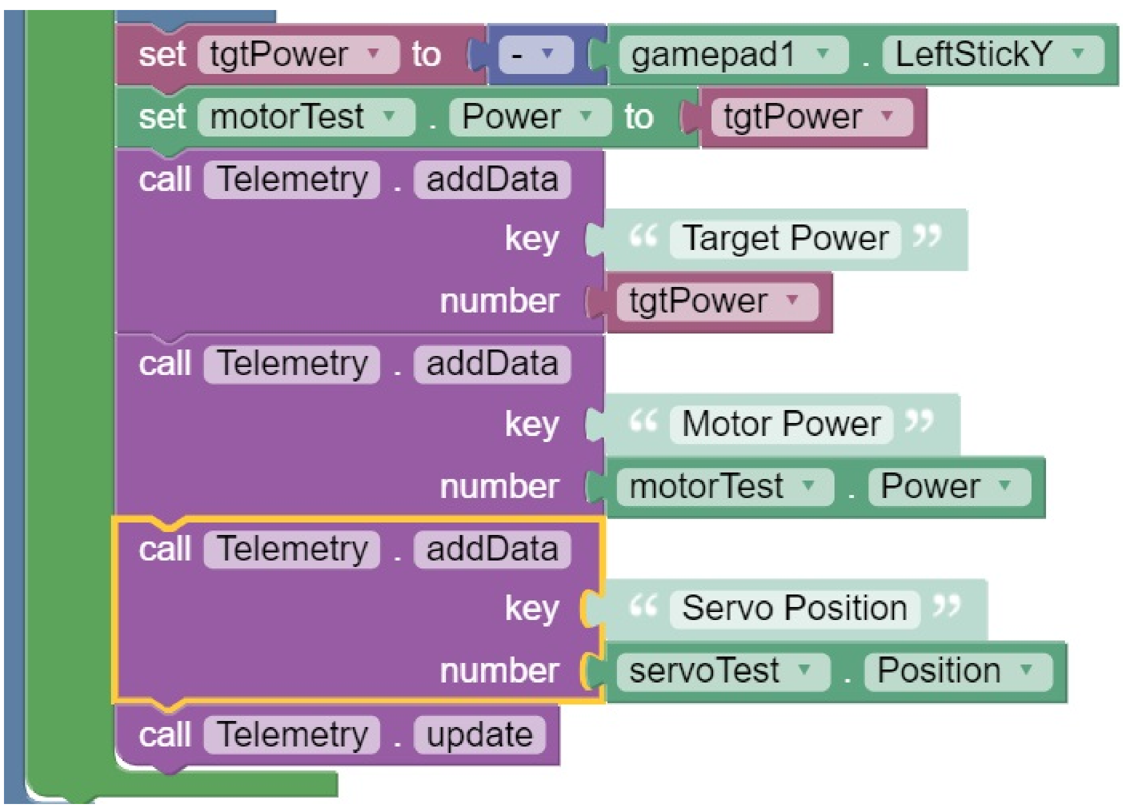


23. Use a “gamepad1.A” block as the test condition for the second “else if” block. Drag a “set servoTest.position to” block to the do clause of the second “else if” block and modify the numeric value so that the servo’s position will be set to a value of 1.



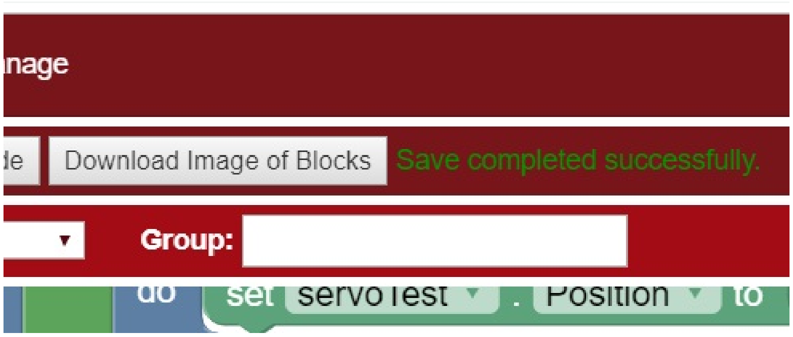
For this clause, if the "A" button is pressed on the #1 gamepad, the op mode will move the servo to the 180-degree position.

24. Insert a "call telemetry.addData" block (numeric) before the "call Telemetry.update" block. Rename the key field to "Servo Position" and insert a "servoTest.Position" block for the number field.

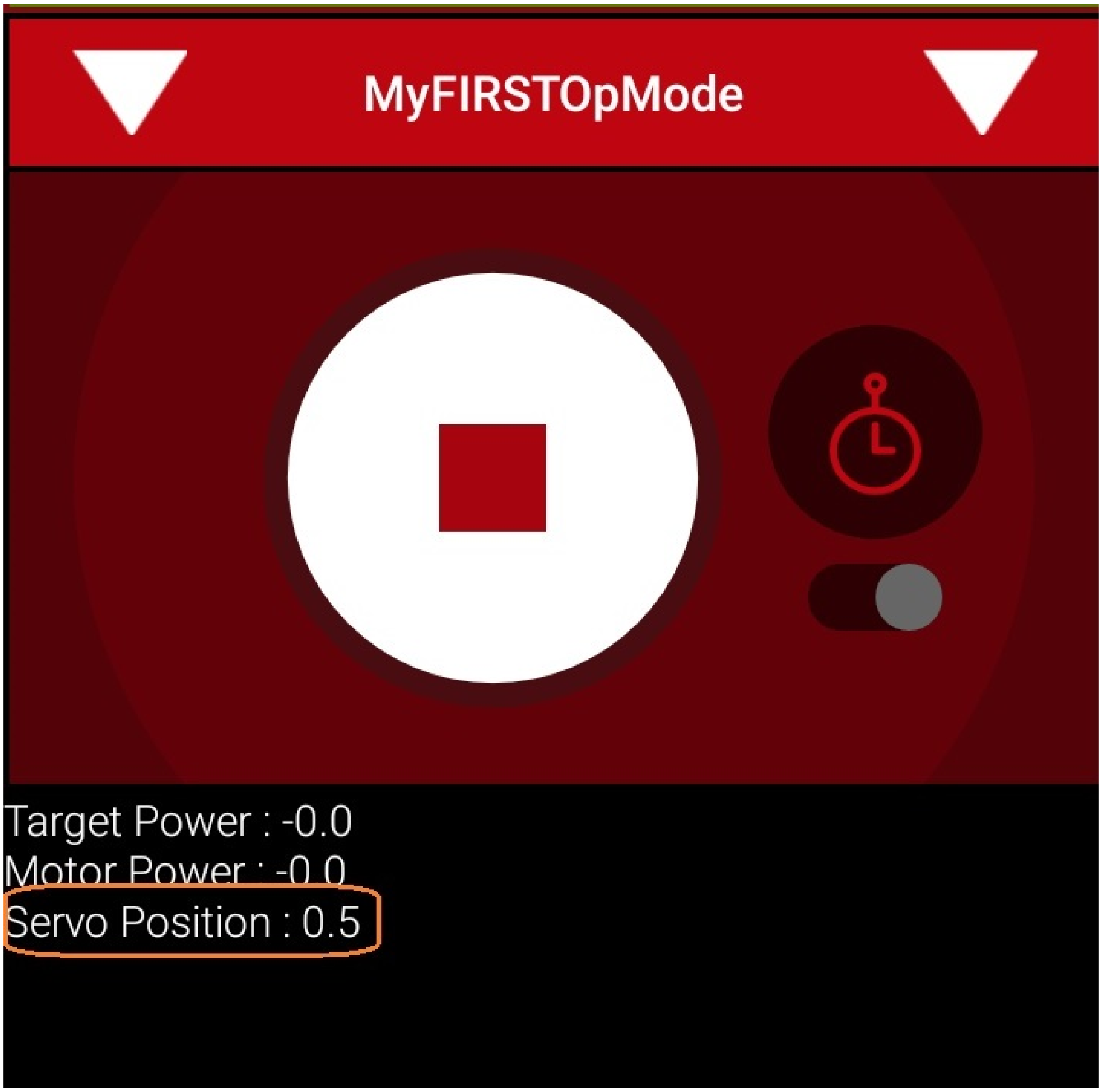


This set of blocks will send the current servo position value to the DRIVER STATION while the op mode is running.

25. Save your op mode and verify that it was saved successfully to the Robot Controller.



26. Follow the procedure outlined in the section titled *Running Your OpMode* to run your updated op mode. Also, make sure that your gamepad is designated as User #1 before running your op mode.



You should now be able to control the servo position with the colored buttons. The servo position should be displayed on the DRIVER STATION.

Using Sensors Blocks

Color-Distance Sensor

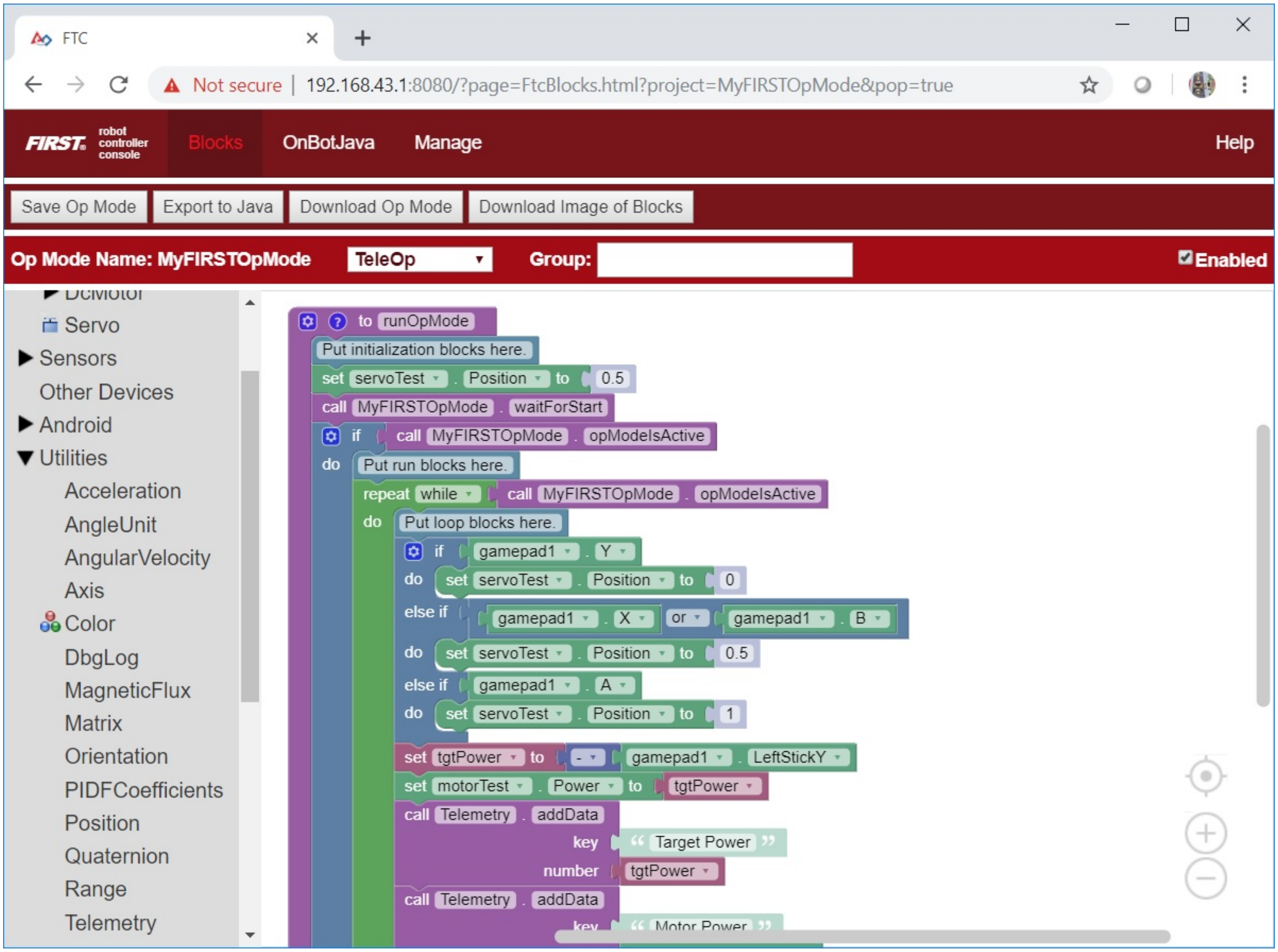
A sensor is a device that lets the Robot Controller get information about its environment. In this example, you will use a REV Robotics Color-Distance sensor to display range (distance from an object) info to the DRIVER STATION.

The Color-Range sensor uses reflected light to determine the distance from the sensor to the target object. It can be used to measure close distances (up 5" or more) with reasonable accuracy. Note that at the time this document was most recently edited, the REV Color-Range sensor saturates around 2" (5cm). This means that for distances less than or equal to 2", the sensor returns a measured distance equal to 2" or so.

Note that it will take an estimated 15 minutes to complete this task.

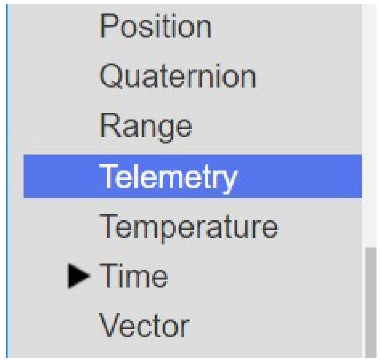
Modifying the Op Mode to Display Distance Instructions

1. Verify that your laptop is still connected to the Robot Controller's Program & Manage Wi-Fi network.
2. Verify that "MyFIRSTOpMode" is opened for editing. If it is not, you can click on the FIRST logo in the upper left hand corner of the browser window on the laptop. This should take you to the main Blocks Development Tool project screen.

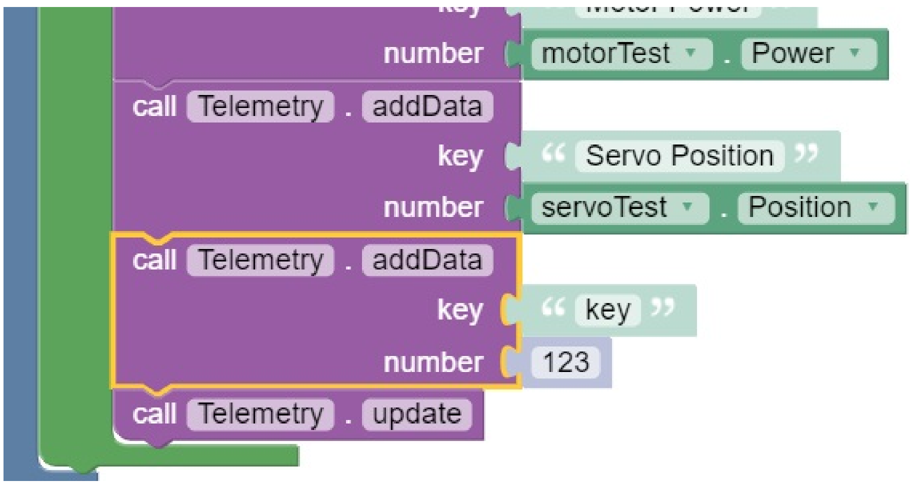


Click on the "MyFIRSTOpMode" project to open it for editing if it is not already opened.

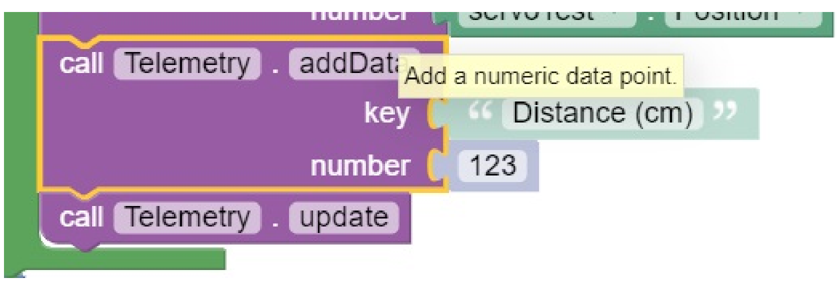
3. Click on the "Utilities" category on the left-hand side of your browser. Find and click on the "Telemetry" subcategory.



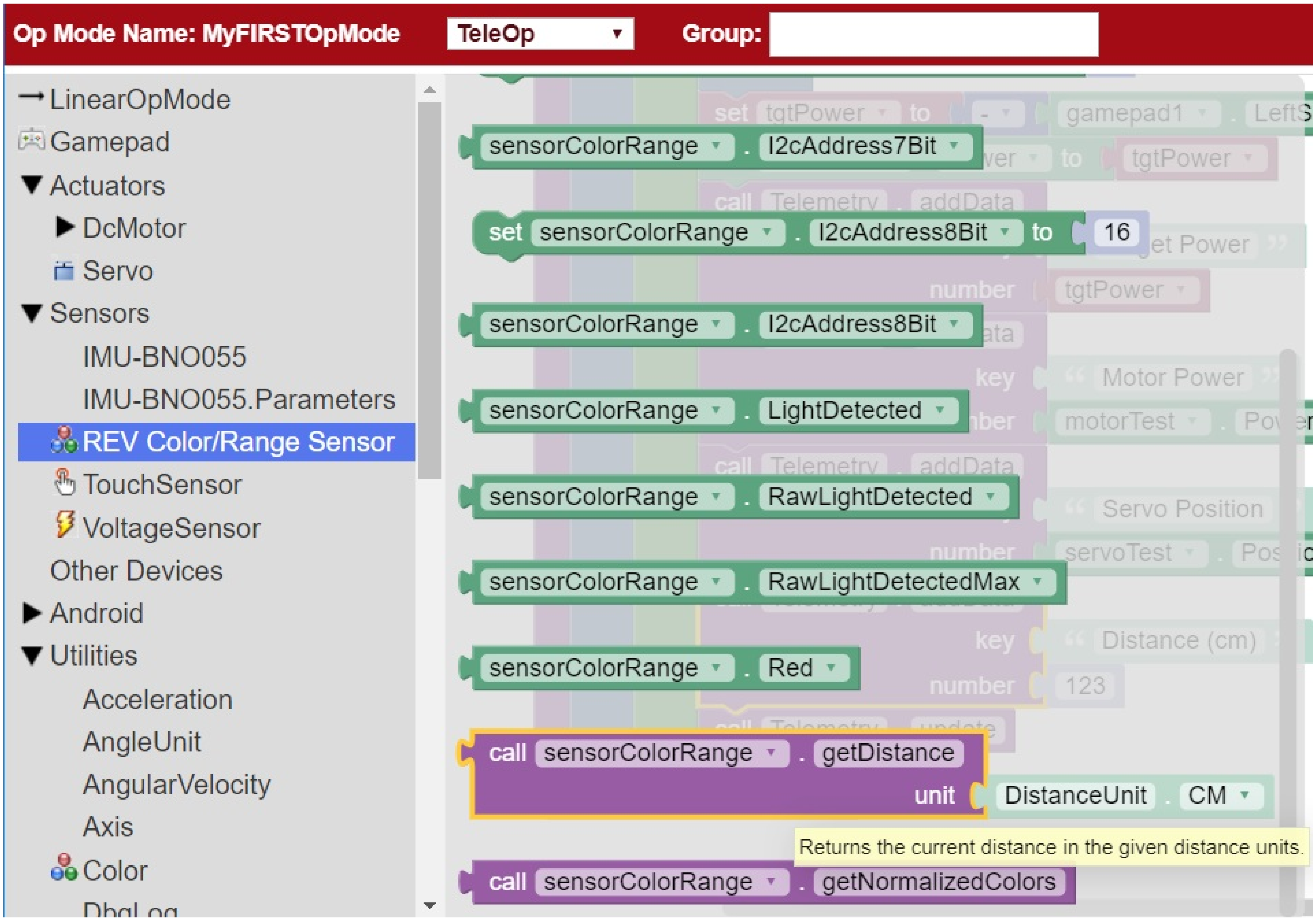
4. Select the "call telemetry.addData" block (the numeric version) and drag it to the spot in your "while" loop block immediately before the "telemetry.update" block.



5. Click and highlight the "key" text and change the text so it reads "Distance (cm)".

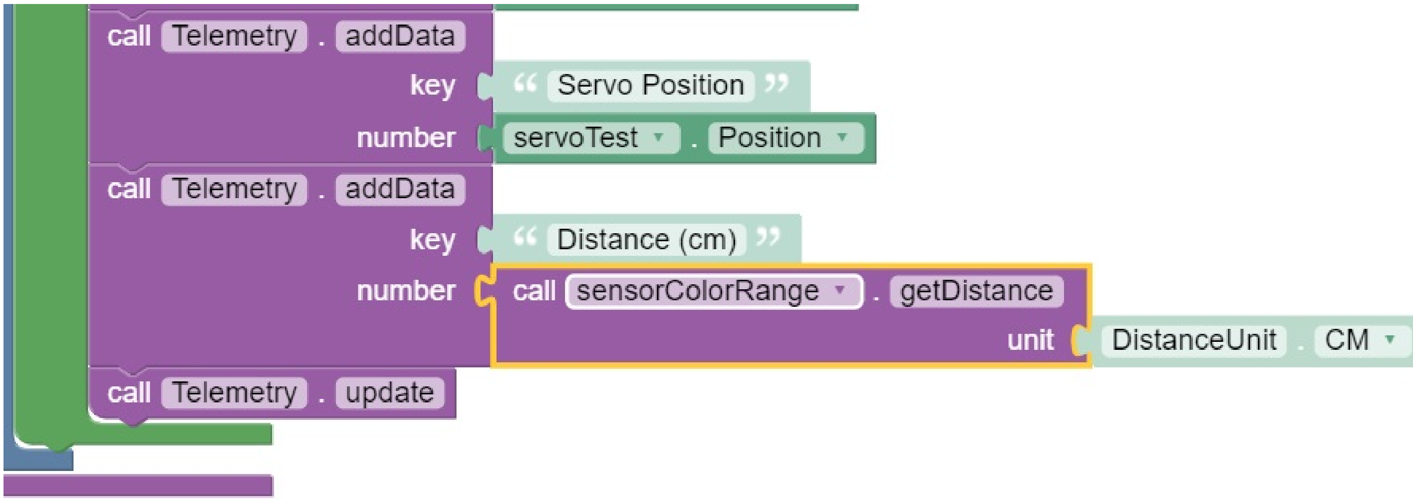


6. Click and expand the "Sensors" category. Click on the "REV Color/Range Sensor" subcategory. Click on and select the "call sensorColorRange.getDistance" programming block.



Note that earlier versions of the Blocks Programming tool refer to the REV Robotics Color-Distance Sensor as the “LynxI2cColorRangeSensor”. Newer versions of the software refer to the device as the “REV Color/Range Sensor”.

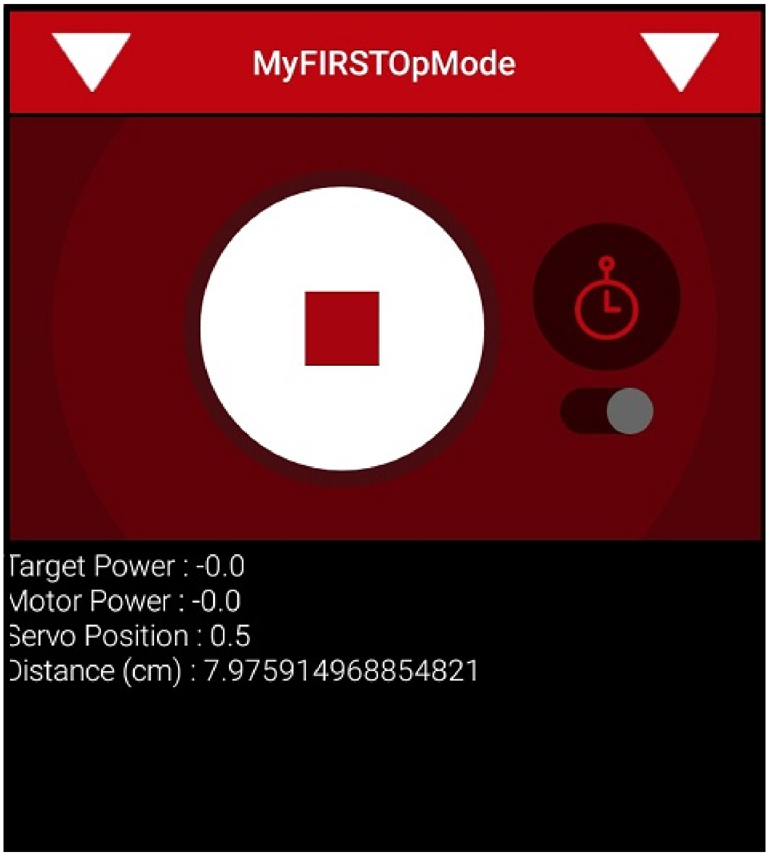
7. Drag the “call sensorColorRange.getDistance” programming block to the “number” field of the “call telemetry.addData” programming block.



This will send the measured distance to the target in centimeters back to the DRIVER STATION.

8. Save your op mode and verify that it was saved successfully to the Robot Controller.

9. Follow the procedure outlined in the section titled *Running Your OpMode* to run your updated op mode.



As you run the op mode, if you move your hand above the color light sensor, you should see the measured distance change on the DRIVER STATION screen. If the expression “NaN” (not a number) is displayed on the DRIVER STATION, the target is most likely out of range (and the sensor does not detect any reflected light).

Touch Sensor

For this example, we assume that the REV Robotics Touch Sensor has been configured as a digital touch sensor in the Robot Controller’s active configuration file. We will use the “isPressed” programming block to determine if the button on the sensor is currently pressed or not.

The Control Hub or Expansion Hub digital ports contain two digital pins per port. When you use a 4-wire JST cable to connect a REV Robotics Touch sensor to a Control Hub or Expansion Hub digital port, the Touch Sensor is wired to the second of the two digital pins within the port. The first digital pin of the 4-wire cable remains disconnected.

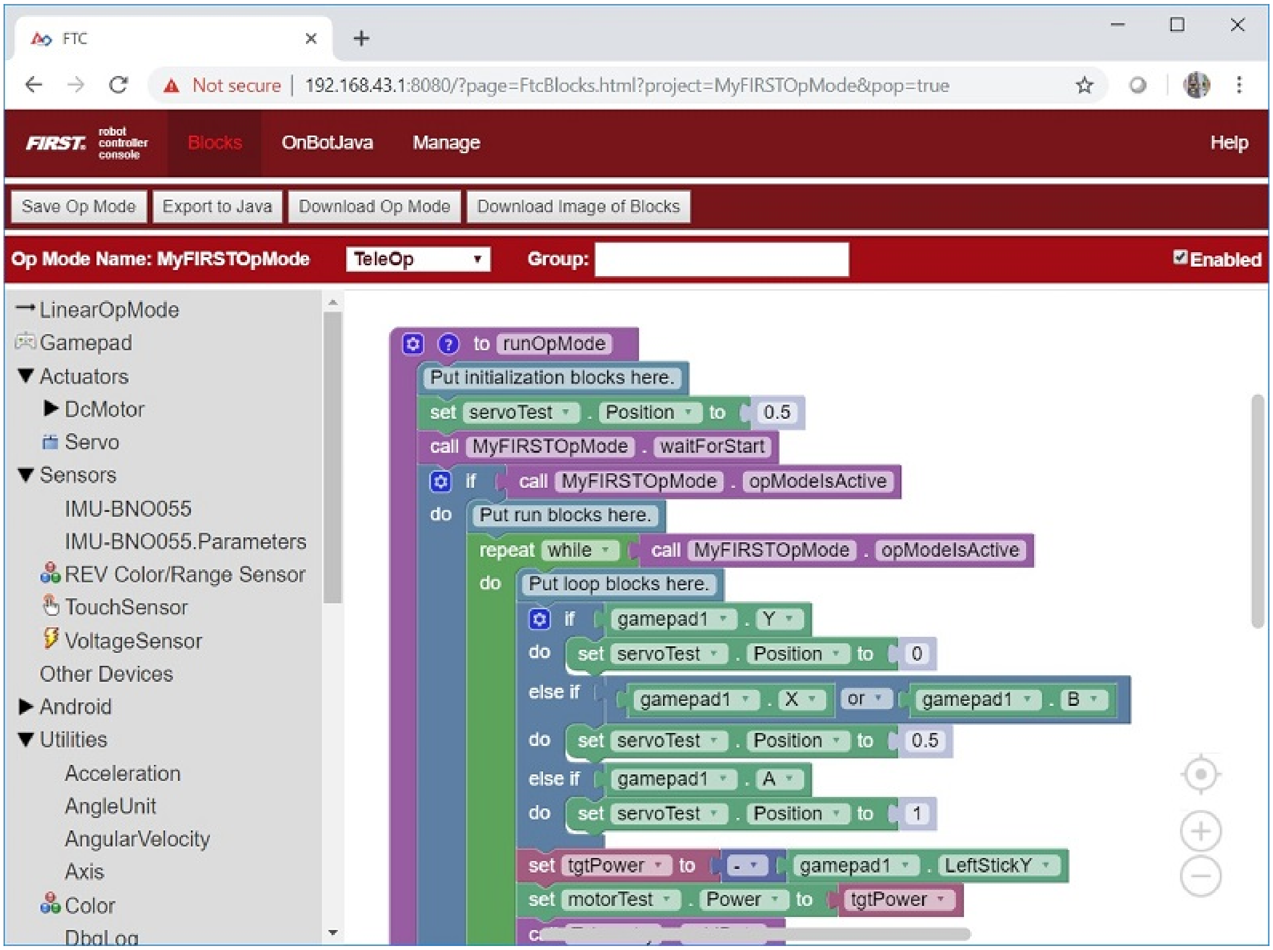
For example, if you connect a Touch Sensor to the "0,1" digital port of the Control Hub or Expansion Hub, the Touch Sensor will be connected to the second pin (labeled "1") of the port. The first pin (labeled "0") will stay disconnected.

Note that it will take an estimated 15 minutes to complete this task.

Modifying the Op Mode to Display Button (Touch Sensor) State Instructions

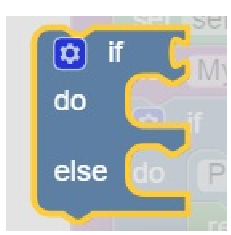
1. Verify that your laptop is still connected to the Robot Controller's Programming Mode Wi-Fi network.

2. Verify that "MyFIRSTOpMode" is opened for editing. If it is not, you can click on the FIRST logo in the upper left hand corner of the browser window on the laptop. This should take you to the main Blocks Development Tool project screen.

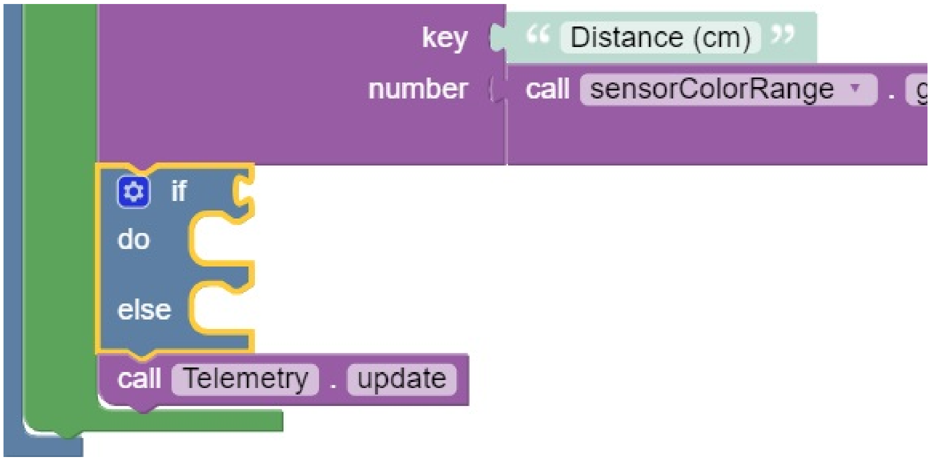


Click on the "MyFIRSTOpMode" project to open it for editing if it is not already opened.

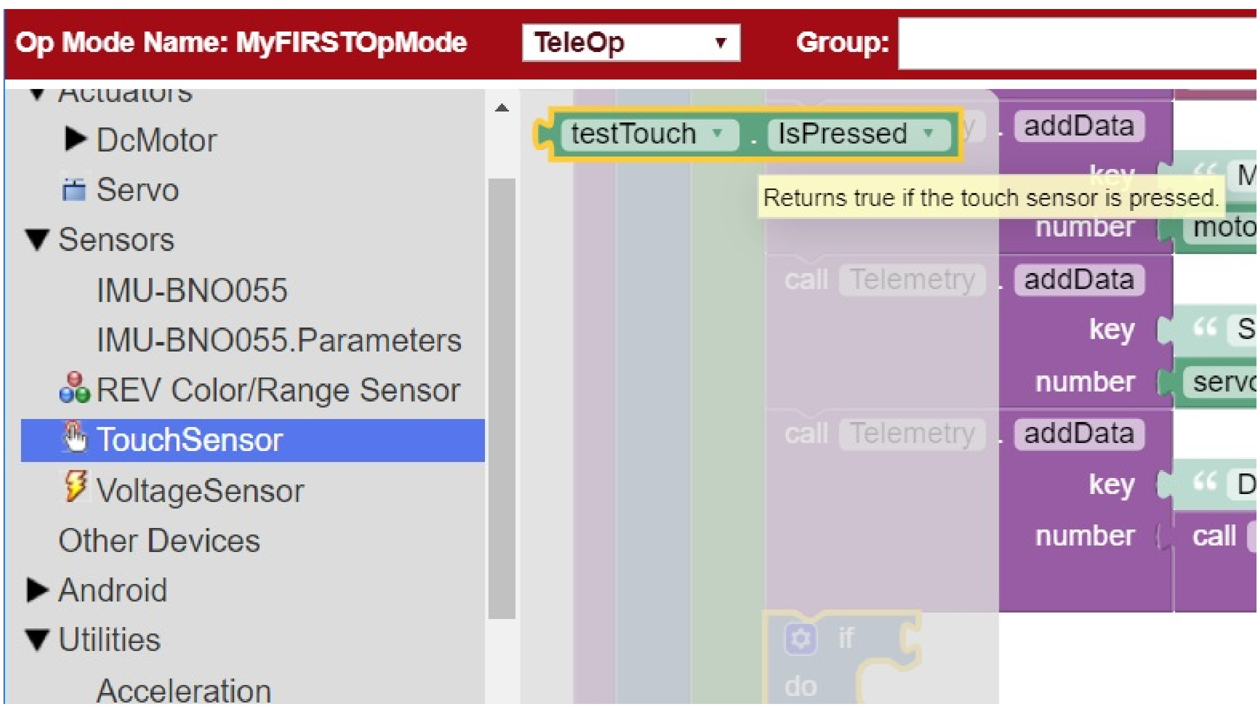
3. Click on the "Logic" category. Find and click on the "if do else" block.



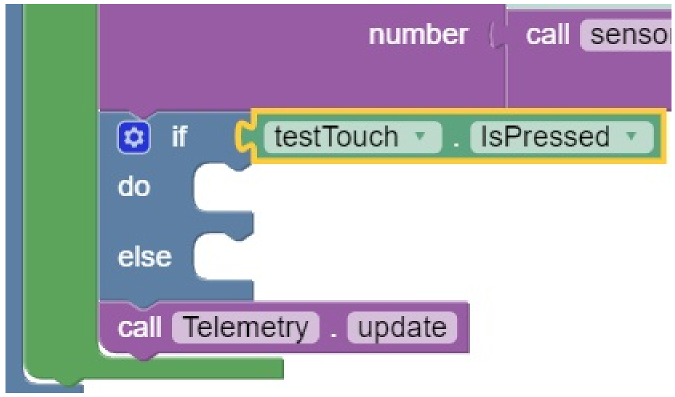
4. Drag the “if do else” block to the position before the “telemetry.update” block.



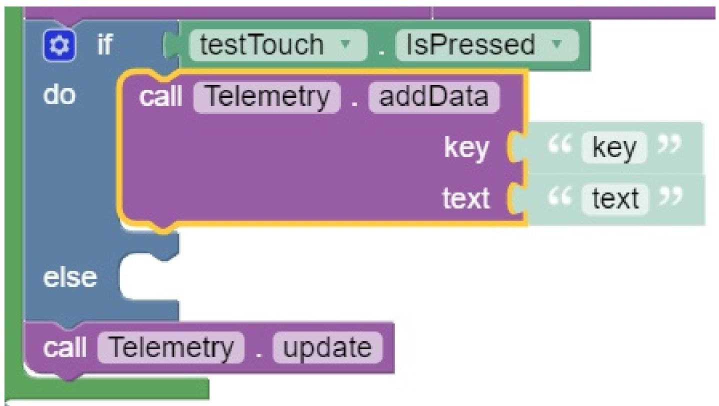
5. Click on the “Sensors” category to expand it (if it isn’t already expanded). Click on the “Touch Sensor” subcategory, then find and select the “.isPressed” block.



6. Drag the “isPressed” block to the test condition of the “if do else” programming block.

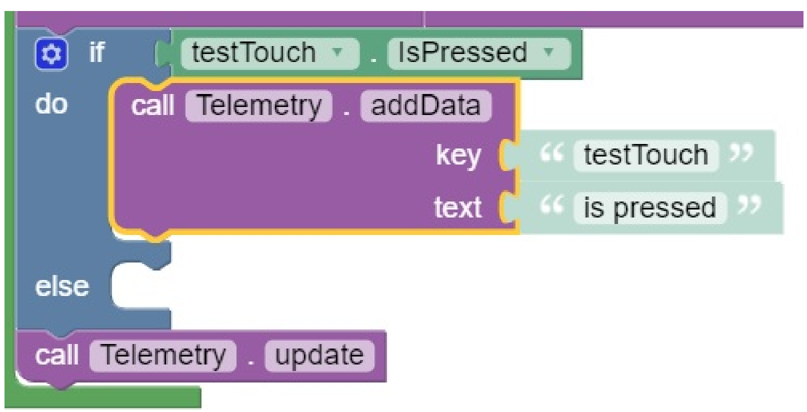


7. Click on the "Utilities" category on the left-hand side of your browser. Find and click on the "Telemetry" subcategory.

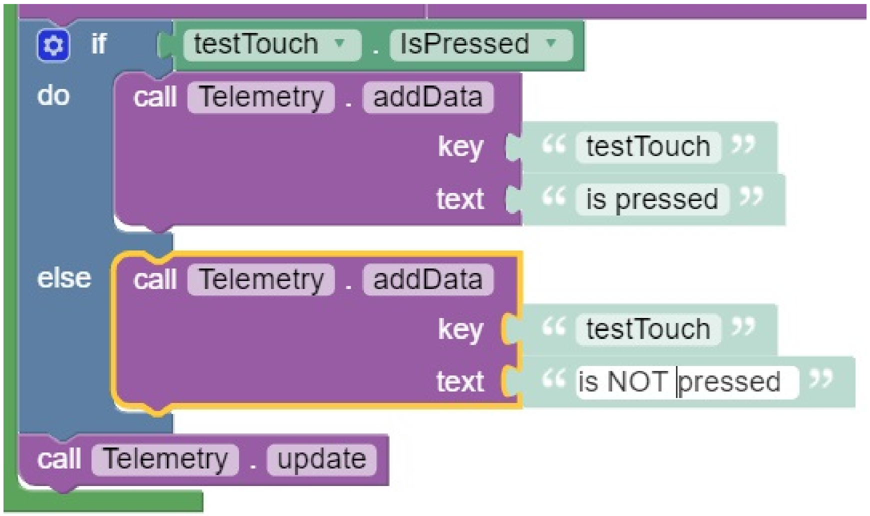


Select the "call telemetry.addData" block (the text version) and drag it to the "do" clause of the "if do else" block.

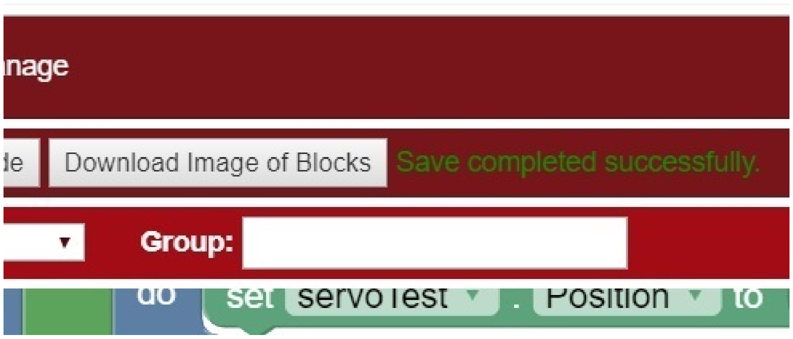
8. Change the "key" value to "testTouch" and the "text" value to "is pressed".



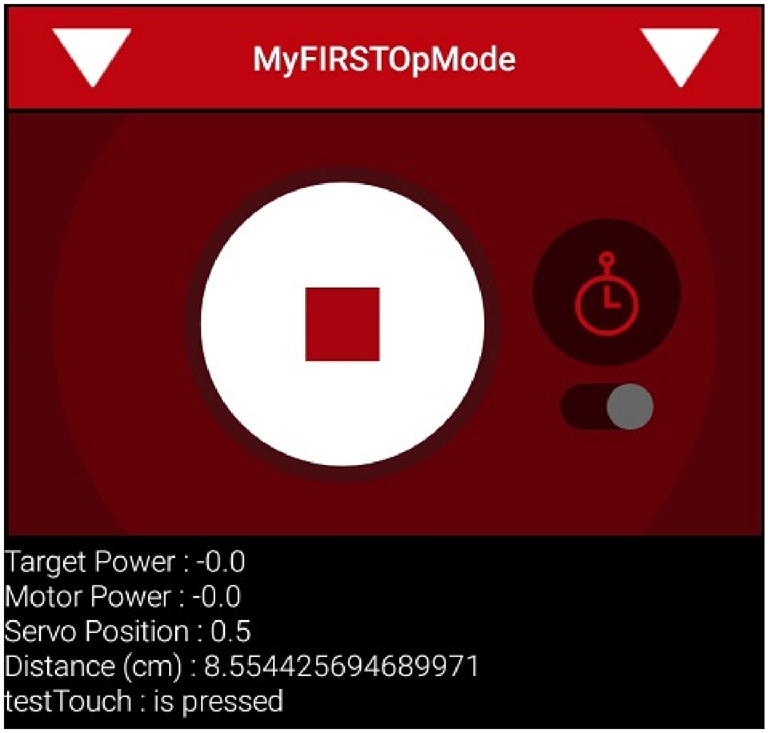
9. Insert another "telemetry.addData" block (the text version) to the "else" clause of the "if do else" block. Change the "key" value to "testTouch" and the "text" value to "is NOT pressed".



10. Save your op mode and verify that it was saved successfully to the Robot Controller.



11. Follow the procedure outlined in the section titled *Running Your OpMode* to run your updated op mode.



As you run the op mode and push or release the button, the telemetry message on the DRIVER STATION should update to reflect the current state of the digital Touch Sensor.

Reference Documents Blocks

Blocks Reference Materials Blocks

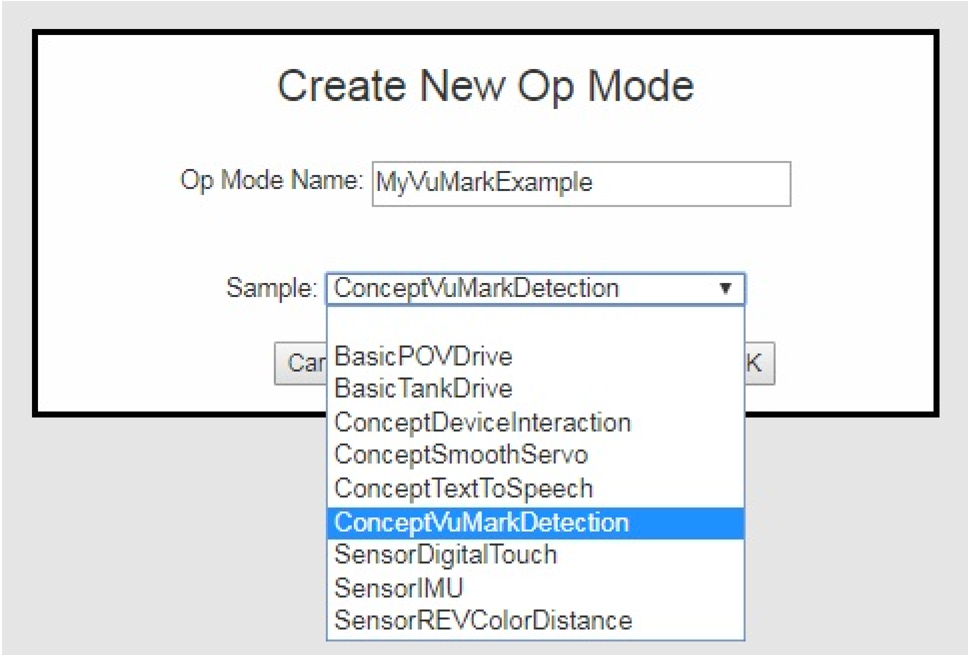
Blocks Reference Manual

As you start to write more complicated op modes, you will need to use more features of the FIRST Tech Challenge software development kit (SDK). Bruce Schafer of the Oregon Robotics Tournament & Outreach Program (ORTOP) created a useful reference document that describes the programming blocks that are available with the Blocks Programming Tool:

Blocks Programming Tool Reference Manual

Sample Op Modes

The Blocks Programming Tool has several built-in example op modes that demonstrate how to do different tasks with the FIRST Tech Challenge control system. As you create a new file, you can use the Sample dropdown list control to display a list of available sample op modes or templates:



Technology Forum

Registered teams can create user accounts on the FIRST Tech Challenge Community forum. Teams can use the forum to ask questions and receive support from the FIRST Tech Challenge community.

The technology forum can be found at the following address:

- https://ftc-community.firstinspires.org

REV Robotics Control Hub Documentation

REV Robotics Control Hub Getting Started Guide

REV Robotics Expansion Hub Documentation

REV Robotics Expansion Hub Getting Started Guide

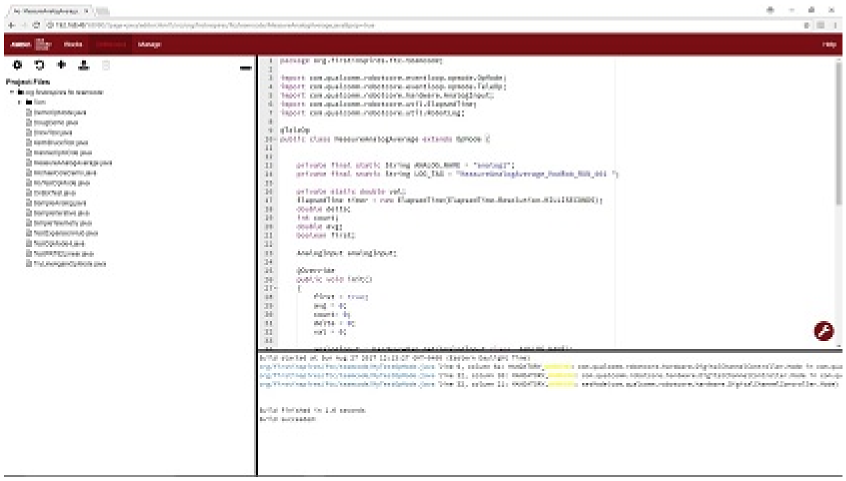
REV Robotics Driver Hub Documentation

REV Robotics Driver Hub Getting Started Guide

21.1.3 OnBot Java Programming Tutorial

This tutorial will take you step-by-step through the process of configuring, programming, and operating your Control System. This tutorial uses the OnBot Java Programming Tool to help you get started programming your robot.

The OnBot Java Programming Tool is a text-based programming tool that lets programmers use a web browser to create, edit and save their Java op modes. This tool is recommended for programmers who have basic to advanced Java skills and who would like to write text-based op modes.



Note: OBJ indicates that the content is specific to OnBot Java Programming

Introduction OBJ

Configuring Your Hardware OBJ

Connecting to the Program & Manage Server OBJ

Writing an Op Mode OBJ

Creating and Running an Op Mode OBJ

The Java Programming Language

This tutorial assumes that you have a sound understanding of the Java programming language. If you do not know Java, then you should consider using the Blocks Programming Tool, which is a visual development tool. Information about the Blocks Programming Tool can be found at the following link:

Blocks Tutorial

Or, you can learn the Java programming language by completing the Oracle Java Tutorial, which is available at the following address:

https://docs.oracle.com/javase/tutorial/

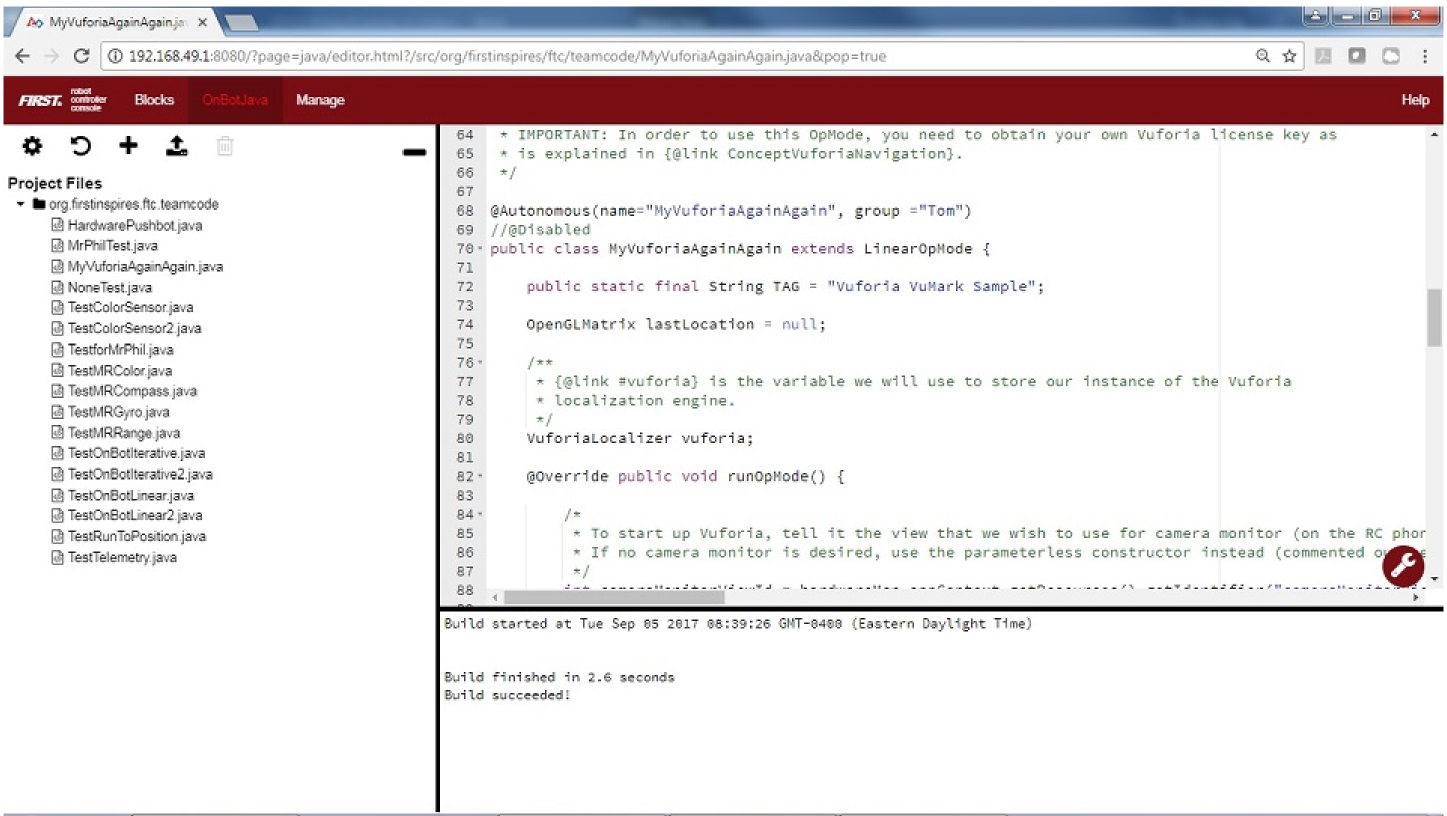
What's an Op Mode?

During a typical FIRST Tech Challenge match, a team's robot must perform a variety of tasks to score points. For example, a team might want their robot to follow a white line on the competition floor and then score a game element into a goal autonomously during a match. Teams write programs called *op modes* (which stands for "operational modes") to specify the behavior for their robot. These op modes run on the Robot Controller phone after being selected on the DRIVER STATION device.

Teams who are participating in the FIRST Tech Challenge have a variety of programming tools that they can use to create their own op modes. This document explains how to use the OnBot Java Programming Tool to write an op mode for a robot.

The OnBot Java Programming Tool

The OnBot Java Programming Tool is a user-friendly programming tool that is served up by the Robot Controller phone. A user can create custom op modes for their robot using this tool and then save these op modes directly onto the Robot Controller. Users write their op modes using Java. The op modes are compiled very quickly on the Robot Controller and then loaded dynamically by the Robot Controller during run time.



The examples in this document use a Windows laptop computer to connect to the Robot Controller. This Windows laptop computer has a Javascript-enabled web browser installed that is used to access the OnBot Java Programming Tool.

Note that the process used to create and edit an op mode is identical if you are using an Android phone as your Robot Controller.

Note that if you prefer, you can use an alternate device, such as an Apple Mac laptop, Chromebook, or an iPad instead of a Windows computer to access the OnBot Java Programming Tool. The instructions included in this document, however, assume that you are using a Windows laptop.

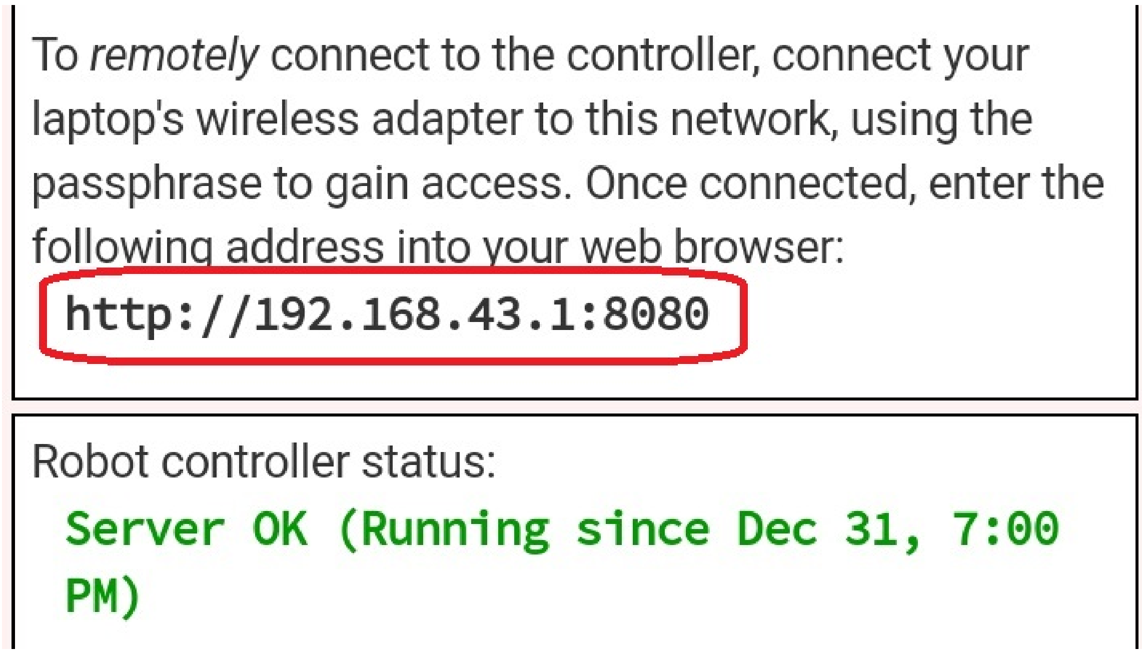
Note that this section of the wiki assumes that you have already setup and configured your Android devices and robot hardware. It also assumes that you have successfully connected your laptop to the Progam & Manage server on the Robot Controller device.

Creating Your First Op Mode

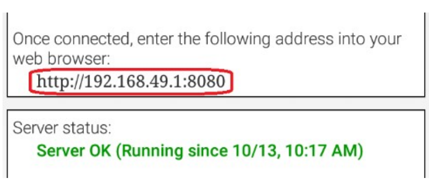
If you connected your laptop successfully to the Program & Manage wireless network of the Robot Controller, then you are ready to create your first op mode. In this section, you will use the OnBot Java Programming Tool to create the program logic for your first op mode.

Creating Your First Op Mode Instructions

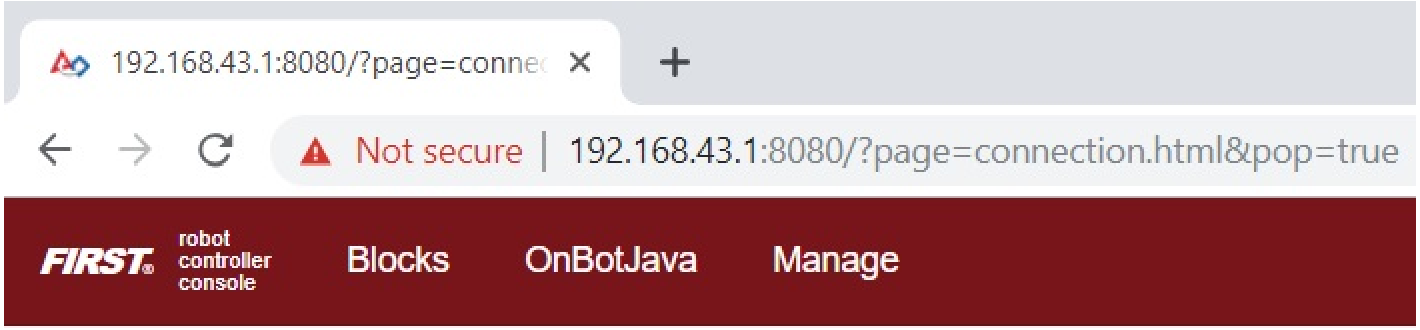
1. Launch the web browser on your laptop (FIRST recommends using Google Chrome) and find the web address that is displayed on the Program & Manage screen of the Robot Controller.



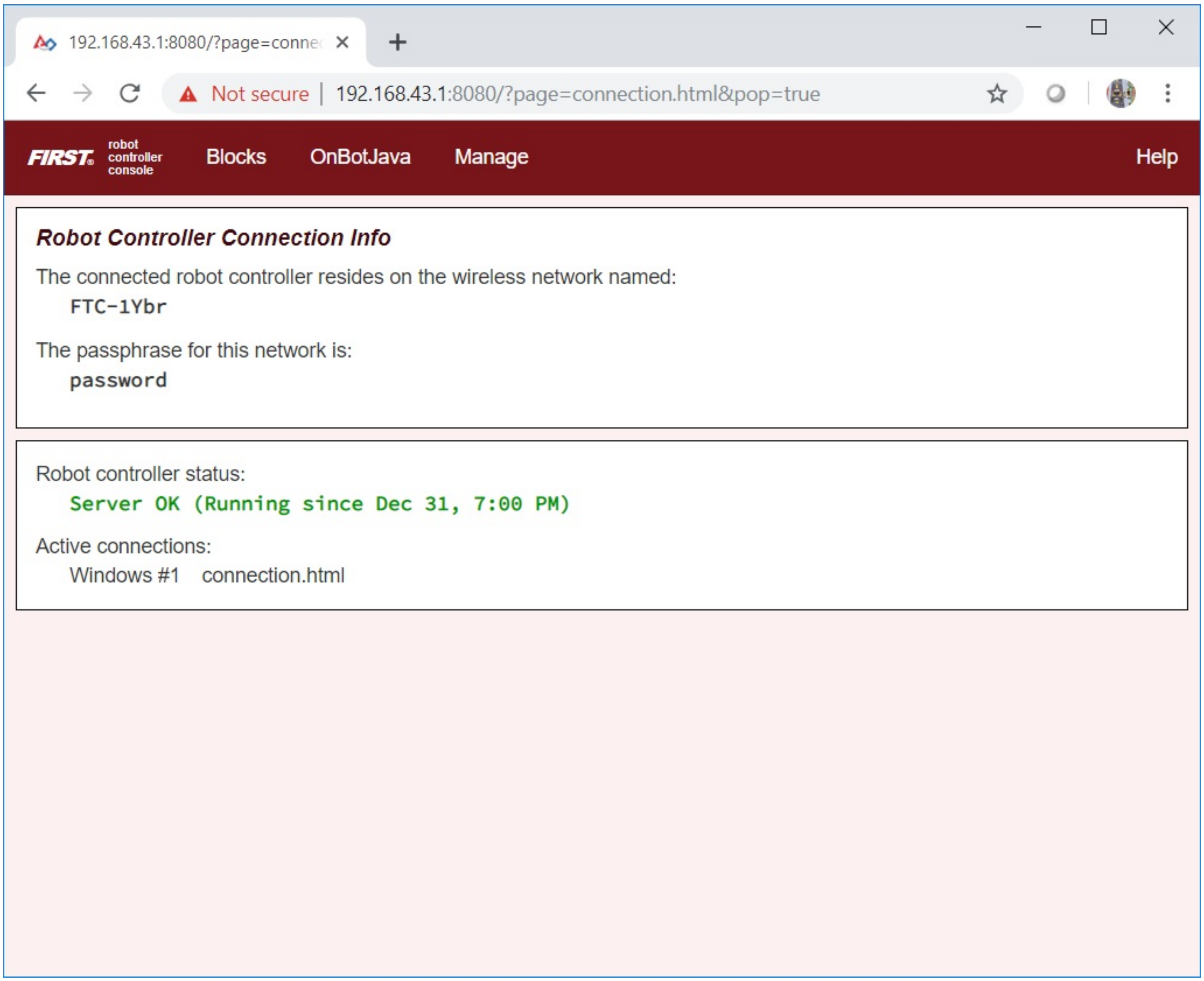
Important: Note: If your Robot Controller is an Android smartphone, then the address to access the Program & Manage server is “192.168.49.1:8080”. Notice the difference in the third octet of the IP addresses (the Control Hub has a “43” instead of a “49”).



Type this web address into the address field of your browser and press RETURN to navigate to the Program & Manage web server.

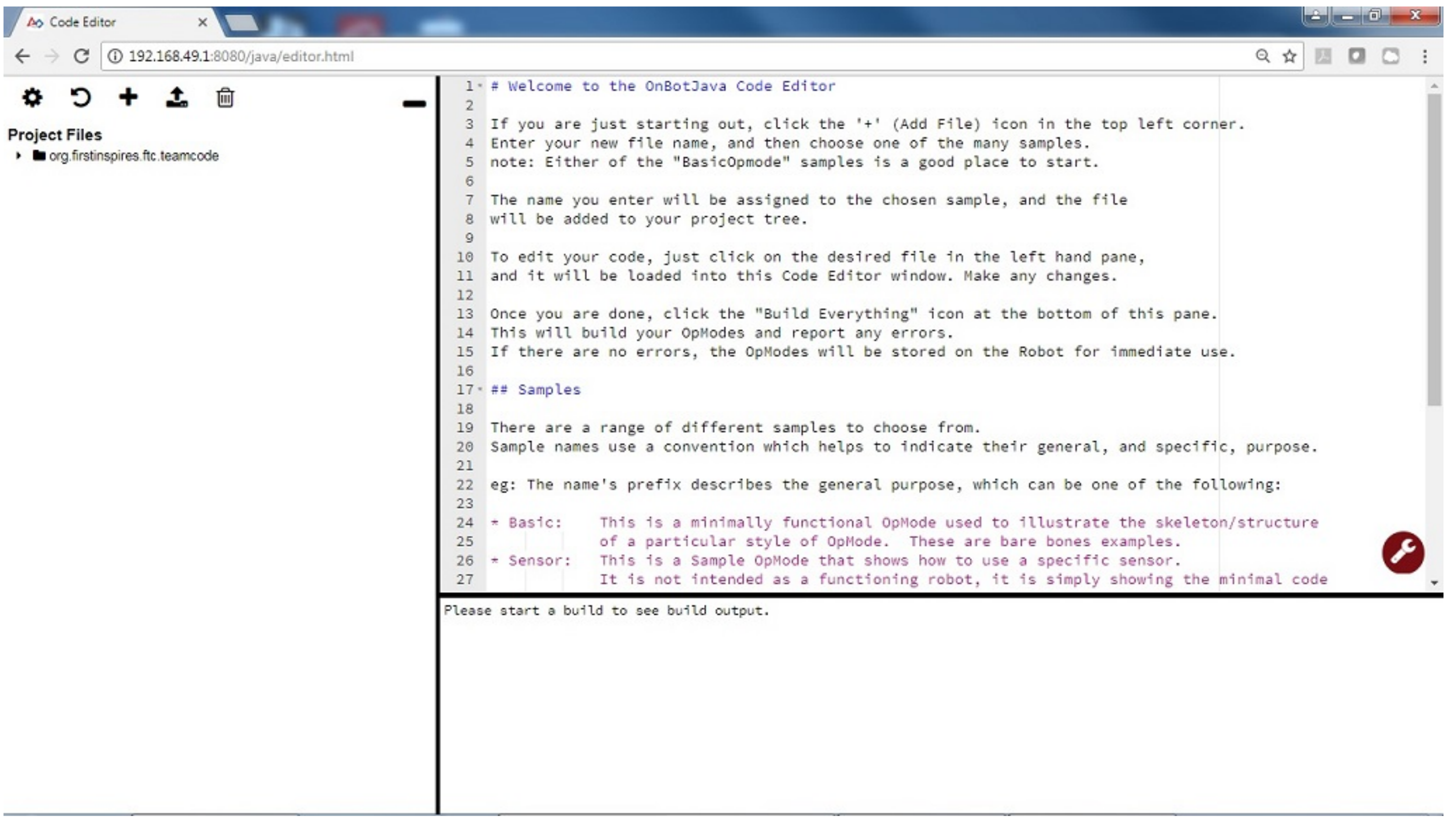


2. Verify that your web browser is connected to the programming mode server. If it is connected to the programming mode server successfully, the Robot Controller Console should be displayed.

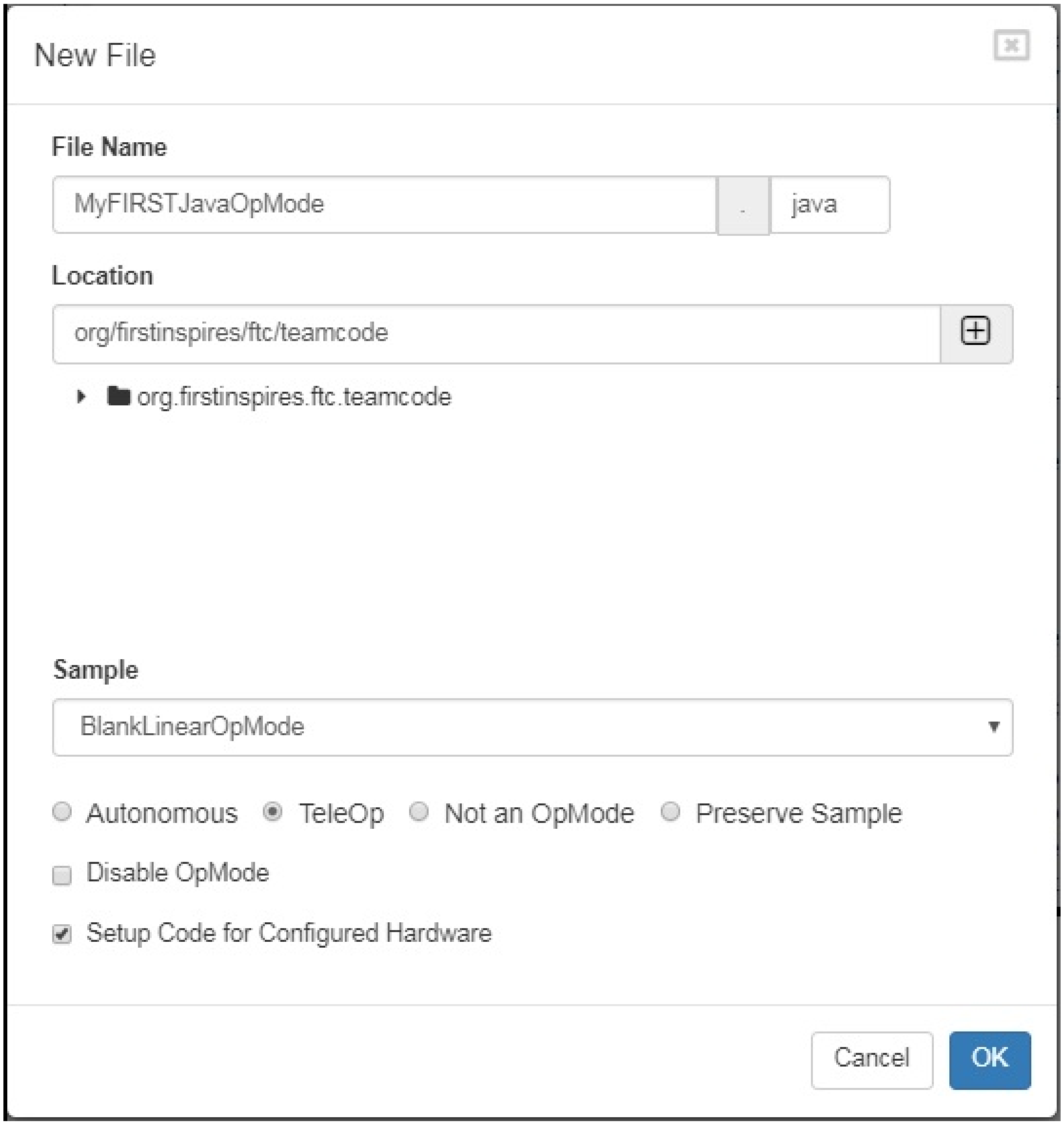


3. Click on the word *OnBotJava* towards the top of the screen. This will switch the browser to OnBot Java Programming mode.

4. Take a look at the OnBot Java user interface. On the left hand side, there is the project browser pane. In the upper right hand corner, there is the source code editing pane. In the lower right hand corner, there is the message pane.



5. In the project browser pane, press the "+" symbol to create a new file. Pushing this button will launch the New File dialog box. This dialog box has several parameters that you can configure to customize your new file.



For this example, specify "MyFIRSTJavaOpMode" as the File Name in the New File dialog box.

Using the Sample dropdown list control, select "BlankLinearOpMode" from the list of available sample op modes (see image above). By selecting "BlankLinearOpMode" the OnBot Java editor will automatically generate a basic LinearOpMode framework for you.

Check the option labeled "TeleOp" to ensure that this new file will be configured as a tele-operated (i.e., driver controlled) op mode.

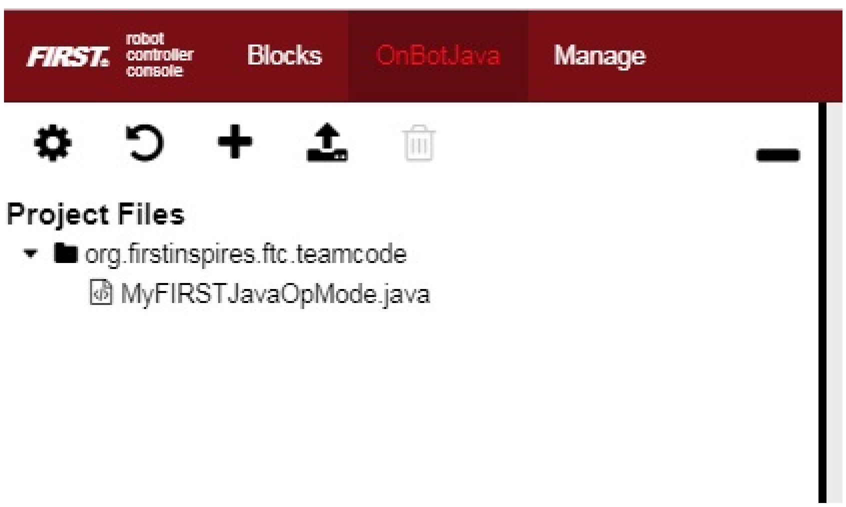
Also, make sure you check the "Setup Code for Configured Hardware" option. If this option is enabled, the OnBot Java editor will look at the hardware configuration file for your Robot Controller and automatically generate the code that you will need to access the configured devices in your op mode.

Press the "OK" button to create your new op mode.

6. You should see your newly created op mode in the editing pane of the OnBot Java user interface.

```
/*
Copyright 2017 FIRST Tech Challenge Team 9999

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and
associated documentation files (the "Software"), to deal in the Software without restriction,
including without limitation the rights to use, copy, modify, merge, publish, distribute,
sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial
portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT
NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
*/
package org.firstinspires.ftc.teamcode;

import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.hardware.Gyroscope;
import com.qualcomm.robotcore.hardware.DigitalChannel;
import com.qualcomm.robotcore.hardware.DistanceSensor;
import com.qualcomm.robotcore.hardware.Servo;
import com.qualcomm.robotcore.eventloop.opmode.TeleOp;
```

Congratulations, you created your first op mode! The op mode currently does not do much, but you will eventually modify it to make it more useful.



Note that when you create an OnBot op mode, you create a .java file that is stored on the Robot Controller. You can access your saved op modes using the project browser on the left side of the screen. You can also organize your saved op modes by right mouse clicking on the project browser to display a list of options to create, edit or delete files and folders.

Also, note that the OnBot Java editor automatically saves your op mode as you are editing it, provided that you are connected to the Program & Manage server.

Examining the Structure of Your Op Mode

It can be helpful to think of an op mode as a list of tasks for the Robot Controller to perform. For a linear op mode, the Robot Controller will process this list of tasks sequentially. Users can also use control loops (such as a while loop) to have the Robot Controller repeat (or iterate) certain tasks within a linear op mode.

If you think about an op mode as a list of instructions for the robot, this set of instructions that you created will be executed by the robot whenever a team member selects the op mode called "MyFIRSTJavaOpMode" from the list of available op modes for this Robot Controller.

Let's look at the structure of your newly created op mode. Here's a copy of the op mode text (minus some comments, the package definition, and some import package statements):

```java
@TeleOp

public class MyFIRSTJavaOpMode extends LinearOpMode {
    private Gyroscope imu;
    private DcMotor motorTest;
    private DigitalChannel digitalTouch;
    private DistanceSensor sensorColorRange;
    private Servo servoTest;


    @Override
    public void runOpMode() {
        imu = hardwareMap.get(Gyroscope.class, "imu");
        motorTest = hardwareMap.get(DcMotor.class, "motorTest");
        digitalTouch = hardwareMap.get(DigitalChannel.class, "digitalTouch");
        sensorColorRange = hardwareMap.get(DistanceSensor.class, "sensorColorRange");
        servoTest = hardwareMap.get(Servo.class, "servoTest");

        telemetry.addData("Status", "Initialized");
        telemetry.update();
        // Wait for the game to start (driver presses PLAY)
        waitForStart();

        // run until the end of the match (driver presses STOP)
        while (opModeIsActive()) {
            telemetry.addData("Status", "Running");
            telemetry.update();

        }
    }
}
```

At the start of the op mode there is an annotation that occurs before the class definition. This annotation states that this is a tele-operated (i.e., driver controlled) op mode:

```java
@TeleOp
```

If you wanted to change this op mode to an autonomous op mode, you would replace the `@TeleOp` with an `@Autonomous` annotation instead.

You can see from the sample code that an op mode is defined as a Java class. In this example, the op mode name is called “MyFIRSTJavaOpMode” and it inherits characteristics from the LinearOpMode class.

```java
public class MyFIRSTJavaOpMode extends LinearOpMode {
```

You can also see that the OnBot Java editor created five private member variables for this op mode. These variables will hold references to the five configured devices that the OnBot Java editor detected in the configuration file of your Robot Controller.

```java
private Gyroscope imu;
private DcMotor motorTest;
private DigitalChannel digitalTouch;
private DistanceSensor sensorColorRange;
private Servo servoTest;
```

Next, there is an overridden method called runOpMode. Every op mode of type LinearOpMode must implement this method. This method gets called when a user selects and runs the op mode.

```java
@Override
public void runOpMode() {
```

At the start of the runOpMode method, the op mode uses an object named hardwareMap to get references to the hardware devices that are listed in the Robot Controller's configuration file:

```
imu = hardwareMap.get(Gyroscope.class, "imu");
motorTest = hardwareMap.get(DcMotor.class, "motorTest");
digitalTouch = hardwareMap.get(DigitalChannel.class, "digitalTouch");
sensorColorRange = hardwareMap.get(DistanceSensor.class, "sensorColorRange");
servoTest = hardwareMap.get(Servo.class, "servoTest");
```

The hardwareMap object is available to use in the runOpMode method. It is an object of type HardwareMap class.

Note that when you attempt to retrieve a reference to a specific device in your op mode, the name that you specify as the second argument of the HardwareMap.get method must match the name used to define the device in your configuration file. For example, if you created a configuration file that had a DC motor named "motorTest", then you must use this same name (it is case sensitive) to retrieve this motor from the hardwareMap object. If the names do not match, the op mode will throw an exception indicating that it cannot find the device.

In the next few statements of the example, the op mode prompts the user to push the start button to continue. It uses another object that is available in the runOpMode method. This object is called telemetry and the op mode uses the addData method to add a message to be sent to the DRIVER STATION. The op mode then calls the update method to send the message to the DRIVER STATION. Then it calls the waitForStart method, to wait until the user pushes the start button on the driver station to begin the op mode run.

```
telemetry.addData("Status", "Initialized");
telemetry.update();
// Wait for the game to start (driver presses PLAY)
waitForStart();
```

Note that all linear op modes should have a waitForStart statement to ensure that the robot will not begin executing the op mode until the driver pushes the start button.

After a start command has been received, the op mode enters a while loop and keeps iterating in this loop until the op mode is no longer active (i.e., until the user pushes the stop button on the DRIVER STATION):

```
// run until the end of the match (driver presses STOP)
while (opModeIsActive()) {
    telemetry.addData("Status", "Running");
    telemetry.update();

}
```

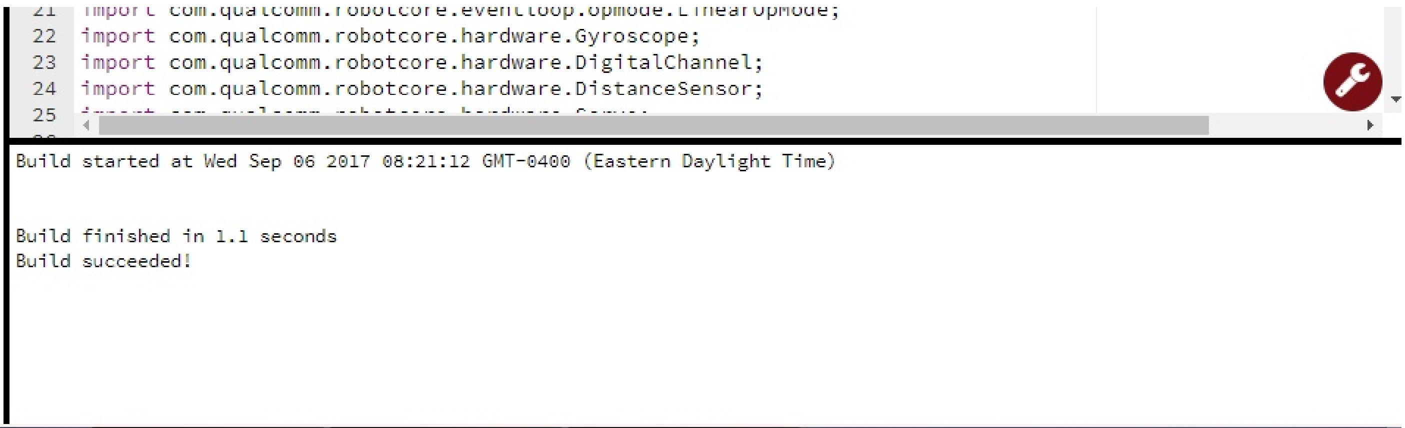
As the op mode iterates in the while loop, it will continue to send telemetry messages with the index of "Status" and the message of "Running" to be displayed on the DRIVER STATION.

Building Your Op Mode

When you create or edit an op mode the OnBot Java editor will auto-save the .java file to the file system of the Robot Controller. However, before you can execute your changes on the Robot Controller, you must first build the op mode and convert it from a Java text file to a binary that can be loaded dynamically into the Robot Controller app.

If you are satisfied with your op mode and are ready to build, press the Build button (which is the button with the wrench symbol, see image below) to start the build process. Note that the build process will build **all of the .java files** on your Robot Controller.

You should see messages appear in the message pane, which is located in the lower right hand side of the window. If your build was successful, you should see a "Build succeeded!" message in the message pane.



Once you have built the binary files with your updated op modes, they are ready to run on the Robot Controller. Before we run our example op mode, let's see what happens if a problem occurs during the build process.

Troubleshooting Build Messages

In the previous section, the build process went smoothly. Let's modify your op mode slightly to cause an error in the build process.

In the editing pane of the OnBot Java window, look for the line that reads `private Servo servoTest;`. This should appear somewhere near the beginning of your op mode class definition. Change the word "Servo" to the word "Zervo":

```
private Zervo servoTest;
```

Also, let's modify the telemetry statement that informs the user that the op mode has been initialized, and let's remove one of the two arguments so that the statement looks like this:

```
telemetry.addData("Status",);
```

Note that when you eliminate the second argument, a little "x" should appear next to the line with the modified addData statement. This "x" indicates that there is a syntax error in the statement.

```
59        sensorColorRange = hardwareMap.get(DistanceSensor.class, "s
60        servoTest = hardwareMap.get(Servo.class, "servoTest");
61
62        telemetry.addData("Status", );
63        telemetry.update();
64        // Wait for the game to start (driver presses PLAY)
65        waitForStart();
66
```

After you have modified your op mode, you can press the build button and see what error messages appear.

```
Build started at Wed Sep 06 2017 09:03:41 GMT-0400 (Eastern Daylight Time)
org/firstinspires/ftc/teamcode/MyFIRSTJavaOpMode.java line 62, column 37: ERROR: illegal start of expression


Build finished in 0.2 seconds
Build FAILED!
```

When you first attempt to build the op mode, you should get an "illegal start of expression error". This is because the addData method is missing its second argument. The OnBot Java system also directs you to the file that has the error, and the location within the file where the error occurs.

In this example, the problem file is called "org/firstinspires/ftc/teamcode/MyFIRSTJavaOpMode.java" and the error occurs at line 62, column 37. It is important to note that the build process builds all of the .java files on the Robot Controller. If there is an error in a different file (one that you are not currently editing) you will need to look at the file name to determine which file is causing the problem.

Let's restore this statement back to its original, correct form:

```
telemetry.addData("Status", "Initialized");
```

After you have corrected the addData statement, push the build button again to see what happens. The OnBot Java system should complain that it cannot find the symbol "Zervo" in a source file called "org/firstinspires/ftc/teamcode/MyFIRSTJavaOpMode.java" at line 51, column 13.

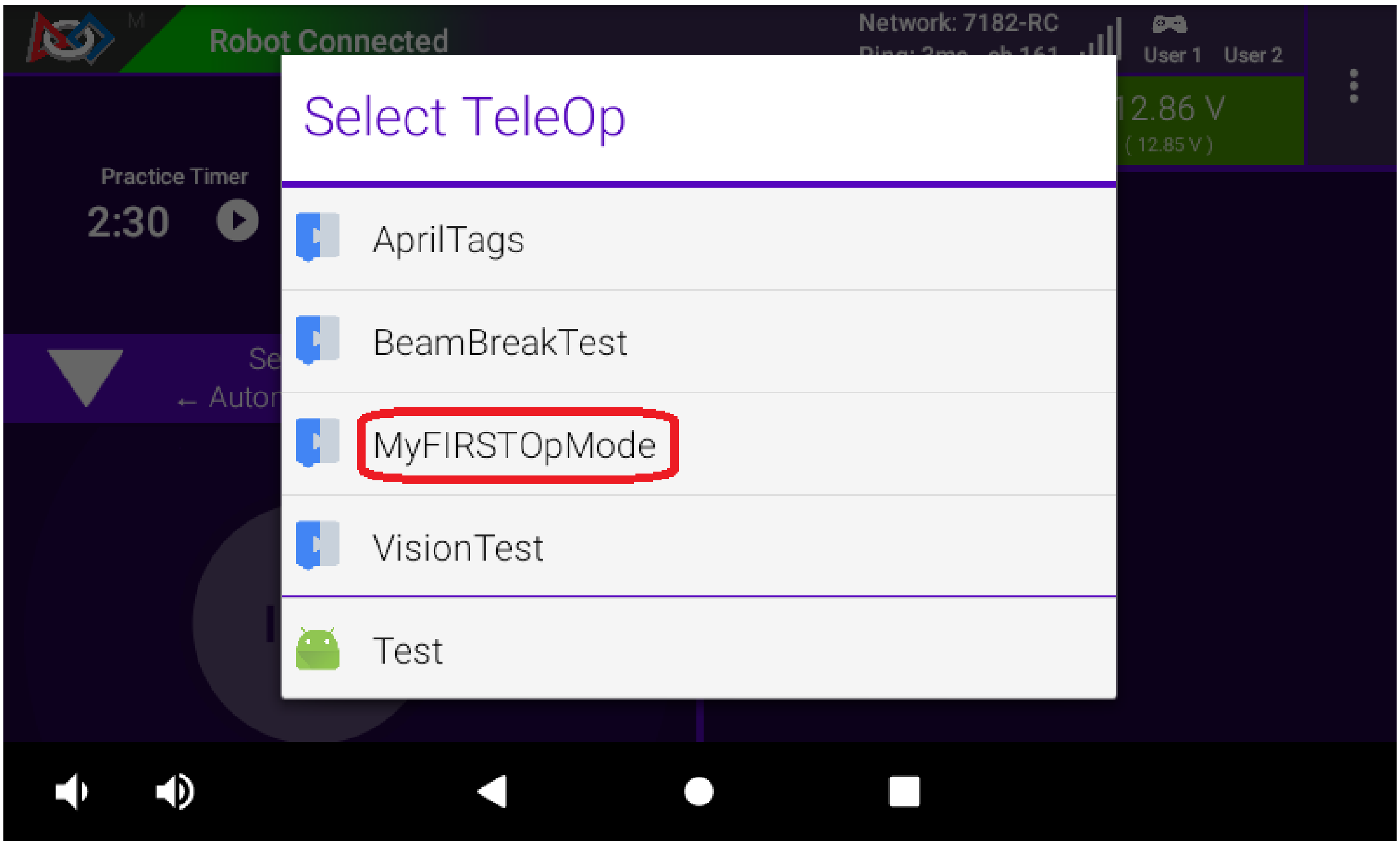
```
Build started at Wed Sep 06 2017 09:10:46 GMT-0400 (Eastern Daylight Time)
org/firstinspires/ftc/teamcode/MyFIRSTJavaOpMode.java line 51, column 13: ERROR: cannot find symbol
  symbol:   class Zervo
  location: class org.firstinspires.ftc.teamcode.MyFIRSTJavaOpMode


Build finished in 0.4 seconds
Build FAILED!
```

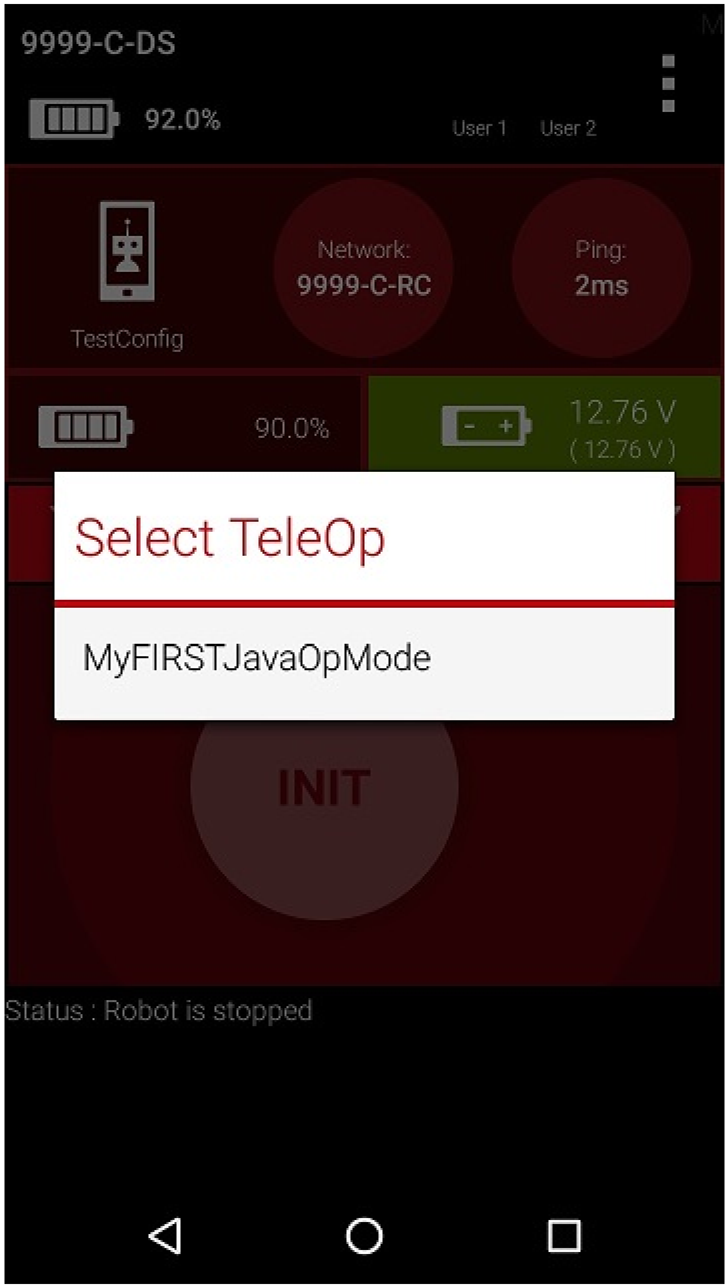
You should restore the statement back to its original form and then push the build button and verify that the op mode gets built properly.

```
private Servo servoTest;
```

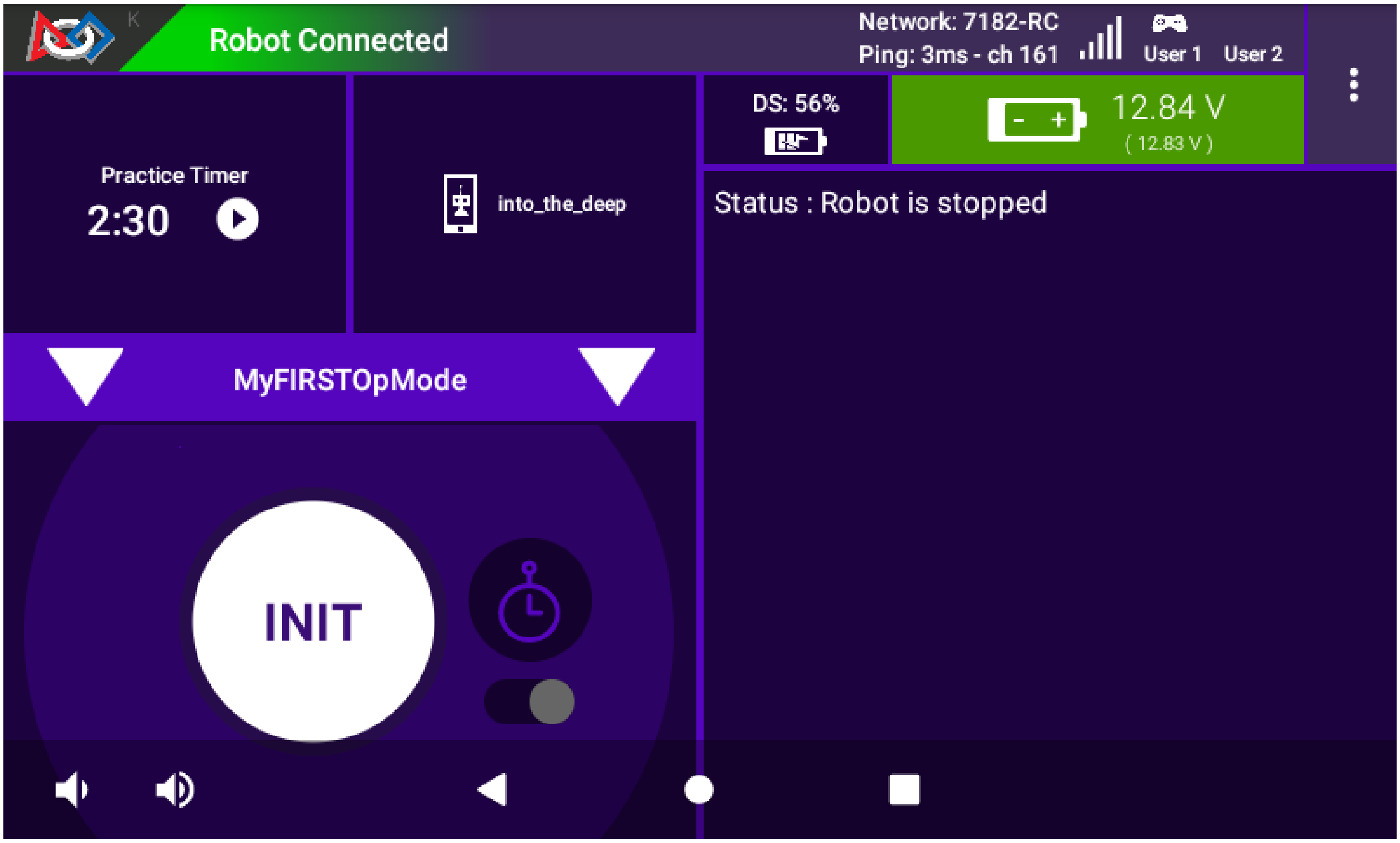
Running Your Op Mode

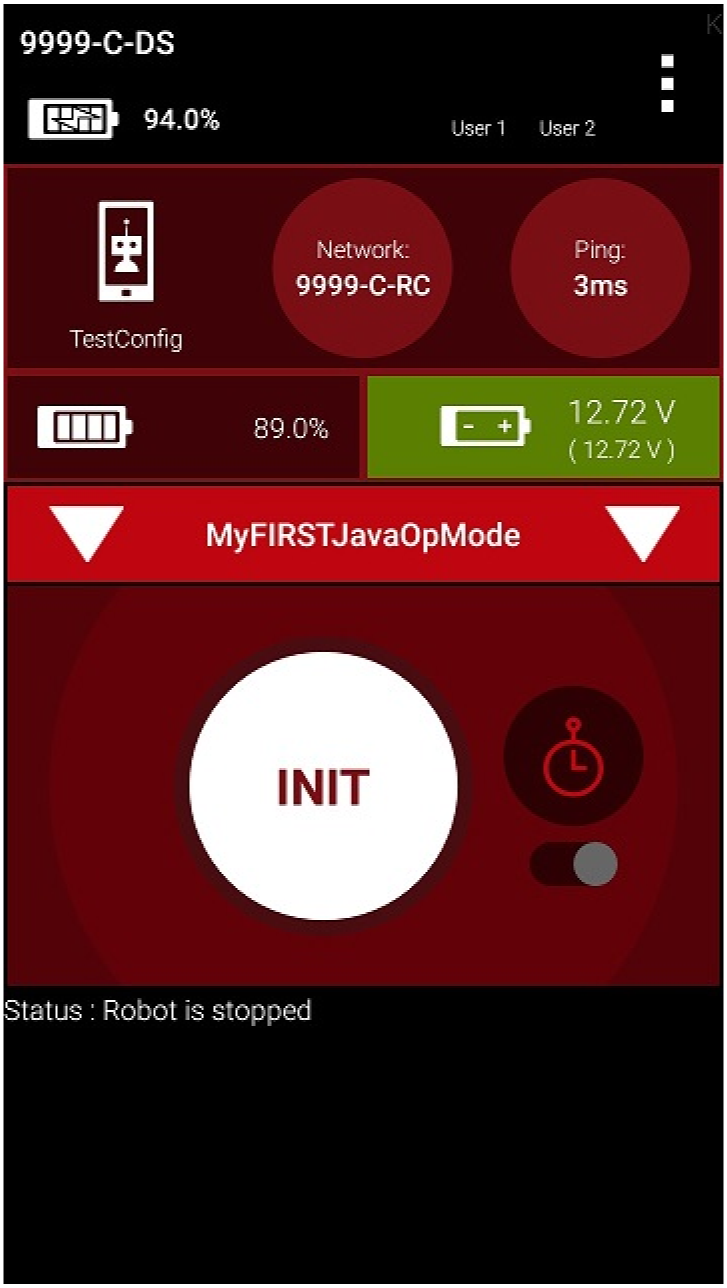
- If you successfully rebuilt your op mode, you are ready to run the op mode. Verify that the DRIVER STATION is still connected to the Robot Controller. Since you designated that your example op mode is a tele-operated op mode, it will be listed as a "TeleOp" op mode.
- On the DRIVER STATION, use the "TeleOp" dropdown list control to display the list of available op modes. Select your op mode ("MyFIRSTJavaOpMode") from the list.



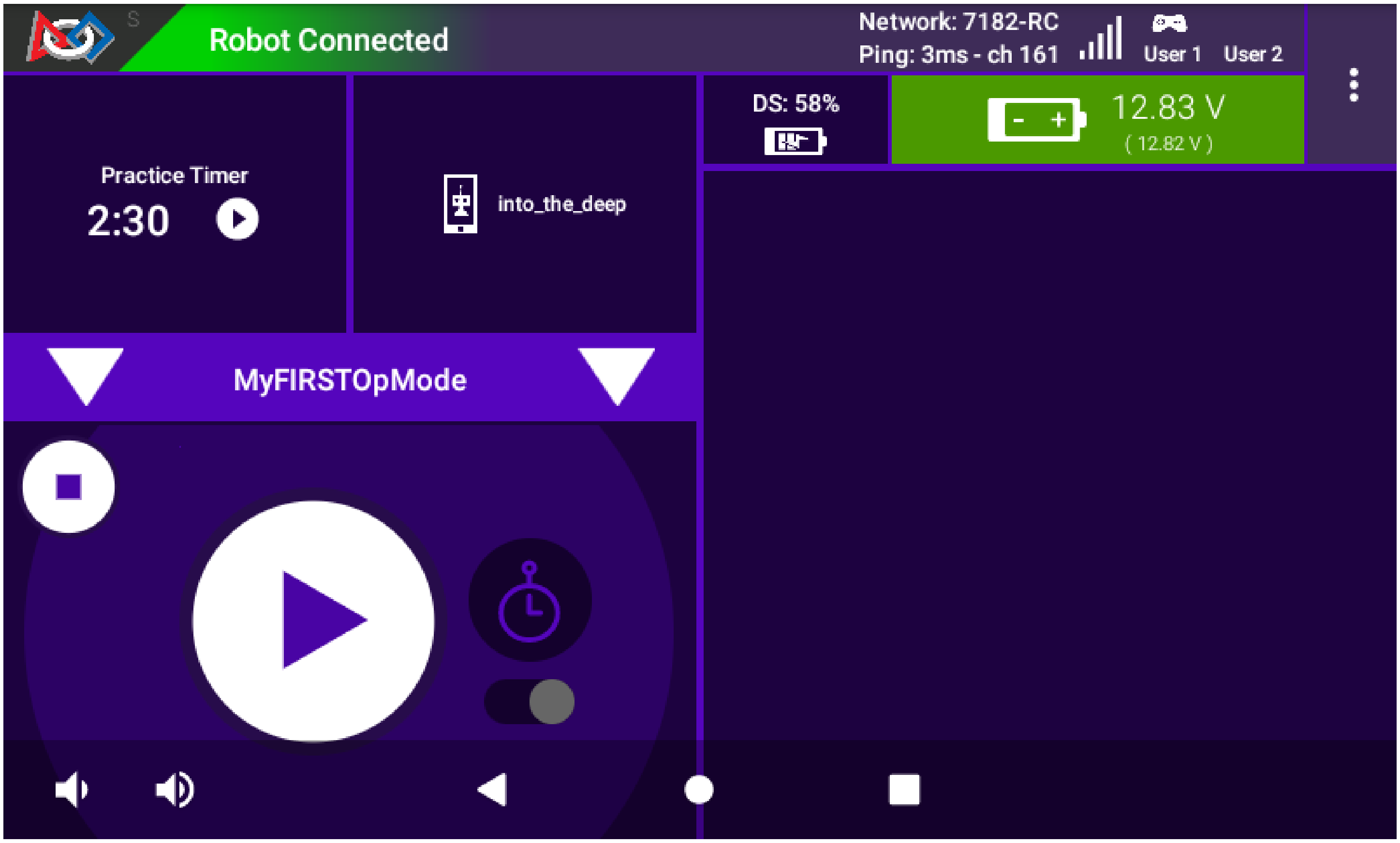


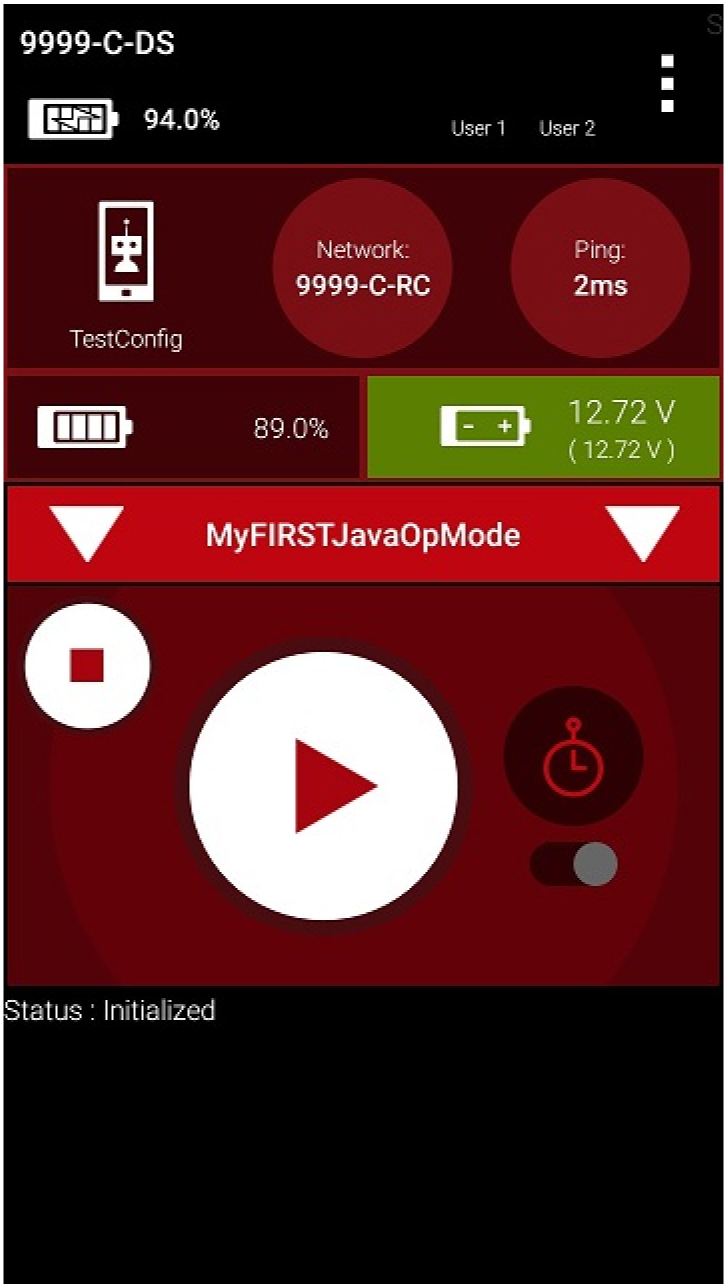
Press the INIT button to initialize the op mode.


Robot Connected
Network: 7182-RC
Ping: 3ms - ch 161
User 1
User 2
DS: 56%
12.84 V
(12.83 V)
Practice Timer
2:30
into_the_deep
Status : Robot is stopped
MyFIRSTOpMode
INIT

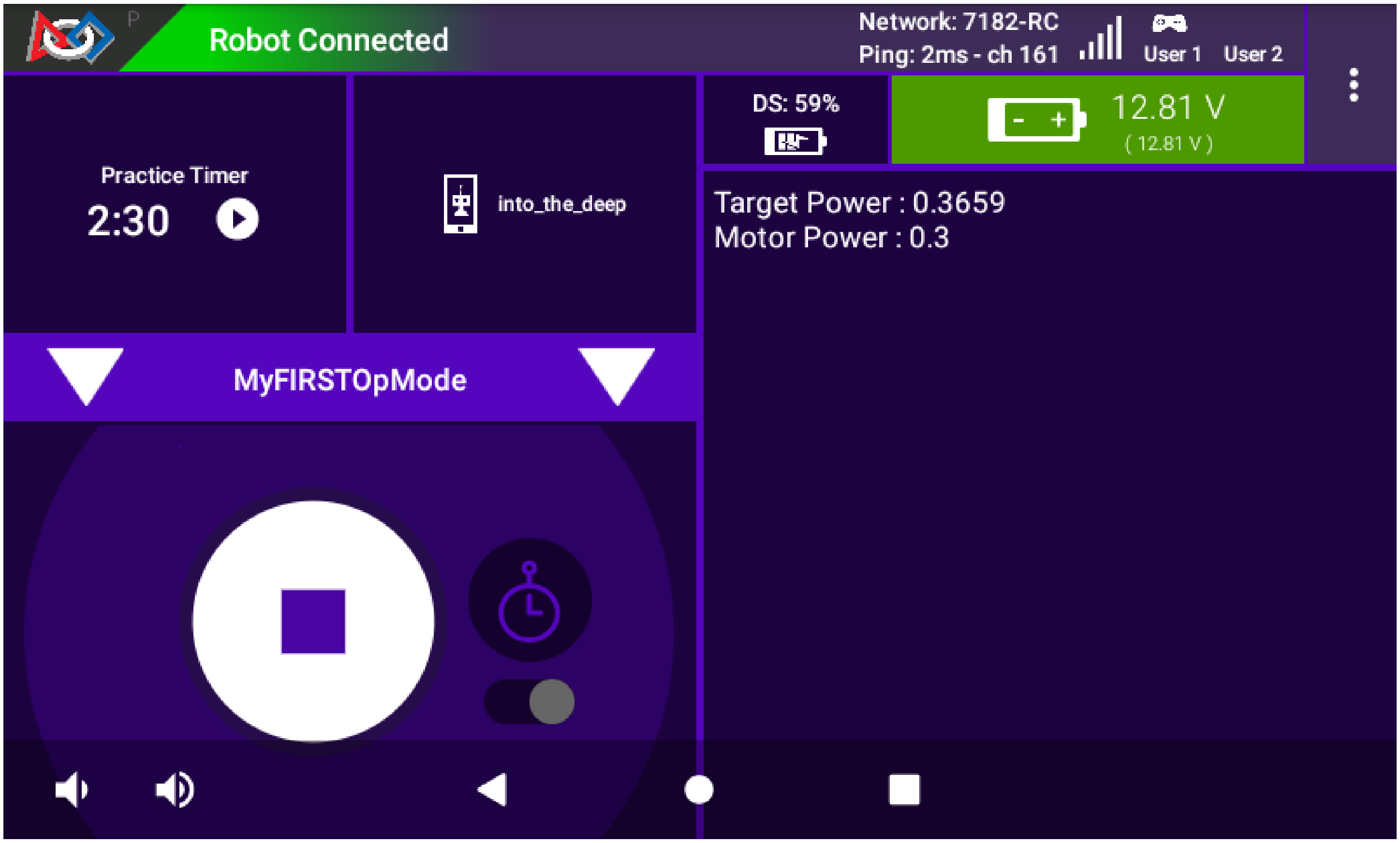


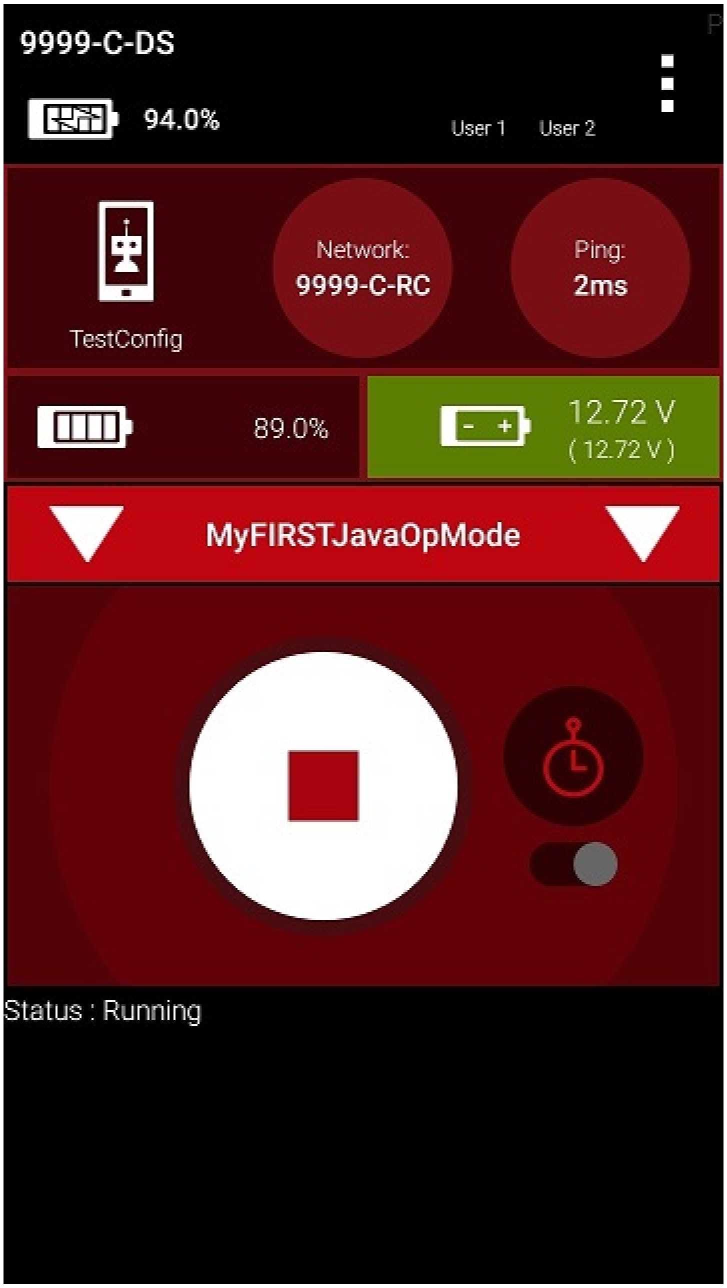
The op mode will execute the statements in the runOpMode method up to the waitForStart statement. It will then wait until you press the start button (which is represented by the triangular shaped symbol) to continue.


Robot Connected
Network: 7182-RC
Ping: 3ms - ch 161
User 1
User 2
DS: 58%
12.83 V
(12.82 V)
Practice Timer
2:30
into_the_deep
MyFIRSTOpMode



Once you press the start button, the op mode will continue to iterate and send the “Status: Running” message to the DRIVER STATION. To stop the op mode, press the square-shaped stop button.


Robot Connected
Network: 7182-RC
Ping: 2ms - ch 161
User 1
User 2
Practice Timer
2:30
into_the_deep
DS: 59%
12.81 V
(12.81 V)
Target Power : 0.3659
Motor Power : 0.3
MyFIRSTOpMode



Congratulations! You ran your first java op mode!

Modifying Your Op Mode to Control a Motor

Let's modify your op mode to control the DC motor that you connected and configured for your REV Expansion Hub. Modify the code for the program loop so that it looks like the following:

```
// run until the end of the match (driver presses STOP)
double tgtPower = 0;
while (opModeIsActive()) {
    tgtPower = -this.gamepad1.left_stick_y;
    motorTest.setPower(tgtPower);
    telemetry.addData("Target Power", tgtPower);
    telemetry.addData("Motor Power", motorTest.getPower());
    telemetry.addData("Status", "Running");
    telemetry.update();

}
```

If you look at the code that was added, you will see that we defined a new variable called target power before we enter the while loop.

```
double tgtPower = 0;
```

At the start of the while loop we set the variable tgtPower equal to the negative value of the gamepad1's left joystick:

```
tgtPower = -this.gamepad1.left_stick_y;
```

The object gamepad1 is available for you to access in the runOpMode method. It represents the state of gamepad #1 in your OPERATOR CONSOLE. Note that for the F310 gamepads that are used during the competition, the Y value of a joystick ranges from -1, when a joystick is in its topmost position, to +1, when a joystick is in its bottommost position. In the example code above, you negate the left_stick_y value so that pushing the left joystick forward will result in a positive power being applied to the motor. Note that in this example, the notion of forwards and backwards for the motor is arbitrary. However, the concept of negating the joystick y value can be very useful in practice.

The next set of statements sets the power of motorTest to the value represented by the variable tgtPower. The values for target power and actual motor power are then added to the set of data that will be sent via the telemetry mechanism to the DRIVER STATION.

```
tgtPower = -this.gamepad1.left_stick_y;
motorTest.setPower(tgtPower);
telemetry.addData("Target Power", tgtPower);
telemetry.addData("Motor Power", motorTest.getPower());
```

After you have modified your op mode to include these new statements, press the build button and verify that the op mode was built successfully.

Running Your Op Mode with a Gamepad Connected

- Your op mode takes input from a gamepad and uses this input to control a DC motor. To run your op mode, you will need to connect a Logitech F310 gamepad to the DRIVER STATION.

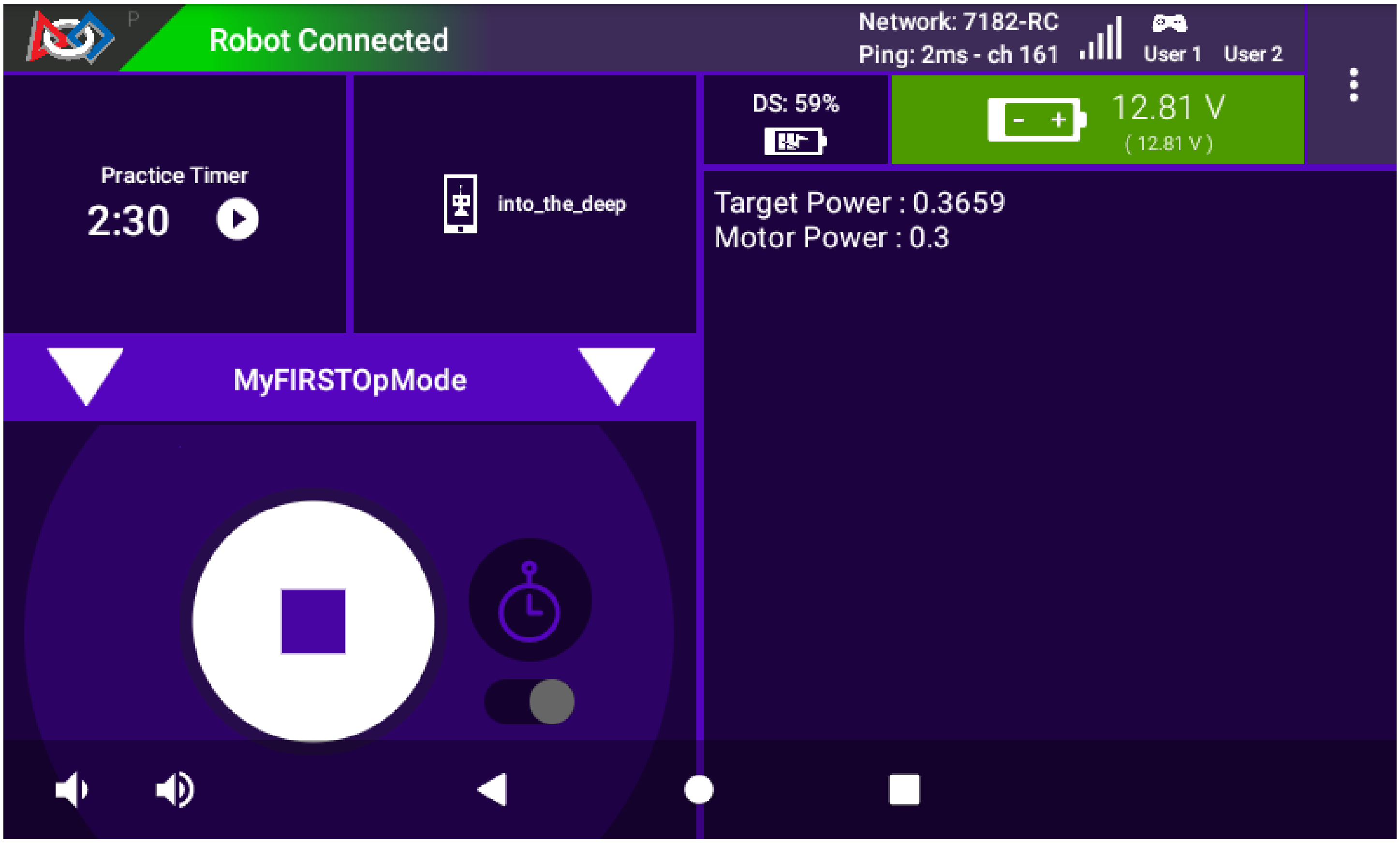
Connect the gamepad to the DRIVER STATION. If using a phone, you will need a Micro USB OTG adapter cable.

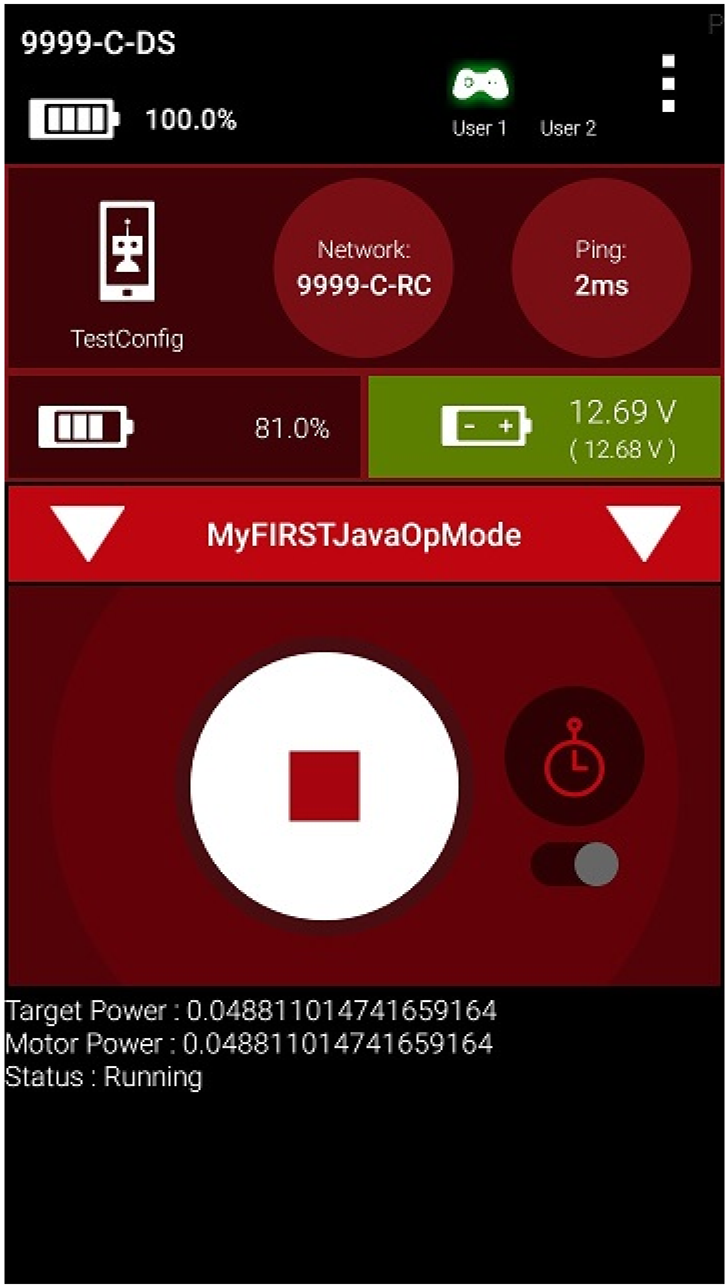
Your example op mode is looking for input from the gamepad designated as the user or driver #1. Press the Start button and the A button simultaneously on the Logictech F310 controller to designate your gamepad as user #1. Note that pushing the Start button and the B button simultaneously would designate the gamepad as user #2.

If you successfully designated the gamepad to be user #1, you should see a little gamepad icon above the text "User 1" in the upper right hand corner of the DRIVER STATION Screen. Whenever there is activity on gamepad #1, the little icon should be highlighted in green. If the icon is missing or if it does not highlight in green when you use your gamepad, then there is a problem with the connection to the gamepad.

Select, initialize and run your "MyFIRSTJavaOpMode" op mode. It is important to note that whenever you rebuild an op mode, you must stop the current op mode run and then restart it before the changes that you just built take effect.

If you configured your gamepad properly, then the left joystick should control the motion of the motor. As you run your op mode, be careful and make sure you do not get anything caught in the turning motor. Note that the User #1 gamepad icon should highlight green each time you move the joystick. Also note that the target power and actual motor power values should be displayed in the telemetry area on the DRIVER STATION.




9999-C-DS
100.0%
User 1
User 2
TestConfig
Network:
9999-C-RC
Ping:
2ms
81.0%
12.69 V
(12.68 V)
MyFIRSTJavaOpMode
Target Power : 0.048811014741659164
Motor Power : 0.048811014741659164
Status : Running

Controlling a Servo OBJ

In this section, you will modify your op mode to control a servo motor with the buttons of the gamepad.

What is a Servo Motor?

A servo motor is a special type of motor. A servo motor is designed for precise motion. A typical servo motor has a limited range of motion.

In the figure below, "standard scale" 180-degree servo is shown. This type of servo is popular with hobbyists and with FIRST Tech Challenge teams. This servo motor can rotate its shaft through a range of 180 degrees. Using an electronic module known as a servo controller you can write an op mode that will move a servo motor to a specific position. Once the motor reaches this target position, it will hold the position, even if external forces are applied to the shaft of the servo.

Servo motors are useful when you want to do precise movements (for example, sweep an area with a sensor to look for a target or move the control surfaces on a remotely controlled airplane).

Modifying Your Op Mode to Control a Servo

Let's modify your op mode to add the logic required to control a servo motor. For this example, you will use the buttons on the Logitech F310 gamepad to control the position of the servo motor.

With a typical servo, you can specify a target position for the servo. The servo will turn its motor shaft to move to the target position, and then maintain that position, even if moderate forces are applied to try and disturb its position.

For the FIRST Tech Challenge control system, you can specify a target position that ranges from 0 to 1 for a servo. A target position of 0 corresponds to zero degrees of rotation and a target position of 1 corresponds to 180 degrees of rotation for a typical servo motor.

In this example, you will use the colored buttons on the right side of the F310 controller to control the position of the servo. Initially, the op mode will move the servo to the midway position (90 degrees of its 180-degree range). Pushing the yellow "Y" button will move the servo to the zero-degree position. Pushing the blue "X" button or the red "B" button will move the servo to the 90-degree position. Pushing the green "A" button will move the servo to the 180-degree position.

Modify your op mode to add the following code:

```
// run until the end of the match (driver presses STOP)
double tgtPower = 0;
while (opModeIsActive()) {
    tgtPower = -this.gamepad1.left_stick_y;
    motorTest.setPower(tgtPower);
    // check to see if we need to move the servo.
    if(gamepad1.y) {
        // move to 0 degrees.
        servoTest.setPosition(0);
    } else if (gamepad1.x || gamepad1.b) {
        // move to 90 degrees.
        servoTest.setPosition(0.5);
    } else if (gamepad1.a) {
        // move to 180 degrees.
        servoTest.setPosition(1);
    }
    telemetry.addData("Servo Position", servoTest.getPosition());
    telemetry.addData("Target Power", tgtPower);
```

(continues on next page)

(continued from previous page)

```
    telemetry.addData("Motor Power", motorTest.getPower());
    telemetry.addData("Status", "Running");
    telemetry.update();

}
```

This added code will check to see if any of the colored buttons on the F310 gamepad are pressed. If the Y button is pressed, it will move the servo to the 0-degree position. If either the X button or B button is pressed, it will move the servo to the 90-degree position. If the A button is pressed, it will move the servo to the 180-degree position. The op mode will also send telemetry data on the servo position to the Driver Station.

After you have modified your op mode, you can build it and then run it. Verify that gamepad #1 is still configured and then use the colored buttons to move the position of the servo.

Using Sensors OBJ

Color-Distance Sensor

A sensor is a device that lets the Robot Controller get information about its environment. In this example, you will use a REV Robotics Color-Distance sensor to display range (distance from an object) info to the driver station.

The Color-Range sensor uses reflected light to determine the distance from the sensor to the target object. It can be used to measure close distances (up 5" or more) with reasonable accuracy. Note that at the time this document was most recently edited, the REV Color-Range sensor saturates around 2" (5cm). This means that for distances less than or equal to 2", the sensor returns a measured distance equal to 2" or so.

Modify your op mode to add a telemetry statement that will send the distance information (in centimeters) to the Driver Station.

```
telemetry.addData("Servo Position", servoTest.getPosition());
telemetry.addData("Target Power", tgtPower);
telemetry.addData("Motor Power", motorTest.getPower());
telemetry.addData("Distance (cm)", sensorColorRange.getDistance(DistanceUnit.CM));
telemetry.addData("Status", "Running");
telemetry.update();
```

After you have modified your op mode, push the build button, then run the op mode to verify that it now displays distance on your Driver Station. Note that if the distance reads "NaN" (short for "Not a Number") it probably means that your sensor is too far from the target (zero reflection). Also note that the sensor saturates at around 5 cm.

Touch Sensor

The REV Robotics Touch Sensor can be connected to a digital port on the Control Hub or Expansion Hub. The Touch Sensor is HIGH (returns TRUE) when it is not pressed. It is pulled LOW (returns FALSE) when it is pressed.

The Control Hub or Expansion Hub digital ports contain two digital pins per port. When you use a 4-wire JST cable to connect a REV Robotics Touch sensor to a Control Hub or Expansion Hub digital port, the Touch Sensor is wired to the second of the two digital pins within the port. The first digital pin of the 4-wire cable remains disconnected.

For example, if you connect a Touch Sensor to the "0,1" digital port of the Control Hub or Expansion Hub, the Touch Sensor will be connected to the second pin (labeled "1") of the port. The first pin (labeled "0") will stay disconnected.

Modify the code in your op mode that occurs before the waitForStart command to set the digital channel for input mode.

```
// set digital channel to input mode.
digitalTouch.setMode(DigitalChannel.Mode.INPUT);

telemetry.addData("Status", "Initialized");
telemetry.update();
// Wait for the game to start (driver presses PLAY)
waitForStart();
```

Also, modify the code in your while loop to add an if-else statement that checks the state of the digital input channel. If the channel is LOW (false), the touch sensor button is pressed and being pulled LOW to ground. Otherwise, the touch sensor button is not pressed.

```
// is button pressed?
if (digitalTouch.getState() == false) {
    // button is pressed.
    telemetry.addData("Button", "PRESSED");
} else {
    // button is not pressed.
    telemetry.addData("Button", "NOT PRESSED");
}

telemetry.addData("Status", "Running");
telemetry.update();
```

Rebuild your op mode, then reinitialize and restart your op mode. The op mode should now display the state of the button ("PRESSED" or "NOT PRESSED").

Reference Documentation OBJ

OnBot Java Reference Info OBJ

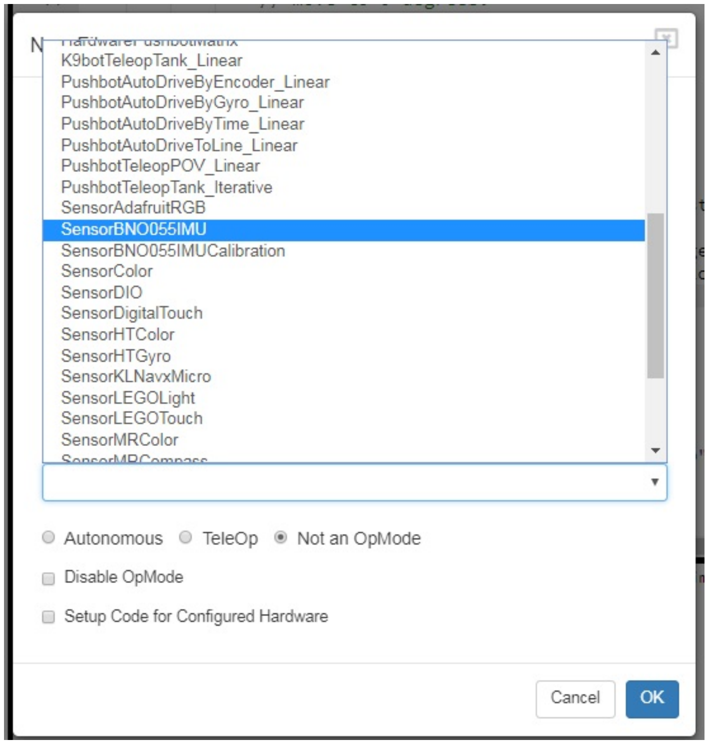
Javadoc Reference Pages

As you start to write more complicated op modes, you will need to use more features of the FIRST Tech Challenge software development kit (SDK). You can reference online Javadoc material that provide descriptions of the available FIRST Tech Challenge-related classes and methods, at the following web address:

https://javadoc.io/doc/org.firstinspires.ftc

Sample Op Modes

The OnBot Java Programming Tool has several built-in example op modes that demonstrate how to do different tasks with the FIRST Tech Challenge control system. As you create a new file, you can use the Sample dropdown list control to display a list of available sample op modes or templates. The comments in these examples help explain what the program statements do.



Technology Forum

Registered teams can create user accounts on the FIRST Tech Challenge forum. Teams can use the forum to ask questions and receive support from the FIRST Tech Challenge community.

The technology forum can be found at the following address:

- https://ftc-community.firstinspires.org

REV Robotics Control Hub Documentation

REV Robotics Control Hub Getting Started Guide

REV Robotics Expansion Hub Documentation

REV Robotics Expansion Hub Getting Started Guide

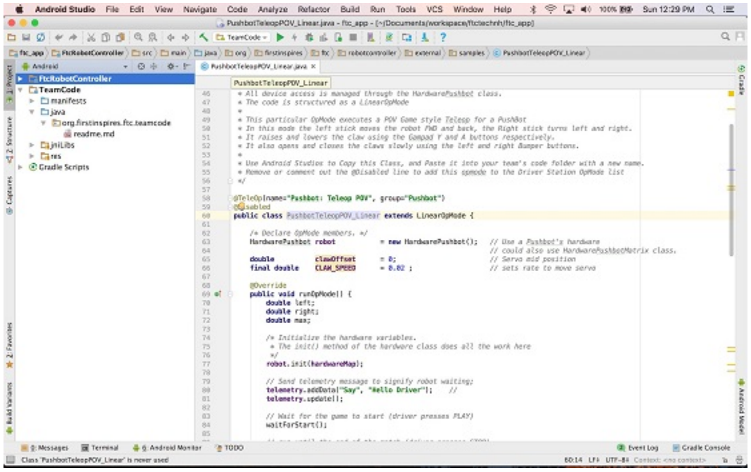
REV Robotics Driver Hub Documentation

REV Robotics Driver Hub Getting Started Guide

21.1.4 Android Studio Programming Tutorial

This tutorial will take you step-by-step through the process of configuring, programming, and operating your Control System. This tutorial uses Android Studio to help you get started programming your robot.

Android Studio is an advanced integrated development environment for creating Android apps. This tool is the same tool that professional Android app developers use. Android Studio is only recommended for **advanced users** who have **extensive Java programming experience**.



Note: AS indicates that the content is specific to Android Studio Programming

Introduction AS

Configuring your Hardware AS

Installing Android Studio AS

Installing Android Studio AS

Android Developer Website

Android Studio is distributed freely by Google, and the most up-to-date reference for installing and using the Android Studio software can be found on the Android developer website:

- https://developer.android.com/studio

Android Studio is available on the Windows, MacOS, and Linux operating systems.

System Requirements

Before you download and install the Android Studio you should first check the list of system requirements on the Android developer's website to verify that your system satisfies the list of minimum requirements:

- Windows
- MacOS
- Linux

Caution: With the introduction of **Android Studio Ladybug**, the JDK that is packaged with Android Studio is incompatible with the FtcRobotController workspace. If you install or update an existing installation to Android Studio Ladybug, you will need to install JDK 17 separately.

Upon initial load of the FtcRobotController workspace using Android Studio Ladybug, an error will be displayed during the Gradle sync and Android Studio will recommend that you upgrade Gradle. Do not upgrade Gradle.

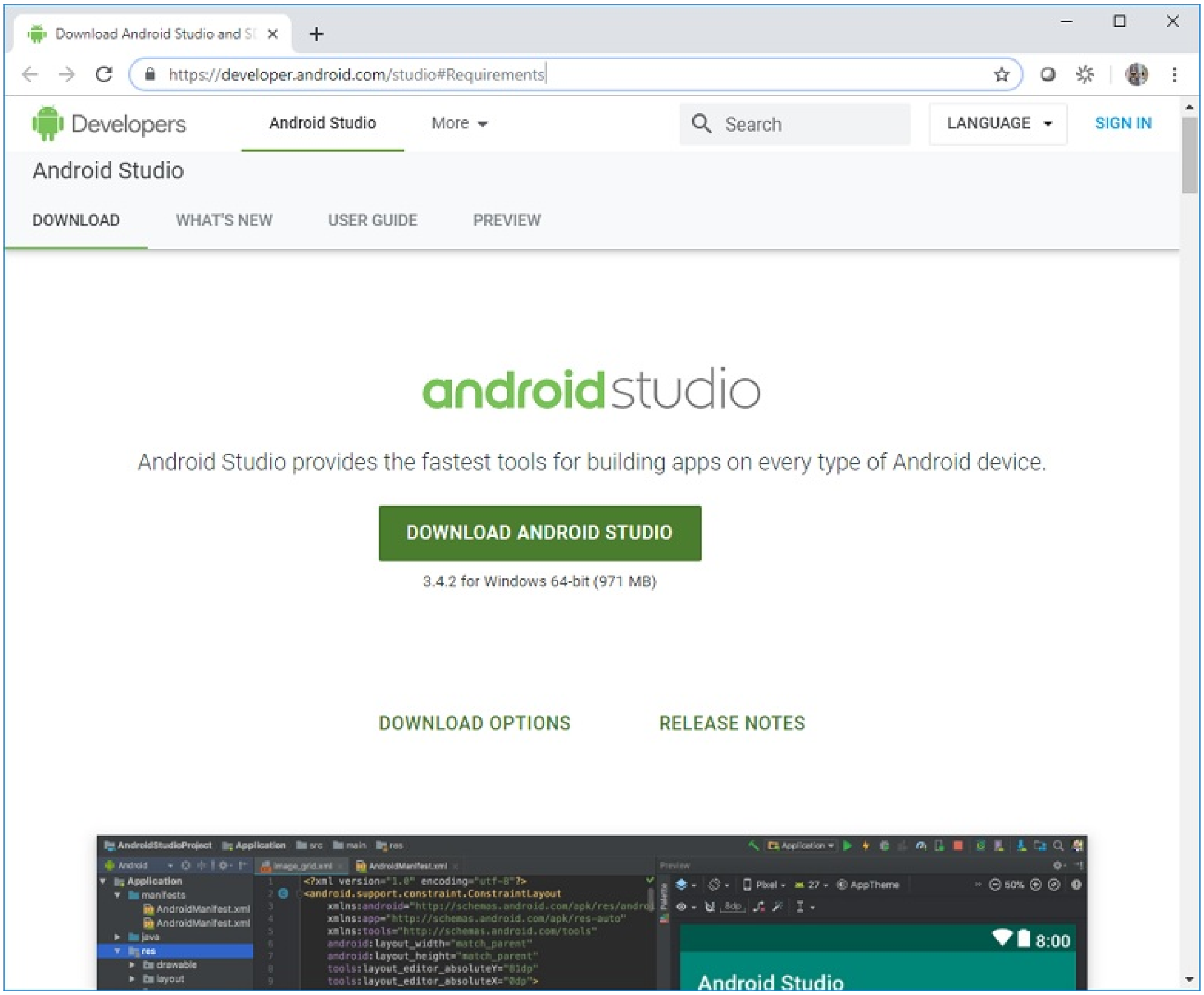
For more detailed instructions see: Configuring

Downloading and Installing Android Studio

Once you have verified that your laptop satisfies the minimum system requirements, you can go to the Android developer's website to download and install Android Studio:

- https://developer.android.com/studio

Click on the green "DOWNLOAD ANDROID STUDIO" button to start the download process.



Accept the license terms and then push the blue “DOWNLOAD ANDROID STUDIO” button on the Android Developer webpage to download the software.

Once the setup package has downloaded, launch the application and follow the on-screen instructions to install Android Studio.

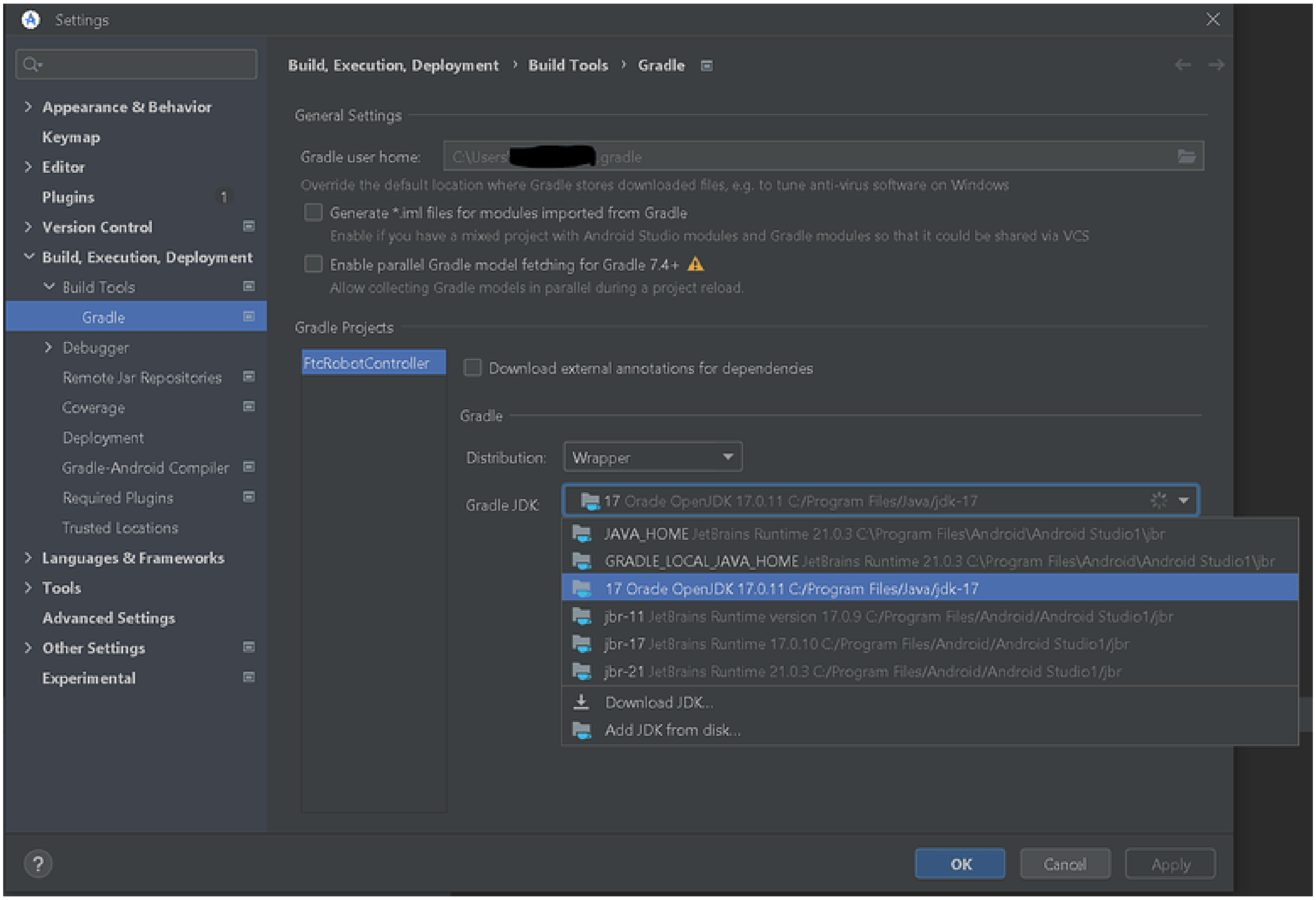
Configuring Android Studio (Ladybug and later)

Note: See the Caution above for why this is necessary.

Note: Android Studio Ladybug updates the underlying JetBrains IntelliJ version such that the interface is a VSCode look alike. The screenshots in this documentation use the JetBrains/Android Studio Classic UI which is no longer supported natively by JetBrains. To follow along, users should install the Classic UI plugin.

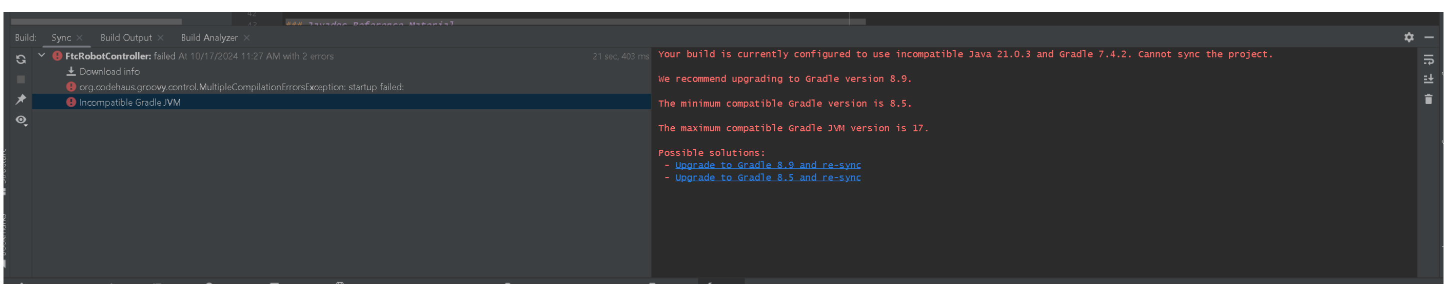
---

1. Install JDK 17 If you did not already have this installed independently of Android Studio. e.g. If you were using Android Studio's bundled JDK, then when Ladybug is installed Android Studio will unhelpfully overwrite your old bundled JDK version. Note there's a bug in the Settings □ Build Tools □ Gradle dialog that may make you think your old version of the JDK is there, but it is not. You must use an unbundled version of the JDK.
2. Go to File -> Settings and under Build, Execution, Deployment -> Build Tools -> Gradle use the Add JDK from disk option to select the newly installed JDK 17. In the image below take careful note of the directory paths for the options labeled jbr-17 and jbr-21. Note that they are the same. This is the aforementioned UI bug, and that is Android Studio overwriting your old JDK. In this image you'll see I've selected the JDK that was installed independently.



Do Not Upgrade Gradle

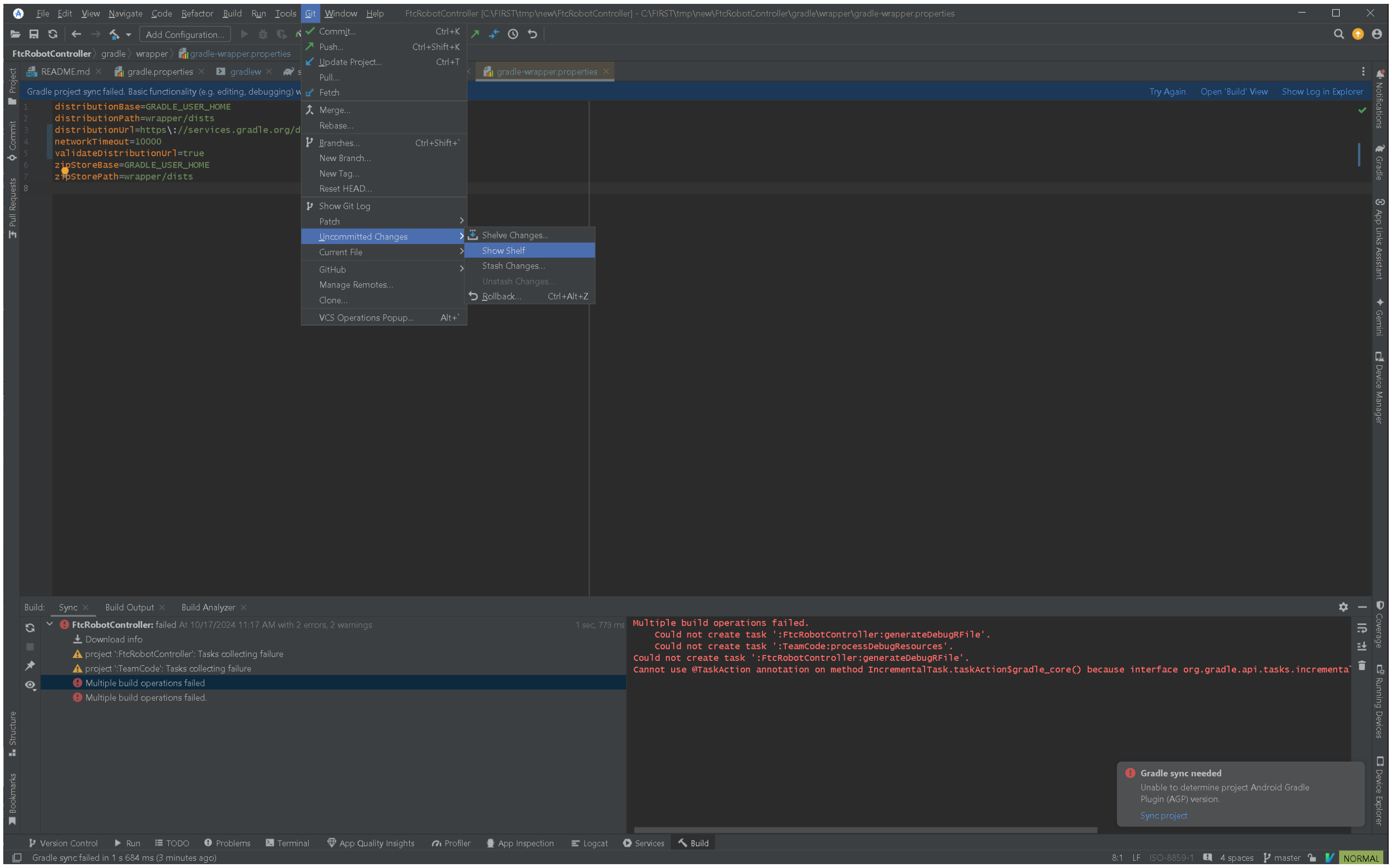
If you have upgraded Android Studio from an earlier version to Ladybug, or you did not install and configure the JDK prior to loading a FtcRobotController workspace, then Android Studio may present an error and recommend that you upgrade Gradle.



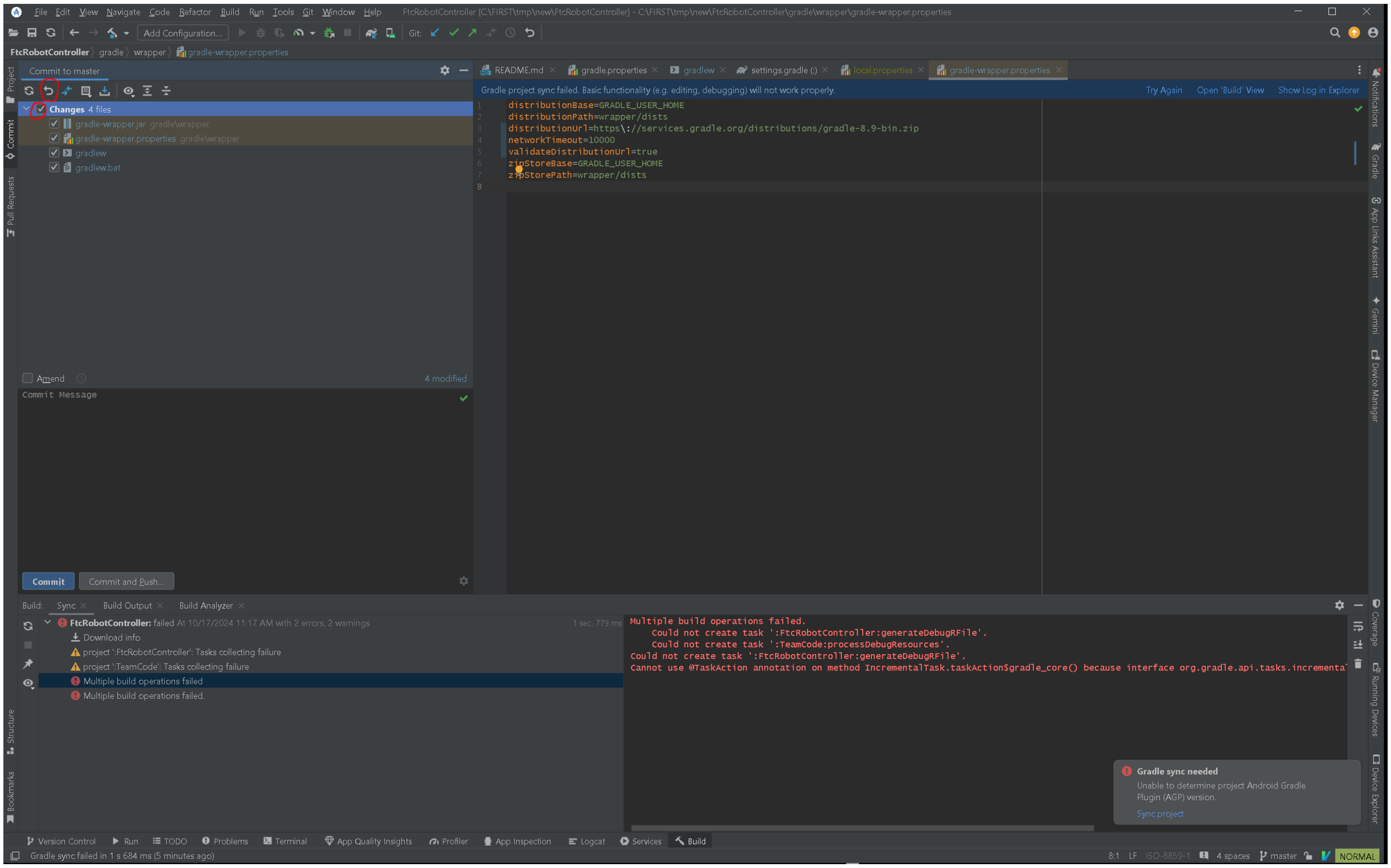
Do not do this. The FtcRobotController build is incompatible with upgraded Gradle. If you do, you will presented with another, even more, indecipherable error.

Gracious Professionalism® - **"Doing your best work while treating others with respect and kindness - It's what makes FIRST, first."**

To recover, you need to rollback the changes that Android Studio made upon that click. To do that select Git -> Uncommitted Changes -> Show Shelf



That will show the changes you have in your workspace. You want to rollback the 4 gradle files shown in the following image. You can either select the Changes checkbox to select all files, or individually select the gradle files. Note that if you have changes in your workspace that haven't been committed, you want to be careful not to select those files or you may lose work.



Once you have the proper files selected, click the Rollback button.

Resync and that should revert you to the error that prompted you to upgrade Gradle in the first place. From there follow the instructions above to install JDK 17.

Downloading the Android Studio Project Folder AS Legacy

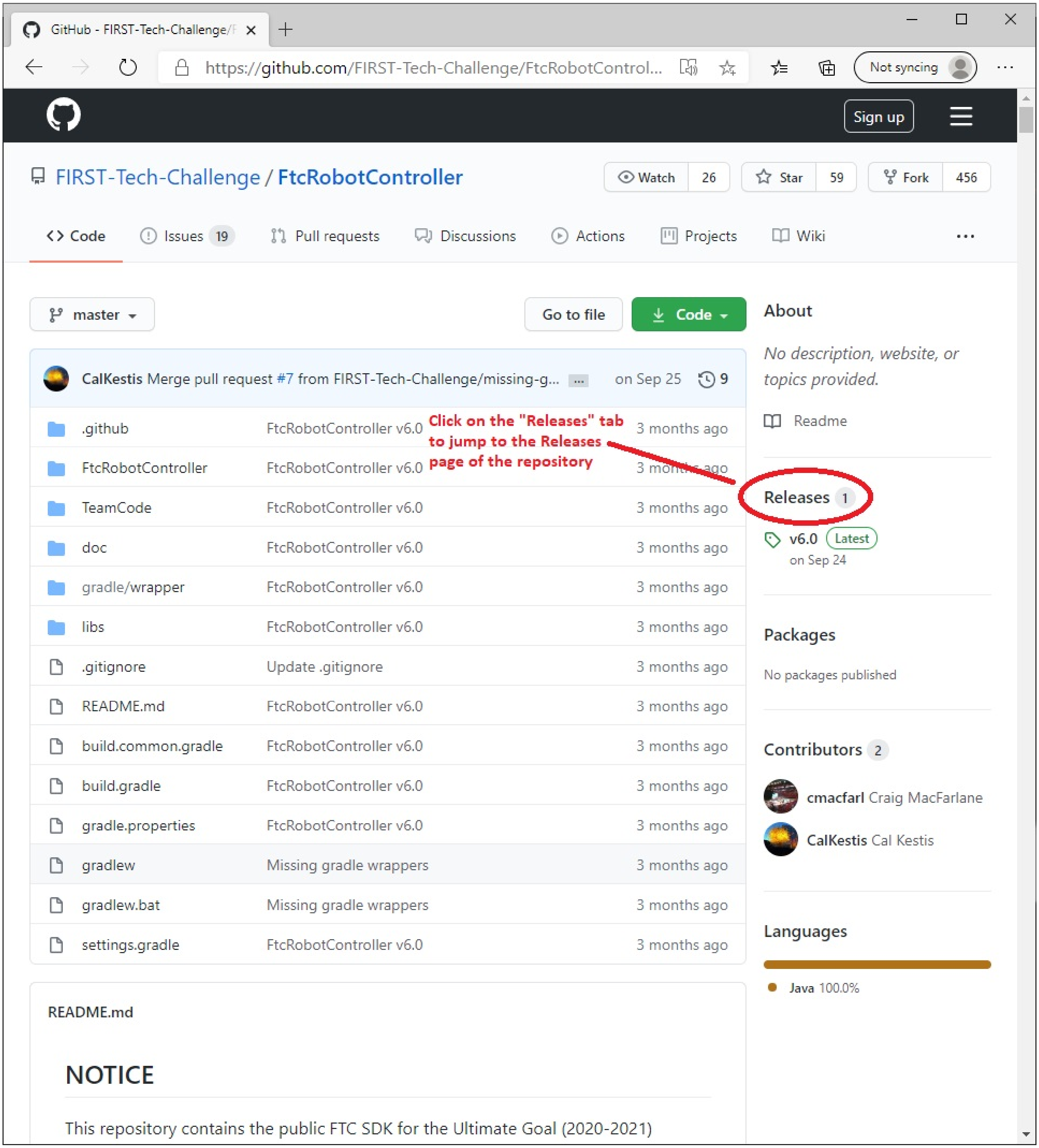
The SDK can be downloaded from a GitHub repository. GitHub is a web-based version control company that lets individuals and organizations host content online. In order to access the Android Studio software, you will need to have a GitHub account. You can create one for free by visiting the GitHub website:

- https://github.com/

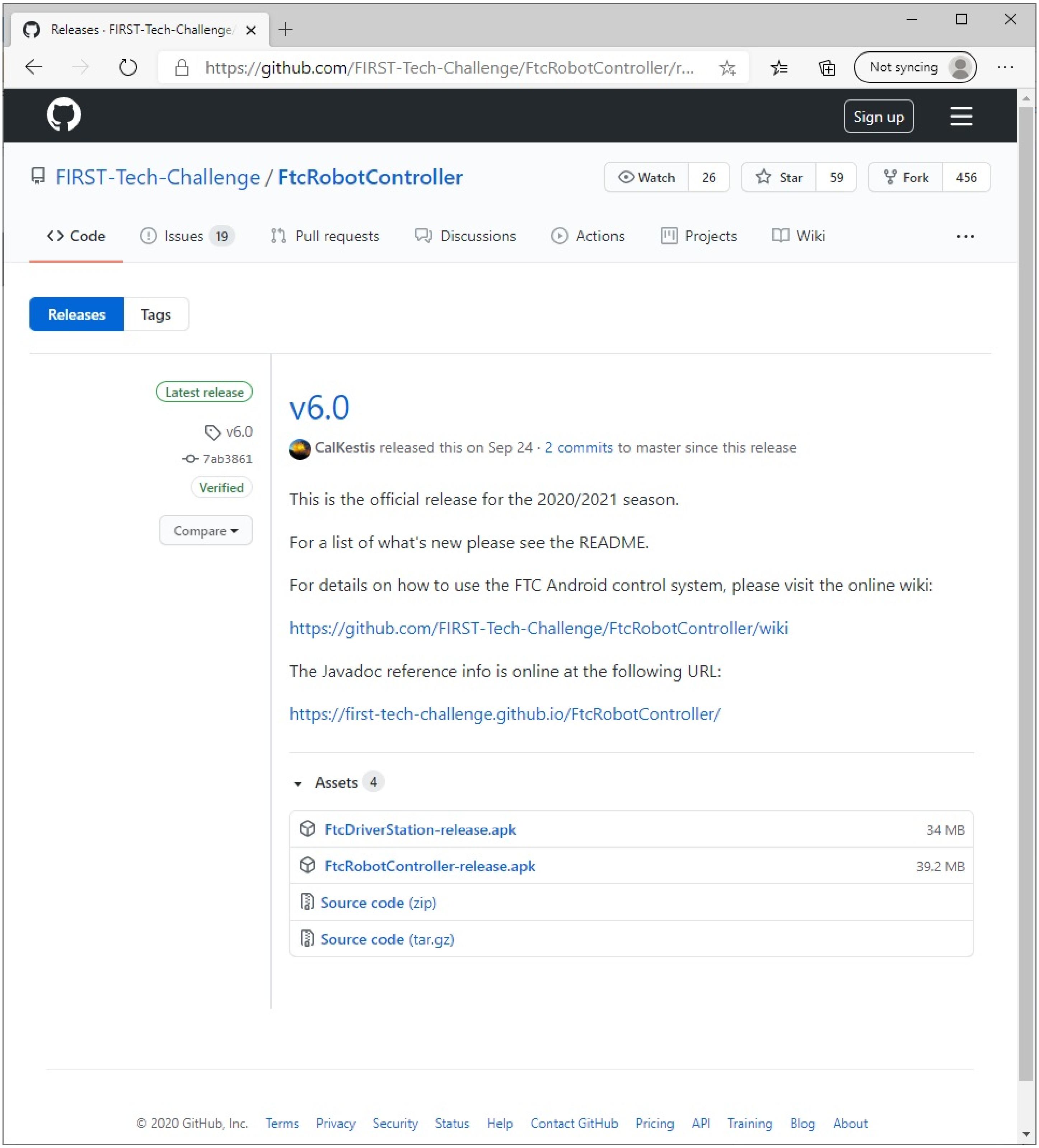
The software is stored in a repository called "FtcRobotController" under the *FIRST-Tech-Challenge* GitHub organization:

- https://github.com/FIRST-Tech-Challenge/FtcRobotController

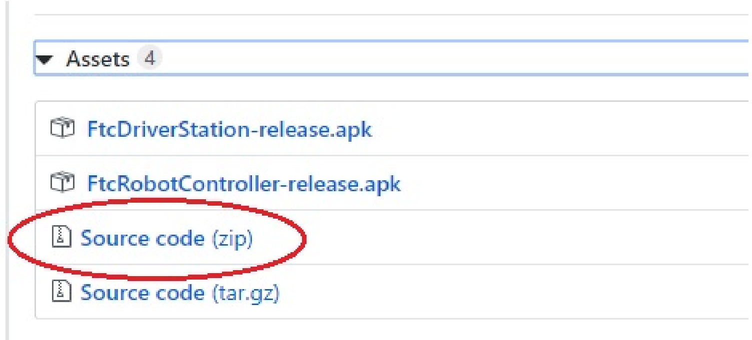
Important: Advanced GitHub Users - this tutorial assumes that the user is a novice with respect to using GitHub and the git version control software. If you are a GitHub power user, you can use git to *clone* a local copy of the public GitHub repository. This document, however, does not explain how to use git to access the repository. It provides instructions on downloading the repository as a .ZIP file instead.



From the main repository web page, click on the "releases" link to jump to the Releases page for the repository. The Releases page should list the available software releases for the repository. The latest release should be displayed near the top of the page.



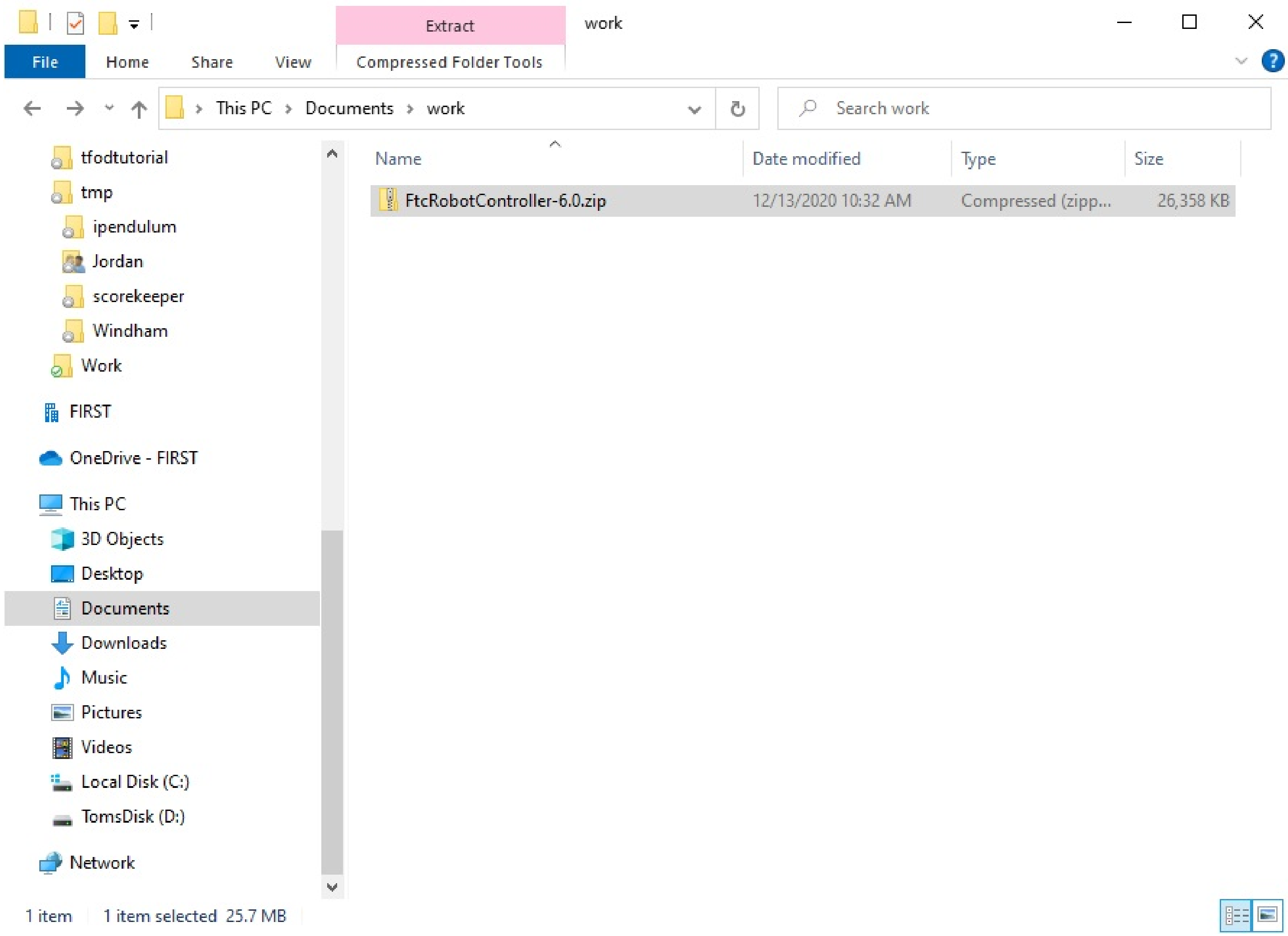
Each software release should include an **Assets** section that you can use to download the software that you will need to program your robot. Note that you might have to click on the triangular symbol to expand this **Assets** section.



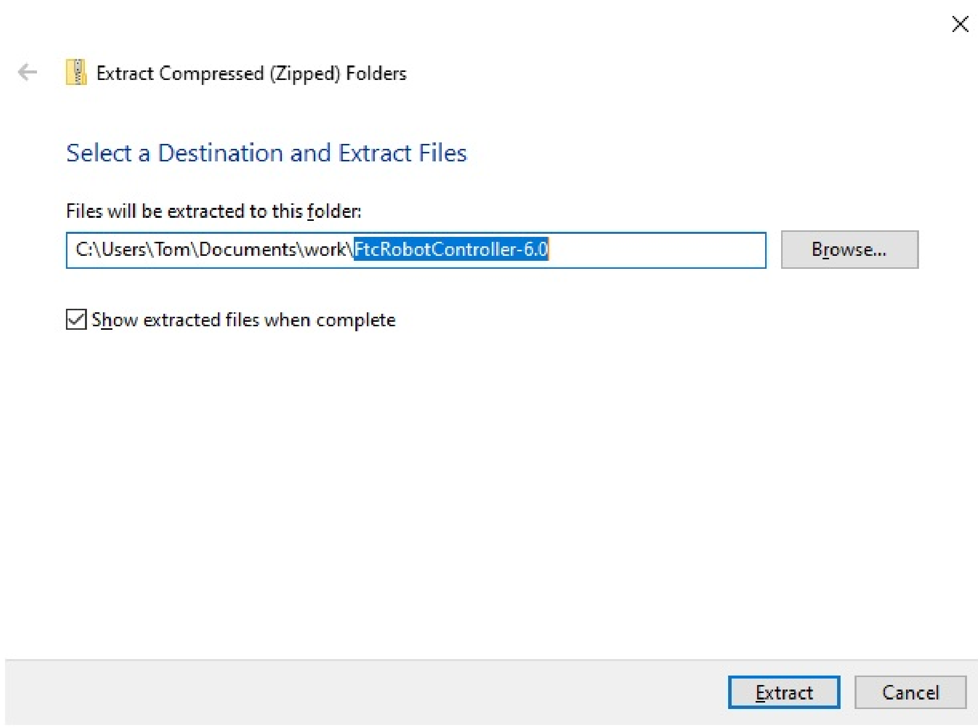
Click on the Source code (zip) link to download the compressed Android Studio project folder.

Extracting the Contents of the Archived Project File

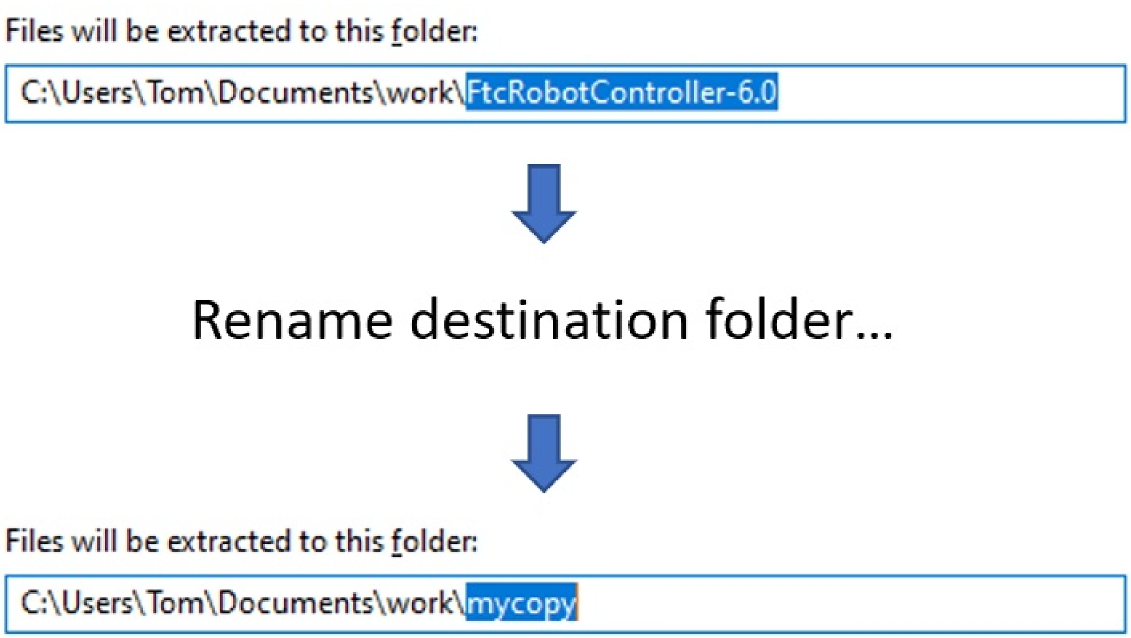
Once you have downloaded the archived (.ZIP) project file you can move this file to the location of your choice.



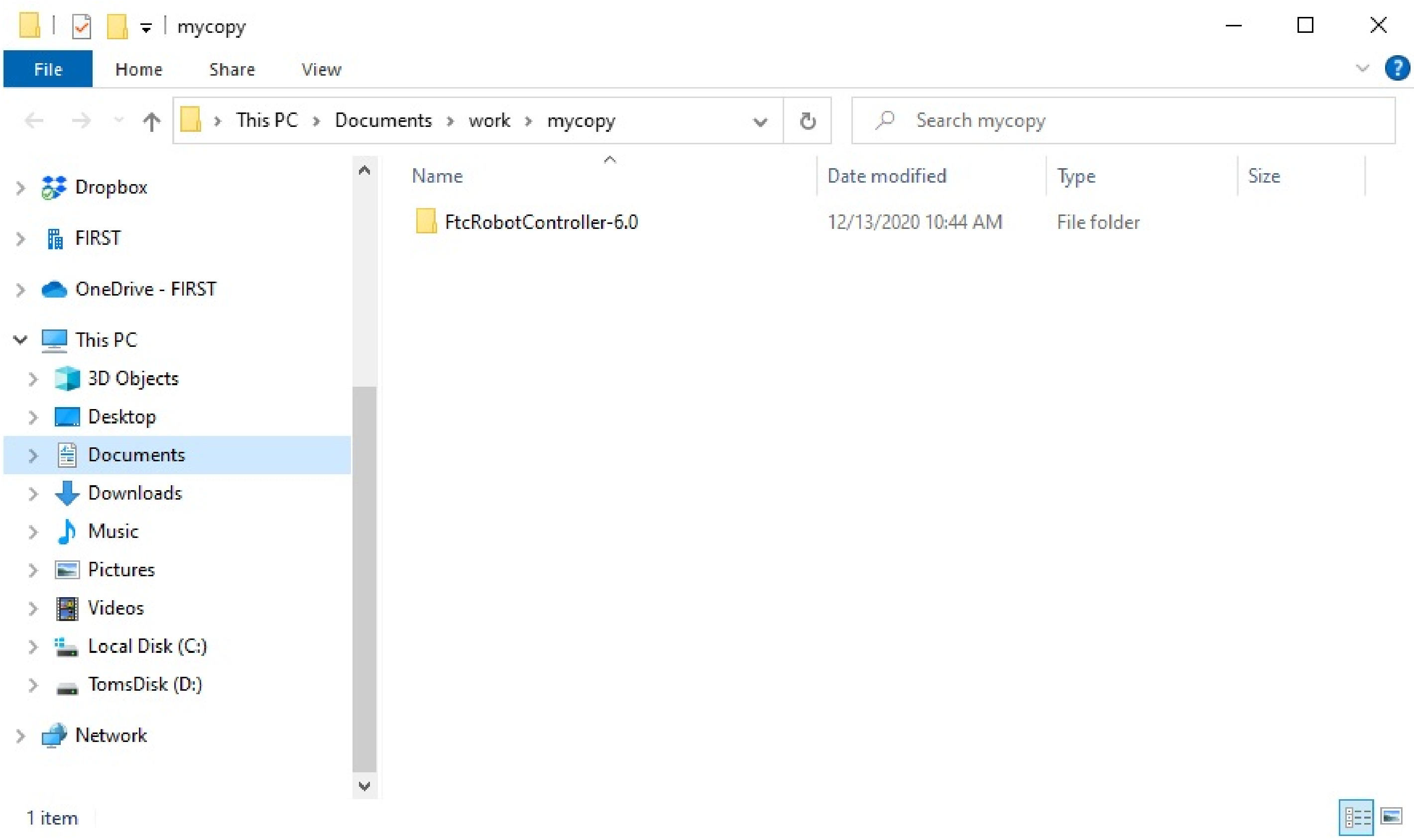
Before you can import the project into Android Studio, you must first extract the contents of the archived project file. For Windows users, right mouse click on the file and select "Extract All" from the pop up menu. Windows should prompt you to select a destination for the extracted project folder. The dialog that appears should look similar to the one show in the figure below.



Highlight the suggested name for the destination folder (in the figure above, the suggested name is "FtcRobotController-6.0") and change the destination folder name into something more user friendly. In this example, we will change the name of the destination folder to "mycopy".



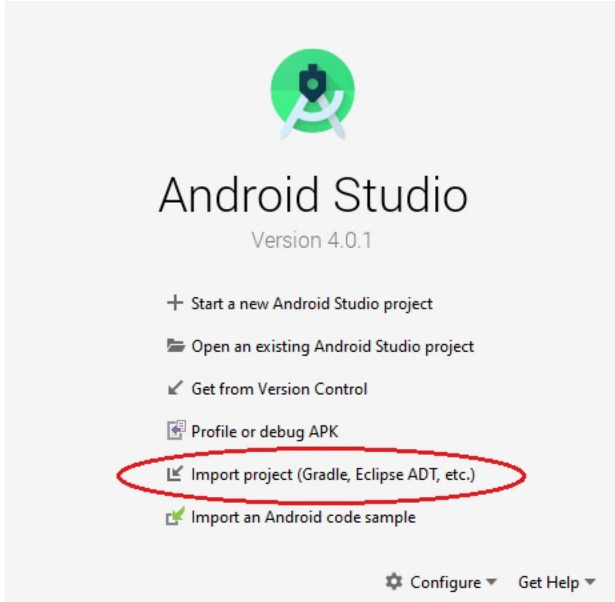
After you have renamed the destination folder, extract the contents of the archive to the folder. After the extraction process is complete, verify that the project folder was successfully extracted to its target destination.



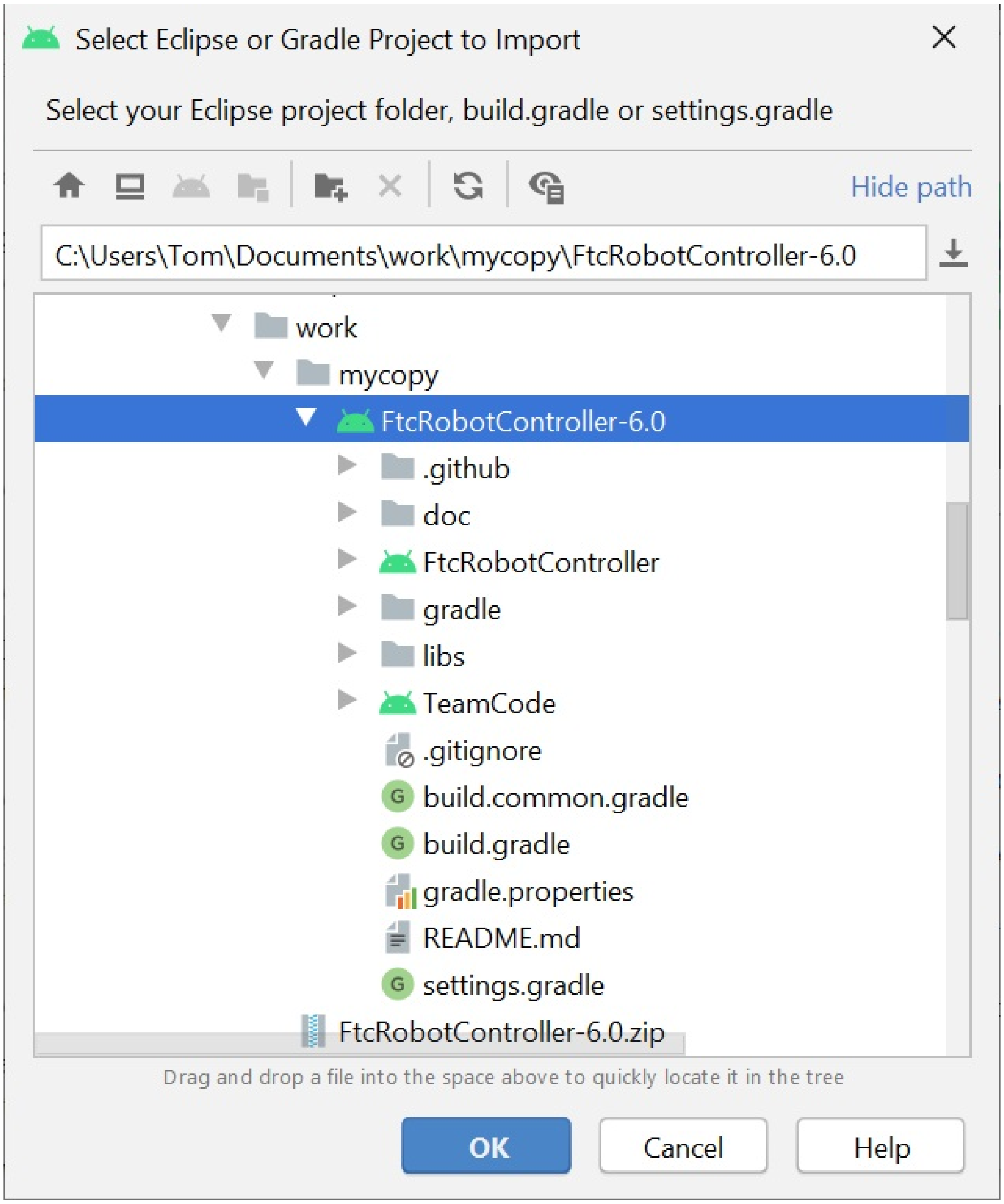
Once you have successfully extracted the contents of the archived file, you are ready to import the project into Android Studio.

Importing the Project into Android Studio

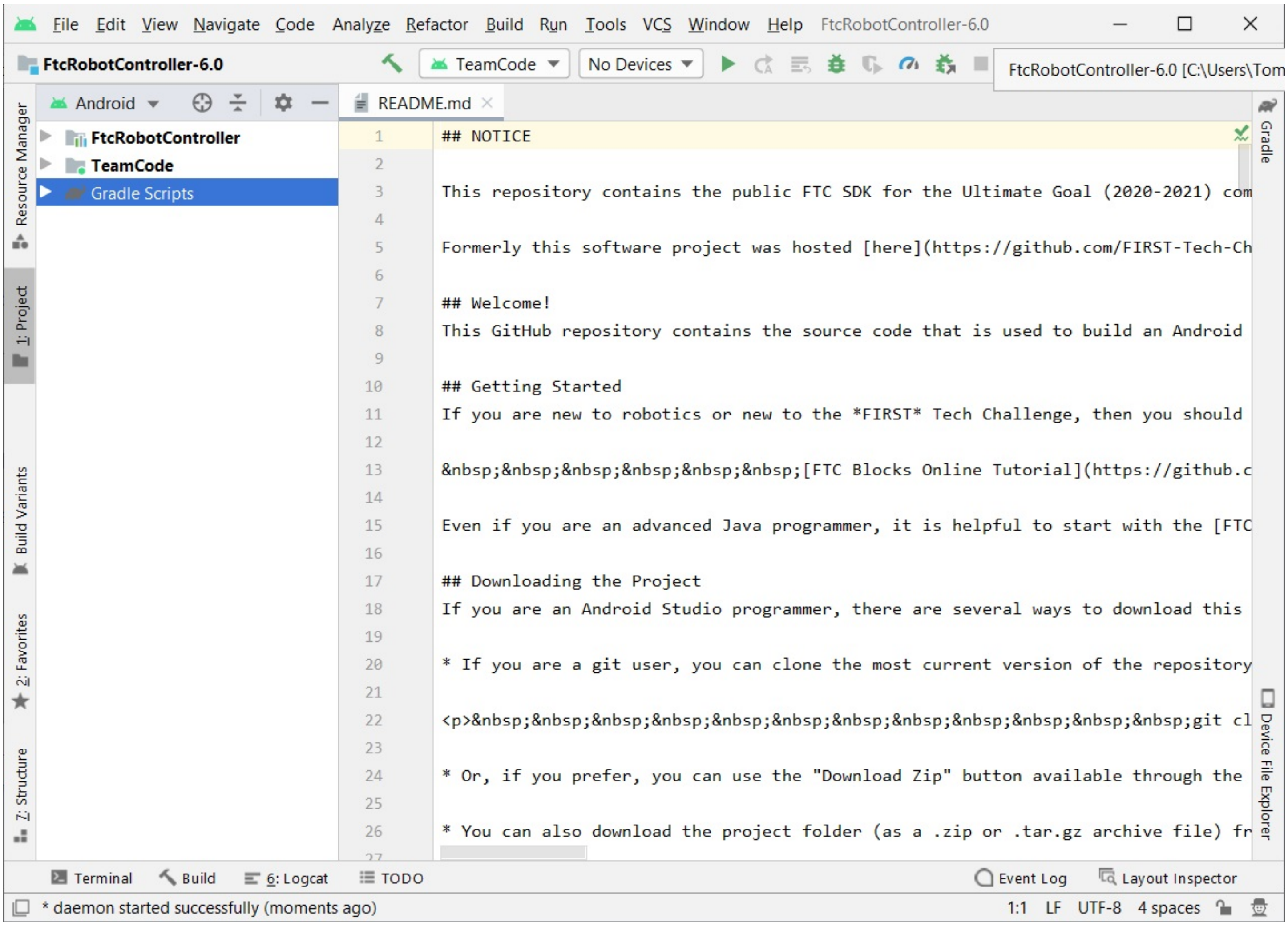
In order to import the Project, you will need to launch the Android Studio software on your computer. On the main Android Studio Welcome screen, select the option to "Import project (Gradle, Eclipse, ADT, etc.)" to begin the import process.



Android Studio should prompt you to select the project folder that you would like to import. Use the file browser in the pop up dialog box to locate and then select the folder that you extracted in an earlier section of this document. Make sure you select the extracted project folder (and not the .ZIP file which might have a similar name to the extracted folder). Hit the "OK" button to import the selected project into Android Studio.



In the figure above the project folder called "FtcRobotController-6.0" is selected to be imported into Android Studio. It might take Android Studio several minutes to import the project. Once the project has been successfully imported, the screen should look similar to the one depicted in the image below.



Disabling Android Studio Instant Run Legacy AS

> **Attention:** *Instant Run* was removed in Android Studio version 3.5, and is no longer an issue for versions of Android Studio that are Android Studio 3.5 or newer. However, this article remains for those using *FIRST* Tech Challenge Software Development Kit (SDK) v7.1 and older with previous versions of Android Studio.

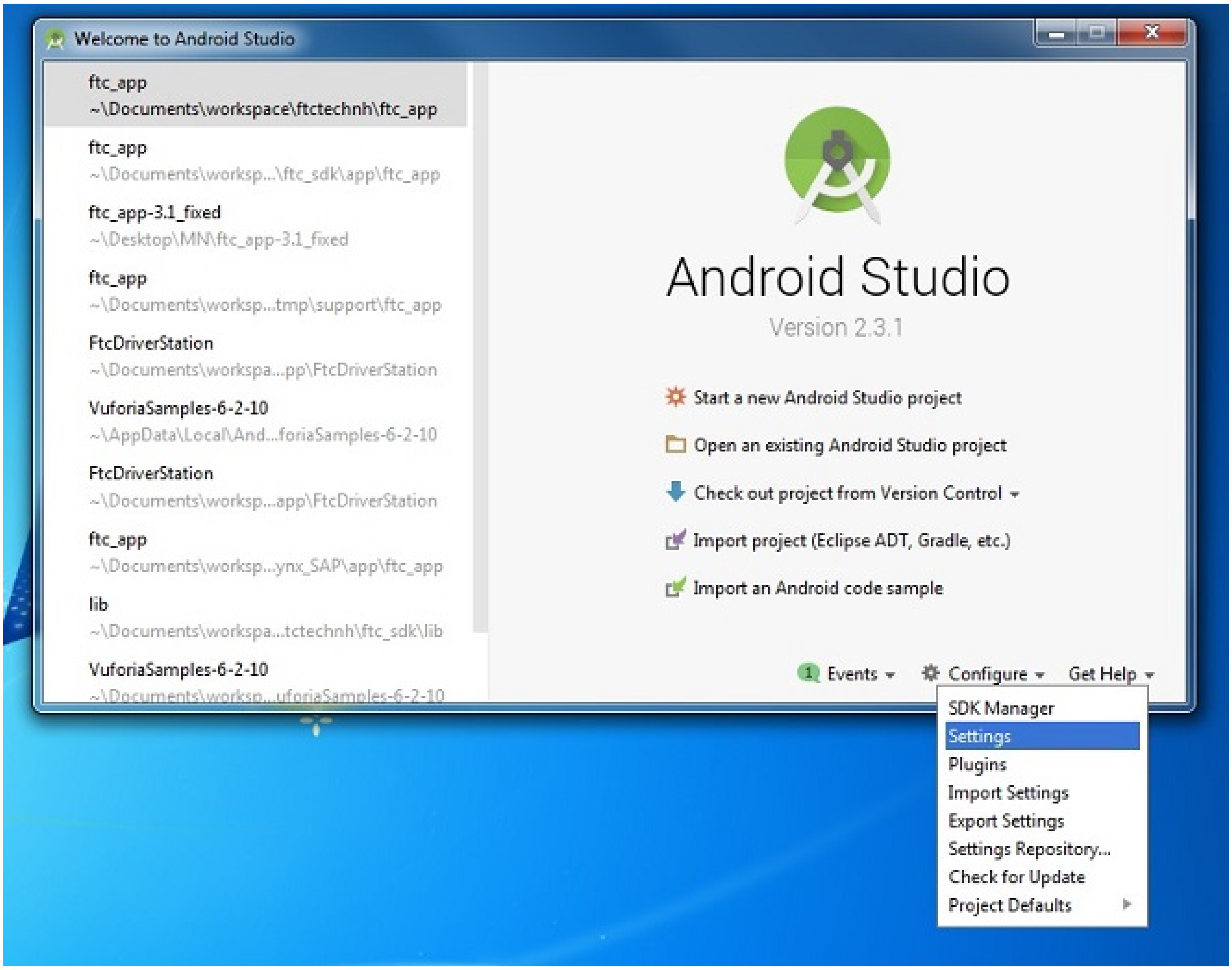
Introduction

If you are an Android Studio user, one of **the most important steps to take** is to disable Android Studio Instant Run. Instant Run is a feature that is designed to streamline the development process by reducing the time to apply code changes to your app. Unfortunately, Instant Run is limited in function and when used with the *FIRST* Tech Challenge Android Studio project folder, can cause **severe** and **difficult-to-troubleshoot** problems.

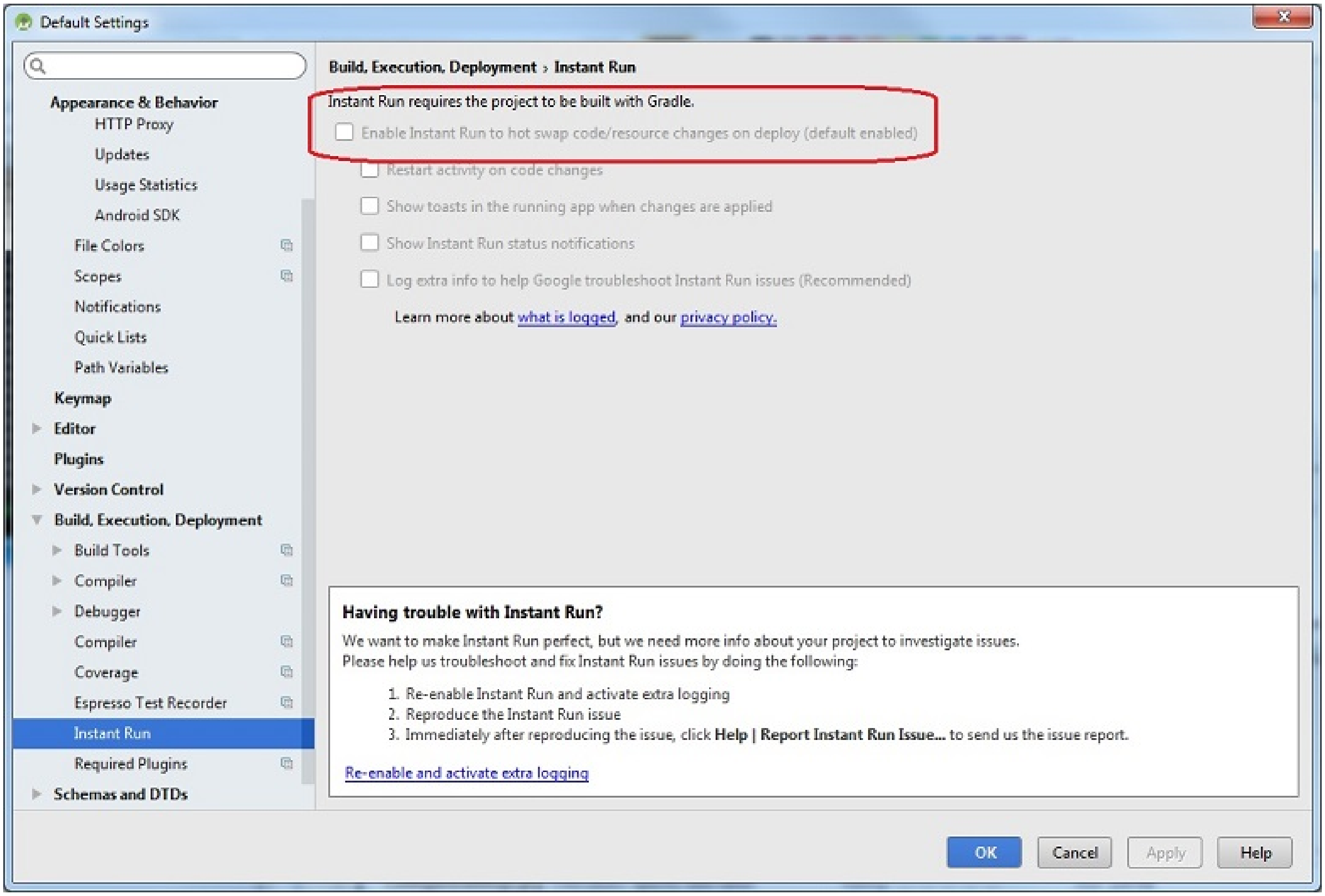
Teams who use Android Studio **must** disable Instant Run.

Locating Instant Run Settings

When you first launch Android Studio a Welcome screen should appear. You can navigate to the Instant Run Settings from this Welcome screen by selecting the "Configure->Settings" item from the "Configure" dropdown list in the lower right hand corner of the screen.



On the left hand side of the Settings window, there should be a category called "Build, Execution, Deployment". Within this category, click on the "Instant Run" subcategory to display the Instant Run settings for your Android Studio installation. By default, Instant Run is enabled when you first install Android Studio. Uncheck the "Enable Instant Run to hot swap code/resource changes on deploy (default enabled)" option and then click on the "OK" button to disable Instant Run.



Additional Information

The Google Android Developer website has additional information about Instant Run. It also has instructions on how to disable this feature:

https://developer.android.com/studio/run

Managing an Android Studio Project AS

Fork and Clone from GitHub AS

Important: This approach assumes a basic familiarity with git and GitHub. As with most things related to git there are many different ways to satisfy any objective. This documentation describes one method for Windows users. Users not comfortable with command line tools and git should obtain the SDK via *Downloading the SDK as a zip archive*.

Forks vs. Clones

A Fork on GitHub is a copy of another repository on GitHub from one account to another account. The new forked repository retains a parent-child relationship with the origin repository. Forks are typically used when software will have an independent line of development, such as when FTC teams develop their own team code using the FIRST-Tech-Challenge/FtcRobotController repository as a basis. FTC teams should create a Fork of the FIRST-Tech-Challenge/FtcRobotController repository as a convenient way to manage their software development process. Thanks to the parent-child relationship, when changes are made to the parent repository those changes can be easily tracked and fetched/merged into the forked repository, keeping the forked repository up to date.

Warning: Teams should not issue pull requests against the upstream parent, the FIRST-Tech-Challenge/FtcRobotContoller repository. Forks of the FIRST-Tech-Challenge/FtcRobotContoller repo may always fetch changes, but should never attempt to push changes up to the repo.

A Clone is a copy of a repository, typically on a local computer. A team member creates a feature branch of the team's repository for feature development, and clones the branch to a local computer. Software development and testing then happens completely within their local clone. Once they're finished, or they've reached a checkpoint, the changes within the local clone can then be pushed from their local clone back to the team fork. That feature branch can then be merged into the team's main repository branch once it has been accepted by the team. Multiple different developers can work seamlessly using this process.

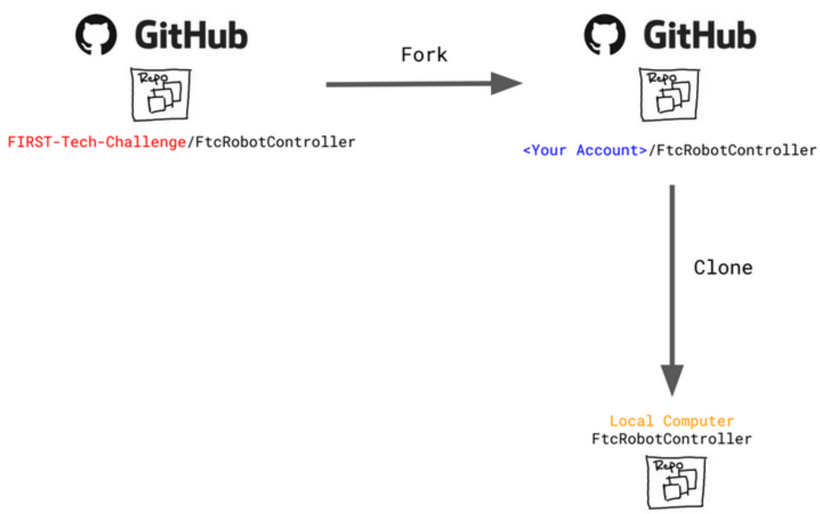


Fig. 7: The relationship between forks and clones. The clone exists on your local laptop while the fork exists on GitHub servers.

Branch Strategies

A branch is a series of commits that are independent of any other lines of development and is typically used to develop new features for the repository. The default branch for the FtcRobotController repository, and its forks and clones, is master (though for all newer repositories created by GitHub the default branch is called main). Using branches judiciously can help developers collaborate on a common set of software by isolating changes, keeping the default branch clean, and providing space for feature development to iterate independent of software that's been deemed 'production ready'.

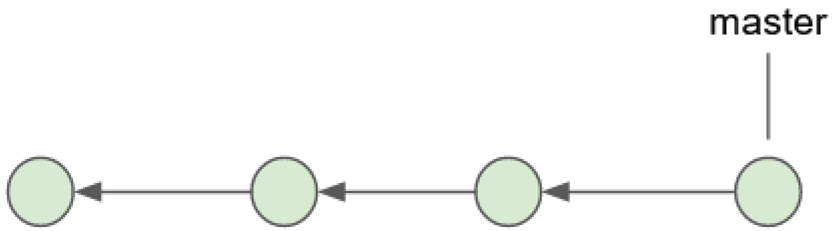


Fig. 8: A single branch with the default name of master

Each circle represents a commit to a branch. The name of the branch always points to the most recent commit, also known as the HEAD. While there may be many branches there is only one HEAD and it always, unless it is in a detached state, points to the latest commit of the currently checked out branch. All other commits point to their immediate parent.

A commit is a snapshot of the entire workspace at a point in time. Git does not store diffs. If you make a change to a file, and create a new commit with the changed file, it stores the entire changed file in the commit. To avoid unnecessary duplication of files, if your repository consists of three files - one changed and the other two were unchanged - then the snapshot merely points back to the unchanged files rather than containing unchanged data.

Note that each commit has a parent which allows git to determine reachability of commits from different branches. It also allows git to determine the common ancestor commit of any two branches, which is important when merging branches. More on that later.

So what is a branch? A branch is simply a named pointer to a commit. When a branch is created you are just telling git to create a name, and point it at a commit. Being on a branch simply means that when you add a new commit, git moves the branch name to the new commit and the new commit's parent is the commit that the branch name was pointing to previously. Since this creates a line of development independent of the parent, developers can experiment, make changes, develop new features, all without disrupting the work of other team members. When a developer is satisfied that a branch is stable enough to be shared, the branch can be merged back into the parent.

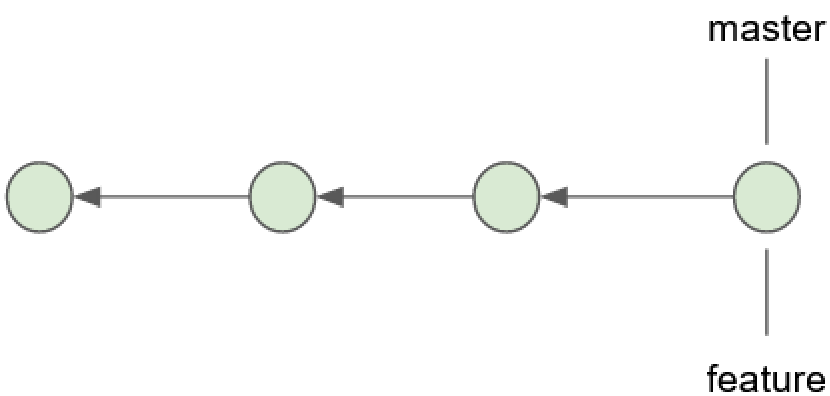


Fig. 9: Two branches that point to the same commit.

Immediately after creating a branch the new branch name simply points to the latest commit from the branch that the new branch was created from. Now imagine that we create a new commit on that branch.

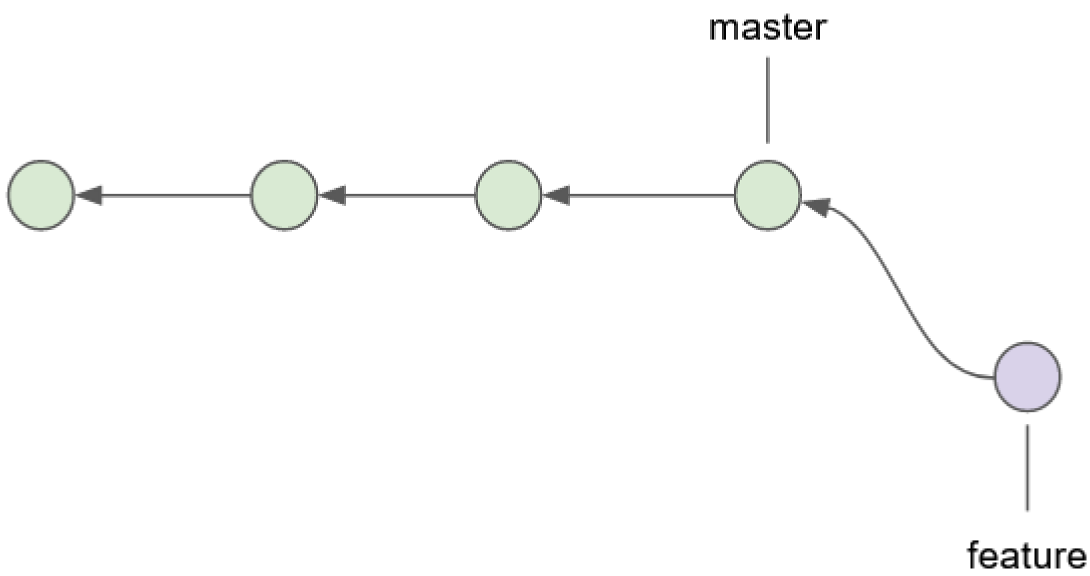


Fig. 10: New commit on the feature branch.

Note how the new commit caused the name pointer of the feature branch to move to the new commit, while the name pointer for the master branch remains on the prior commit, but the parent of the new commit is the commit that the name pointer for master points to. If a new commit is added to the master branch then the parent of the new commit is also the commit that master is pointing to thereby creating independent lines of development.

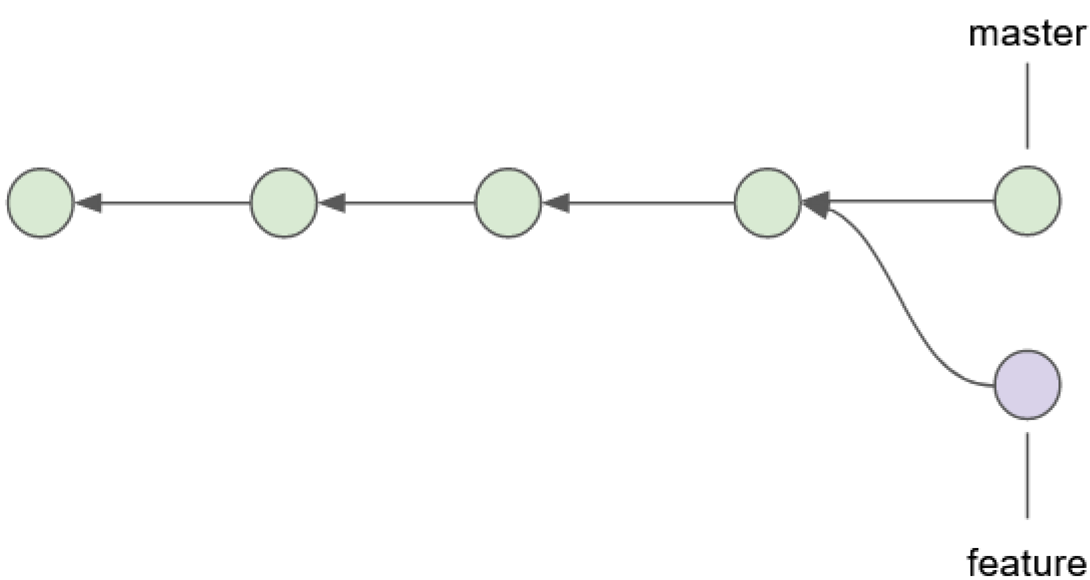


Fig. 11: Two independent lines of development.

Eventually you typically want to merge that feature branch back into the main line of development represented by the master branch. When you merge one branch into another, git traverses the ancestor commits of the branches to find the common ancestor. It then determines what changed from the common ancestor, to the head of each branch, and applies those changes to a new commit called a *merge commit*. An artifact of this process is that the merge commit will have two parents.

As shown above, the feature branch still exists. New commits added to the feature branch will diverge again from the master branch. However if development of the feature is finished, the branch can be deleted. Deletion of the branch simply results

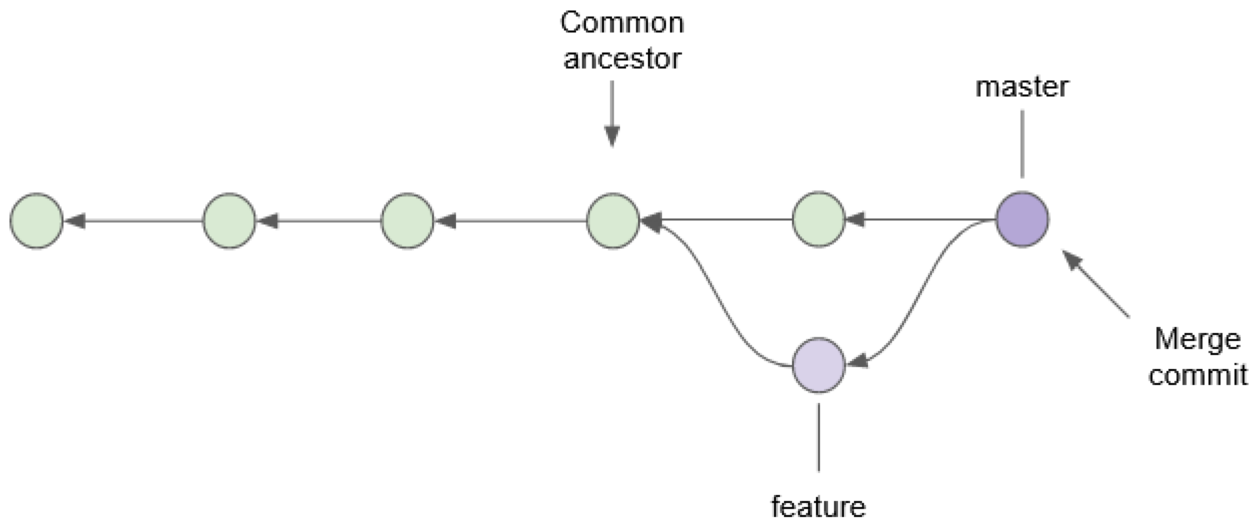


Fig. 12: Merging the feature branch back into the master branch.

in the name pointer being deleted. Branch deletion does not result in the deletion of any commits that were made on that branch. As you can see here, the commit that was on the feature branch still exists and is reachable by referencing the correct parent from the merge branch.

It can be useful to ensure that the default branch in team forks and clones matches the default branch for FIRST-Tech-Challenge/FtcRobotController. However a typical development pattern will have team developers committing team software back to the master branch, whether via merges from feature branches, or direct commits to master.

Team commits are represented by blue circles, while commits containing SDK updates are represented by green circles. The purple circle is a merge commit. More on merges later. In this instance team commits are interleaved with SDK updates (1), which produces a situation where the two default branches do not match.

> (1) Not really, or maybe depending upon how the commit parentage lays out. This is a vastly simplified view of things, but is sufficient to demonstrate the logical concept and is the view of things you get if you simply execute git log. For an in-depth, approachable, explanation of exactly what is happening with commits as they relate to branches see this tutorial.

While this is a perfectly acceptable, and a very common branch management strategy, certain benefits can be obtained if we isolate the default branch so that it always matches the parent. The following figure demonstrates a clone whose master branch is tracking the master branch from FIRST-Tech-Challenge/FtcRobotController.

The purple commit is a merge of v7.1 into the competition branch. In this diagram, v7.2 and v8.0 remain unmerged and the competition branch will be building against v7.1 of the SDK.

Following this model means that commit history for the master branch for the team's repository will always match the commit history for the FIRST-Tech-Challenge/FtcRobotController's master branch. All software that teams intend to compete with is merged into a competition branch. Features, new software, experiments, etc, are worked on in child branches of the competition branch and merge back into the competition branch, not the master branch. SDK updates to a team clone's master branch should always be conflict free, updates can be done independent of merges into a competition branch, and if something goes sideways when doing a merge of an SDK update into development it can be more straightforward to recover as opposed to backing out of an update straight into master where the branches do not match.

More detailed information on the mechanics of branching can be found here Using Branches

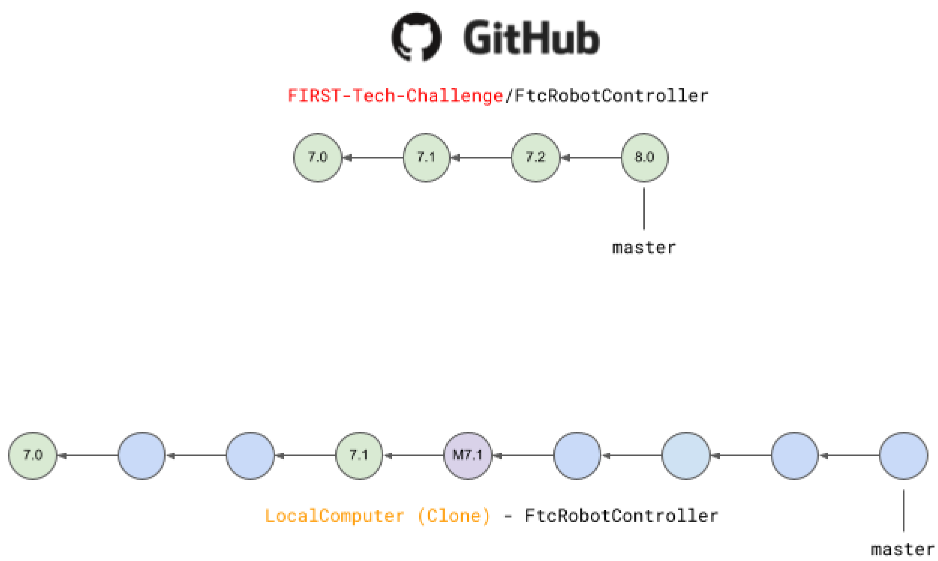


Fig. 13: FIRST-Tech-Challenge/FtcRobotController master vs. typical team repository master.

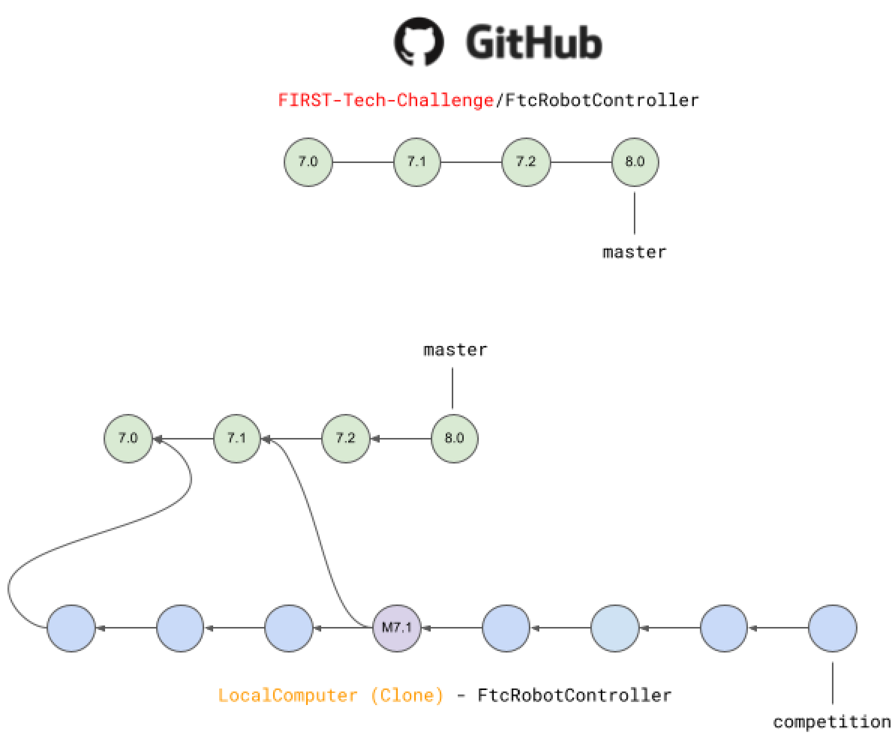


Fig. 14: Team repository's master always matches FIRST-Tech-Challenge/FtcRobotController's master branch.

Getting Started (Quick-Start Guide)

Important: The following assumes all operations are done on the master branch of your local repository.

---

1. Obtain and install GitForWindows This software contains a git client along with a bash shell. All of the command line snippets below assume you are using a bash shell and that git is in your path. GitForWindows is the easiest way to provide this for Windows machines. Macs have a built in bash shell called terminal, but git must be installed separately.
2. Fork the FIRST-Tech-Challenge/FtcRobotController repository into your account on GitHub.

Tip: This step requires you to have a GitHub account, and you need to be logged in to GitHub in order to Fork a repository.

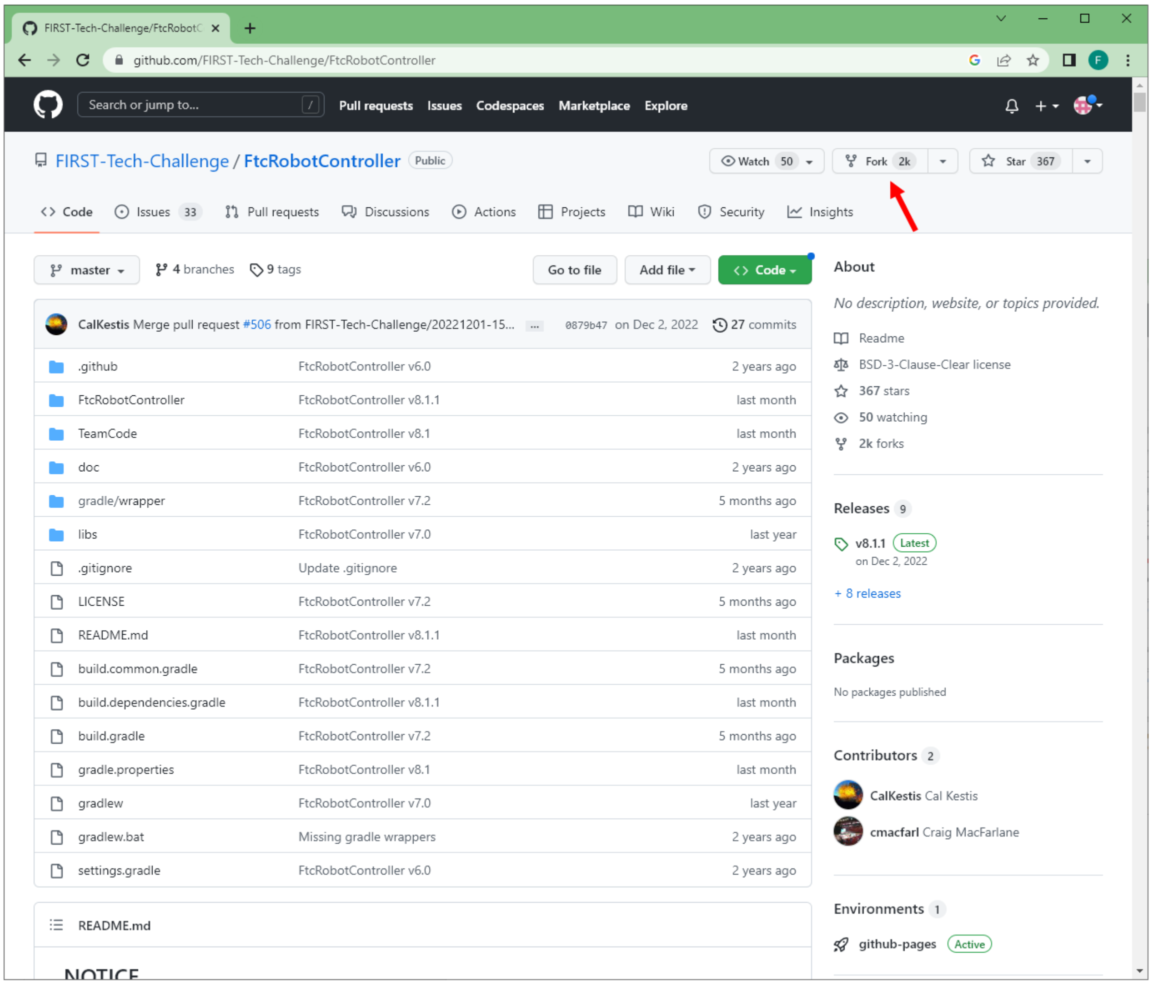


Fig. 15: Forking a GitHub repository.

Forking the repository is as easy as clicking the " Fork" button shown in the image above. This will take you to the "Create a new fork" page, and will auto-fill the "Owner" and "Repository name" fields. Just enter a description (optional), leave the "Copy the `master` branch only" option checked, and click the green "Create fork" button.

Once created, your new fork will be located at `github.com/<username>/FtcRobotController` unless you edited the fork name.

3. Clone from your fork onto your local computer. Note in the image below the account is FIRST-Tech-Challenge, but after your fork, the account should be your team account. In all other respects the user interface will be identical.

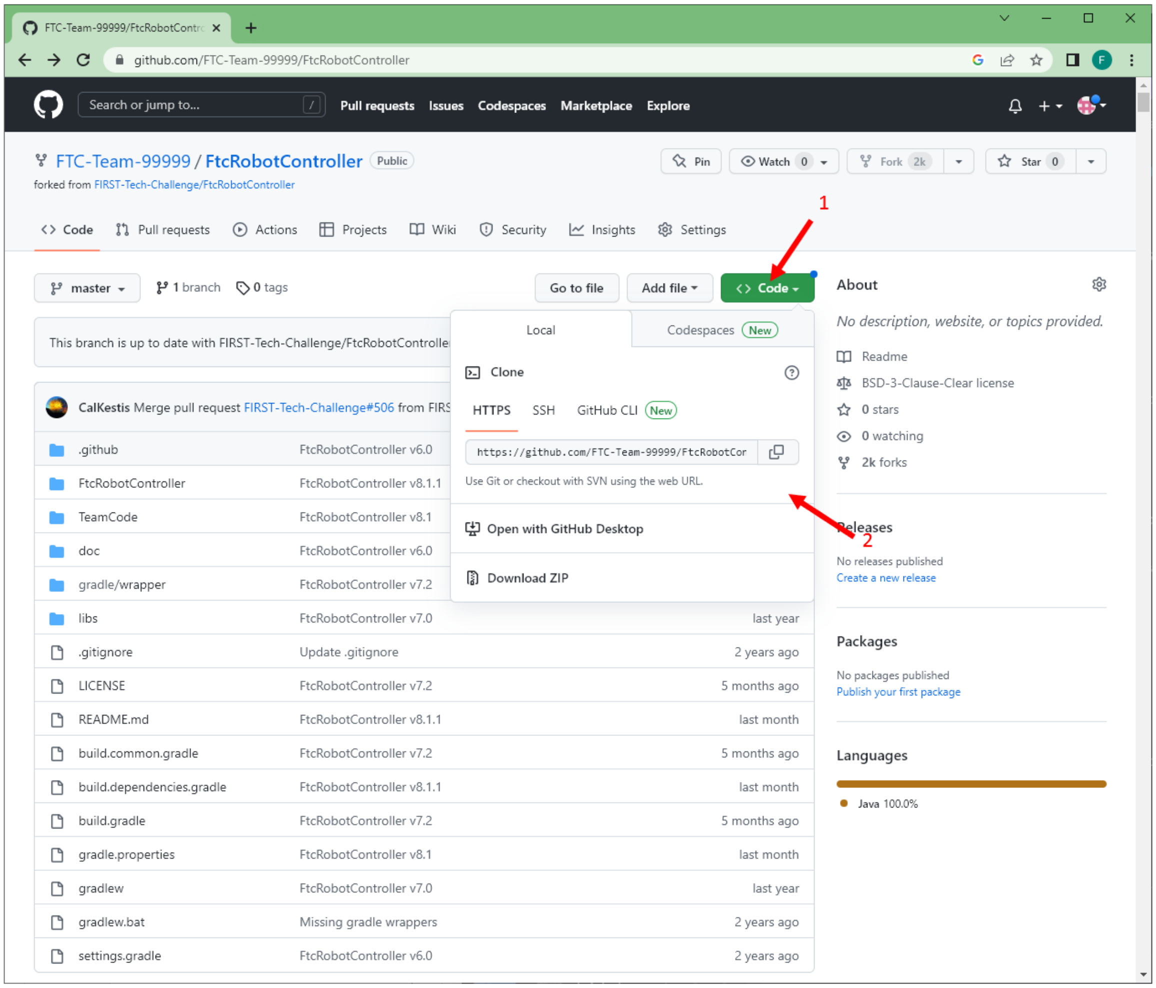


Fig. 16: Cloning a forked repository.

To clone your fork of the FtcRobotController, follow these steps:

1. Click the green " Code" button shown in the image above.
2. Ensure the "Local" and "HTTPS" sub-tabs are selected.
3. Click the "" button to copy the url in the text entry box.
4. Open a "Git Bash" shell in a suitable directory. This is easily done on Windows by opening the File Explorer, finding the directory you want to clone the repository into, right clicking on that directory folder and selecting "Git Bash here"
5. Within the Git Bash shell, execute the following command

```
git clone <copied-url>
```

4. Git will download a clone of your repository. When it's done, Code away…
5. This is the point where you can create a branch for feature development, if desired. To create a branch, we can create and switch to a new branch via the following git-checkout command:

```
git checkout -b <branchname>
```

Using the `-b` option creates the new branch specified by `<branchname>` and automatically switches to that branch. Omitting the `-b` option will simply *switch* to an existing branch if one exists.

Best Practices

- Do not make changes to software in the FtcRobotController directory within the repository. SDK updates will be much easier if you do not change anything within the FtcRobotController directory.
- Limit the use of long-lived branches. Branches should implement a feature. Branches should not track milestones. For example a branch named 'league-meet-1' is tracking a milestone. It is much better if your branches track smaller units of development. 'detect-target', 'drive-to-parking', 'drop-game-element'. Break your software down into tasks for the robot to do, and use branches to implement those tasks. This will allow for much easier collaborative development, much smaller change sets when merging, and much easier fetches and merges.
- Try to keep your git index clean. This will make fetches and merges easier. `git status` is your best friend here. Use `git status` often to see what has changed in your local workspace. Commit often in logical chunks so that it is easy to see the most recent changes.
- Use short, meaningful, commit messages. Do not use slang, offensive, or personal messaging in a commit message. When you push your software to GitHub, those commit messages will be public. If you plan to eventually become a professional software developer, and you retain your existing GitHub account any potential employer will be able to review your commit messages. Tread lightly here.

Updating your Fork and Local Clone.

Updating the SDK involves pulling newly released software into both your local clone's and your fork. There are two ways to go about this. Either directly fetch and merge software from the parent into your fork on github, then fetch and merge to your local, or fetch from the parent into your local clone, merge locally and then push to your fork.

This author prefers the latter because it gives the developer the opportunity test new software before pushing to the fork. It also allows for merge conflict resolution locally instead of through GitHub's UI.

Obtaining the Latest Software

When describing how to update a repository many basic tutorials will use the `git pull` command. The `git pull` command is actually doing a *fetch* and *merge* for the user behind the scenes. This can be fine, but it is useful to understand the concepts of *fetching* and *merging* as independent operations. If things go south, and you have a good concept of the underlying mechanics, you are much more likely to be able to fix any subsequent problems.

Remotes

Git is fundamentally built around the idea that there can be many copies of a repository floating about on the internet, or other people's machines, or corporate file servers, or any number of locations. And that these repositories can linked to each other remotely. A remote repository is simply defined as a version of a repository hosted somewhere else. In the preceding examples, your fork of FtcRobotController is a remote of your local clone.

Remotes may be referenced in git commands and a repository can have any number of remotes. The default name for the remote of a repository that has been cloned is 'origin'. The conventional name of a remote that tracks the parent of a fork is 'upstream'.

To see what remote are established for a given repository

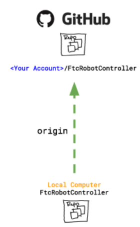


Fig. 17: Illustration of FtcRobotController as remote named *origin*.

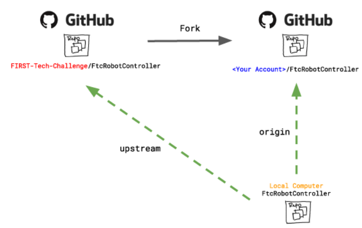


Fig. 18: A local repository with two remotes.

```
$ git remote -v
```

To add the parent of your team's fork as a remote of your local clone

```
$ git remote add upstream https://github.com/FIRST-Tech-Challenge/FtcRobotController.git
```

Important: Setting the FIRST Tech Challenge FtcRobotController repository as an upstream remote of your local clone allows you to fetch changes from the FIRST-Tech-Challenge/FtcRobotController to your local clone using the alias name 'upstream'. This is very powerful. If the reason why this is important isn't immediately obvious, please re-read the two paragraphs under header marked `Updating your Fork and Local Clone` above.

The rest of this tutorial assumes that you have added FIRST-Tech-Challenge/FtcRobotController as an upstream in your local clone.

Fetching

Fetching is the process of downloading software changes from a remote repository. Note specifically that fetching **does not** modify any of the existing software in the repository that you are fetching into, git isolates the changes in the local repository.

If you are working with a team, and a teammate has pushed software to your FtcRobotController fork, you may fetch that software to a local clone by running

```
$ git fetch origin
```

This will download any changes in all branches on the remote named origin that are not present in the local repository.

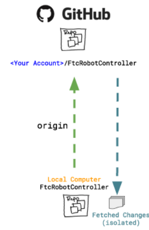


Fig. 19: Fetching changes from origin.

Merging

Merging is the process of merging fetched software into a branch, most commonly the current branch of the repository. A merge is where things are most likely to get a bit confusing. However, if you are simply merging from a remote master into a local master, and your local master is always tracking the remote, your merges should go smoothly.

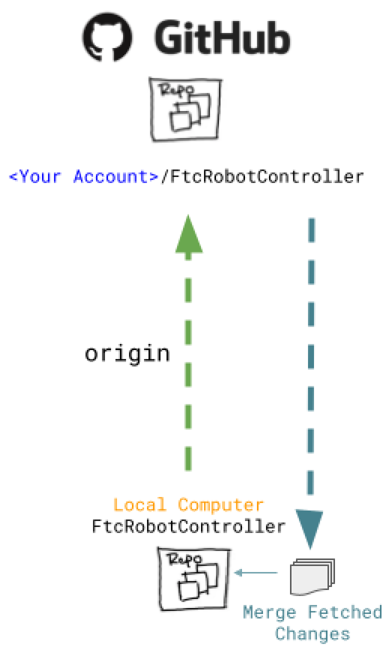


Fig. 20: Merging fetched changes from the origin repository.

Ensure you are on the `master` branch and run the following:

```
$ git merge origin/master
```

The `master` branch should be *clean* (i.e. `git status` on the `master` branch shows no files that are modified but uncommitted) when this operation is performed. Team members should be doing development work in feature branches, not in the `master` branch.

Conflicts

Conflicts, or "What happens when more than one change is pending for a given piece of code." It's best to read this great tutorial on Git merge conflicts. Merge conflicts are a normal part of working in teams, and only with experience can you learn to effectively manage conflicts. Always approach with patience and a deep respect for the process.

Updating the SDK to the Latest Version

Important: Remember to use `git remote -v` to ensure that the upstream has been set as a remote on your clone. If not, be sure to review the "Remotes" section again to add the FtcRobotController repository to the upstream remote on your clone.

To update from the SDK, we simply fetch from upstream, FIRST-Tech-Challenge/FtcRobotController, the parent of your team fork, then merge and push to origin to complete the update.

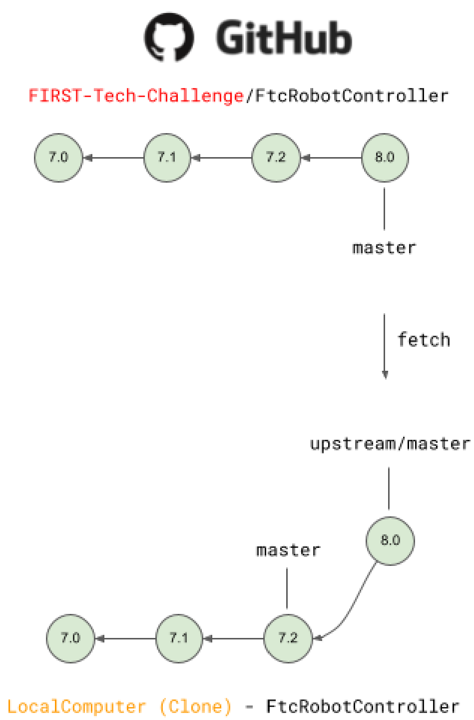


Fig. 21: Fetching changes from the upstream repository.

Instead of fetching from origin, fetch from upstream. This copies in any commits that you don't already have in your local clone. In the diagram above that is the v8.0 commit. Your local master is not changed. It is still pointing to, and representing, the v7.2 commit. Since a commit is a complete snapshot of a workspace at a point in time, nothing changes in your workspace, but your repository has a new commit with the branch name upstream/master.

```
$ git fetch upstream
```

After fetching, merge the upstream/master branch into master. If your local master matches your upstream master then a merge is as simple as moving the master branch label to the commit that upstream/master is pointing to. This is referred to

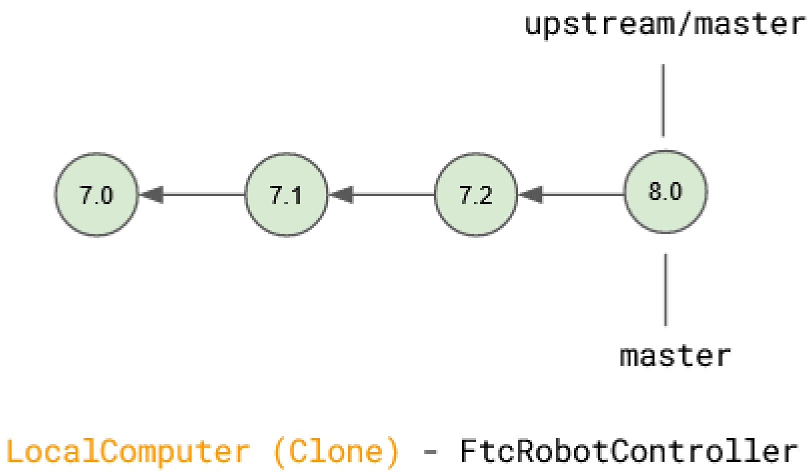


Fig. 22: Merging fetched changes from the upstream repository.

as a fast-forward merge. And since a commit is a complete snapshot of a workspace at a point time, your local workspace now contains the snapshot represented by v8.0.

```
$ git merge upstream/master
```

Once you've merged the upstream/master into your local clone's master branch, push those changes to GitHub so that your GitHub clone reflects the upstream repository.

```
$ git push origin master
```

If you were working in a feature branch and want to bring the new SDK changes into that feature branch you merge from master into the branch by checking out the branch and running the merge command. This is where things might get dicey as this is where you are most likely to encounter merge conflicts.

```
$ git checkout <feature-branch>
$ get merge master
```

Downgrading the SDK to a Previous Version

Typically, the working branch of a local repository, whether it's master, or a competition branch will eventually contain a series of team commits interleaved with SDK update commits. In this scenario a team can not simply roll back to a prior SDK version without also rolling back all of their team commits. Consider the following diagram.

If you just chopped off the branch at M7.2, you'd lose the three blue team commits. In order to retain team work, instead create a new merge commit that reverts the 8.0 commit. Do not revert merge commits, e.g. M8.0. The merge commit itself may contain work that represents the divergence of the the two branches that were merged. This is not what you want. You want to revert the parent of the merge commit that represents the new, old, SDK version.

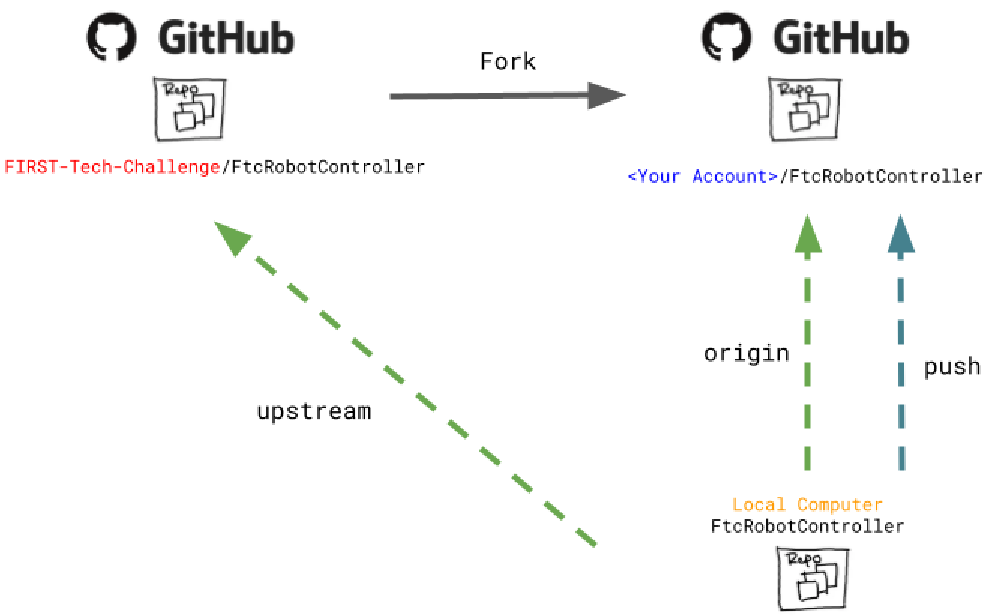


Fig. 23: Pushing fetched and merged changes back to your team fork.

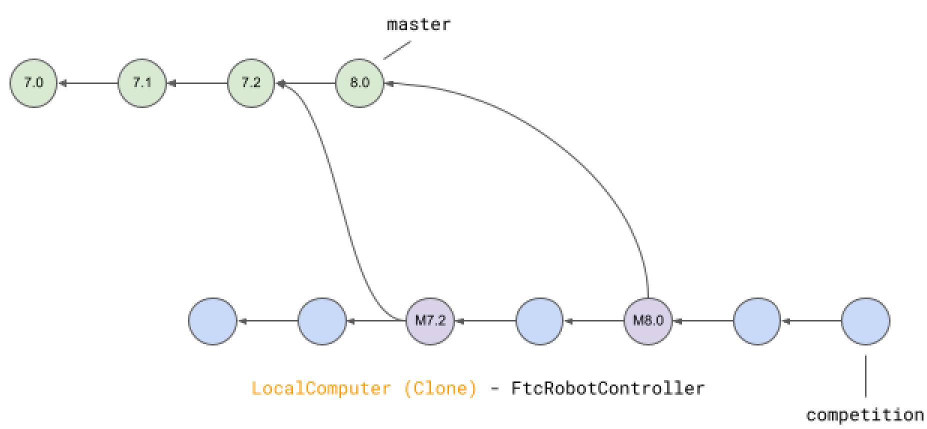


Fig. 24: A repository with both team commits and SDK update commits.

A Short Digression on Tags

A tag is simply a named pointer to a commit, that unlike a branch pointer, or HEAD, never moves. Since a commit is a snapshot in time of an entire workspace, this allows a developer to *tag* a point in time in an immutable fashion. *FIRST* uses tags to track SDK versions through a standard semantic versioning naming scheme. When a new SDK version is released, the FTC engineering team pushes a release candidate branch to FIRST-Tech-Challenge/FtcRobotController, then merges that branch into master. This results in two commits, the new SDK version commit that contains all the good stuff, and a merge commit representing the merge from the candidate branch into master. The release is then formally cut, where a tag is then created, on the merge commit.

Tags from remotes are not automatically copied into a repository on a clone. To retrieve tags execute.

```
$ git fetch --all --tags
```

The –all option fetches at once from all remotes, the –tags option tells git to fetch the tags. Tags always follow the semantic versioning rules. e.g. v7.0, v7.1, v7.2, v8.0, etc.

The ^ syntax allows one to reference parents of a commit and can be applied to tag names. tag^ is the immediate parent of the commit tag points to. For commits with multiple parents such as merge commits one can apply a number to refer to a specific parent. tag^1 is the same as tag^ and is the first parent of the commit, tag^2 is the second parent of the commit.

Merging the Inverse of an SDK Update

The diagram below shows the v8.0 tag pointing to the v8.0 merge commit along with references to the parents of v8.0.

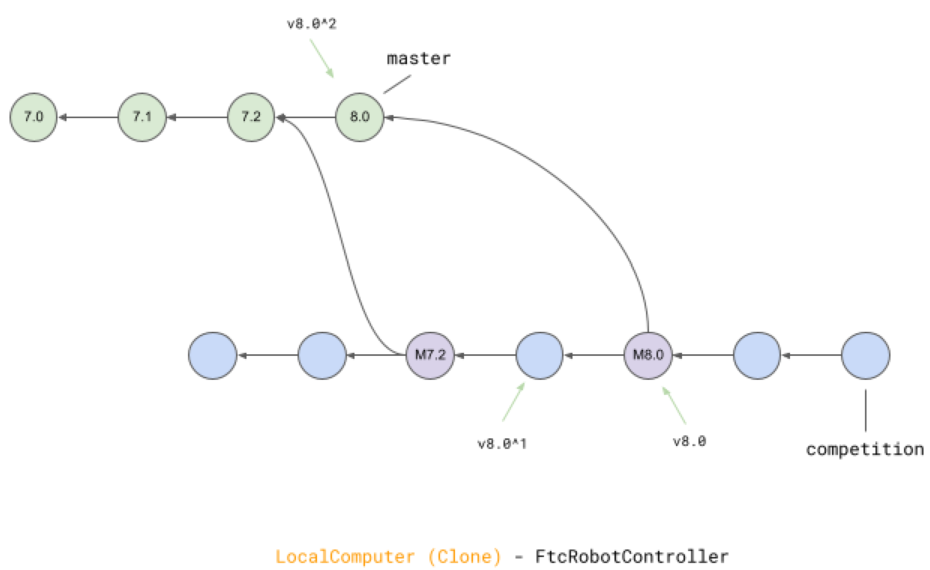


Fig. 25: v8.0 tag pointing to the v8.0 merge commit.

Important: If any commits have dependencies on new features or APIs introduced in the reverted versions, then your build will break. You will have to manually figure out how to fix your software so that it is no longer depends upon reverted software.

Remember that Git does not delete commits (with a few exceptions), so in order to revert a commit we must create a new commit that is the inverse of the commit you want to revert *from*. And you'll want to do this for every version, in reverse order, that you want to undo. The target of the command below is the tag of the version you want to undo, not the tag of the version you want to revert to.

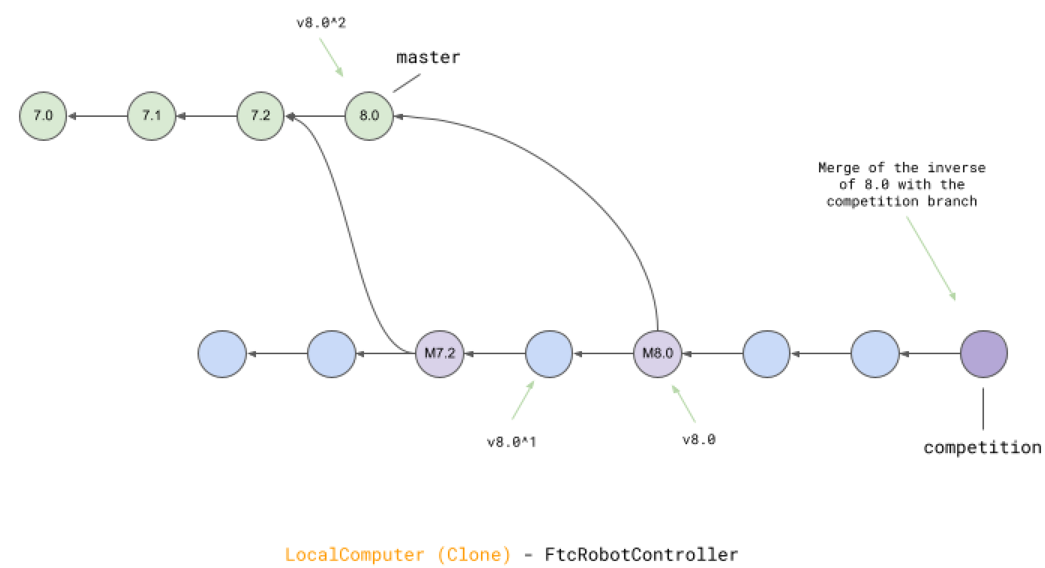


Fig. 26: Result of revert - a new merge commit representing the revert from v8.0 to v7.2.

Because the merge commit has two parents, and you want to reference the SDK version commit, use the tag name you want to roll back and append ^2. For example to roll back v8.0, resulting in the SDK compiling against v7.2 use.

```
$ git revert -Xtheirs v8.0^2
```

The -Xtheirs option is a convenience that says, "If there are any conflicts, automatically take the software from the v8.0^2 side."

Warning: If you want to downgrade more than one revision you must revert each revision in sequence otherwise you could wind up with changes remaining after reversion from the SDK version in between latest and the target you are referring to. For example if you need to downgrade from v8.1.1 to v8.0, for reference all SDK versions can be found here, you must revert v8.1.1 followed by v8.1. If you don't follow this order, then changes in v8.1.1 that don't overlap with v8.1 will remain in your workspace and that's not what you want.

Summary

Assumes all commands are run from the root directory of your local clone. Also assumes you are not committing team code to your local master branch, but instead are working in a competition branch.

Add FIRST-Tech-Challenge/FtcRobotController as a remote

```
$ git remote add upstream https://github.com/FIRST-Tech-Challenge/FtcRobotController.git
```

Update the to latest SDK version

```
$ git checkout master
$ git fetch upstream
$ git merge upstream/master
$ git push origin master
$ git checkout competition
$ git merge master
```

Writing an Op Mode AS

Enabling Developer Options AS

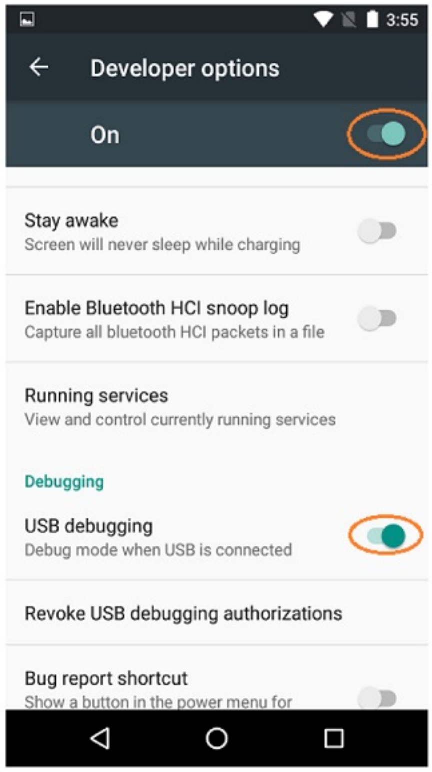
After you have configured your Android phone, you will also have to make sure that your phone is in developer mode before you will be able to install apps onto the phone using the tools that are included with Android Studio.

Important: Control Hub Users - The Control Hub has Developer Options automatically enabled from the factory, so you do **NOT** need to do this step for your Control Hub.

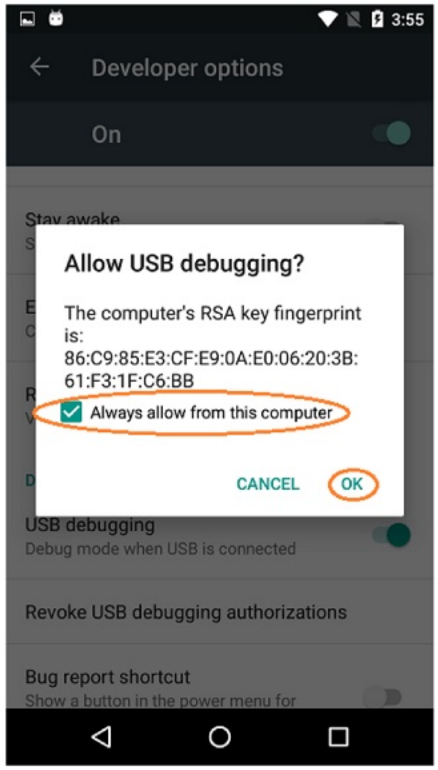
The Android Developer website contains information on how to enable Developer Options onto your phone. If you visit the following link and read the section entitled "Enabling On-device Developer Options" you will see that you can enable Developer Options on your Android phone by going to Settings->About phone on the phone, and then tapping the Build number seven times.

- https://developer.android.com/studio/run/device#setting-up

In order to be able to use the Android Studio tools to install apps onto your phone, you will need to make sure that the Developer Options and USB debugging are enabled for both of your phones.



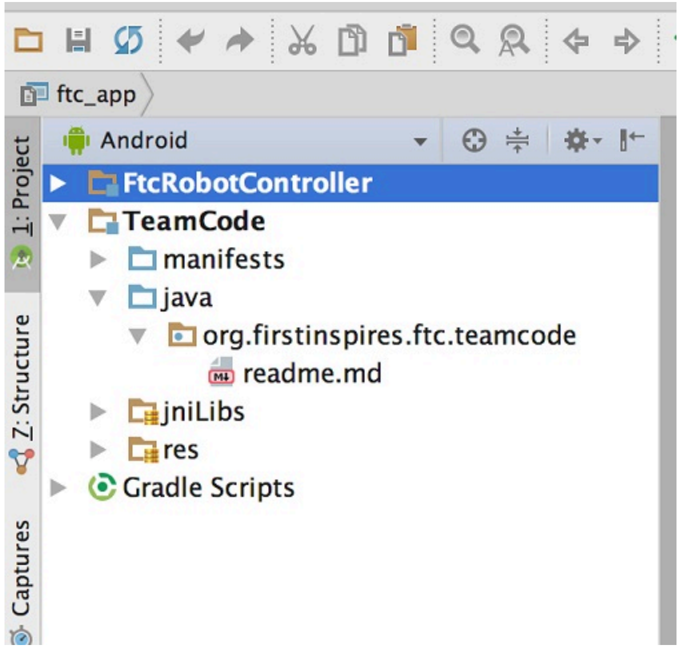
When you first connect a phone to your computer with Android Studio running, the phone might prompt you if it is OK to allow the computer to have USB debugging access to the phone. If this happens, make sure that you check the "Always allow from this computer" option and hit the OK button to allow USB debugging.



Creating and Running an OpMode AS

TeamCode Module

If you successfully imported the Android Studio project folder, you will see on the project browser an Android module named `TeamCode`. The Android Studio project folder will be used to build a version of the Robot Controller app that includes the custom OpMode that you will write to control your competition robot.



When you create your classes and OpModes, you will to create them in the `org.firstinspires.ftc.teamcode` package that resides in the TeamCode module. This package is reserved for your use within the Android Studio project folder.

Javadoc Reference Information

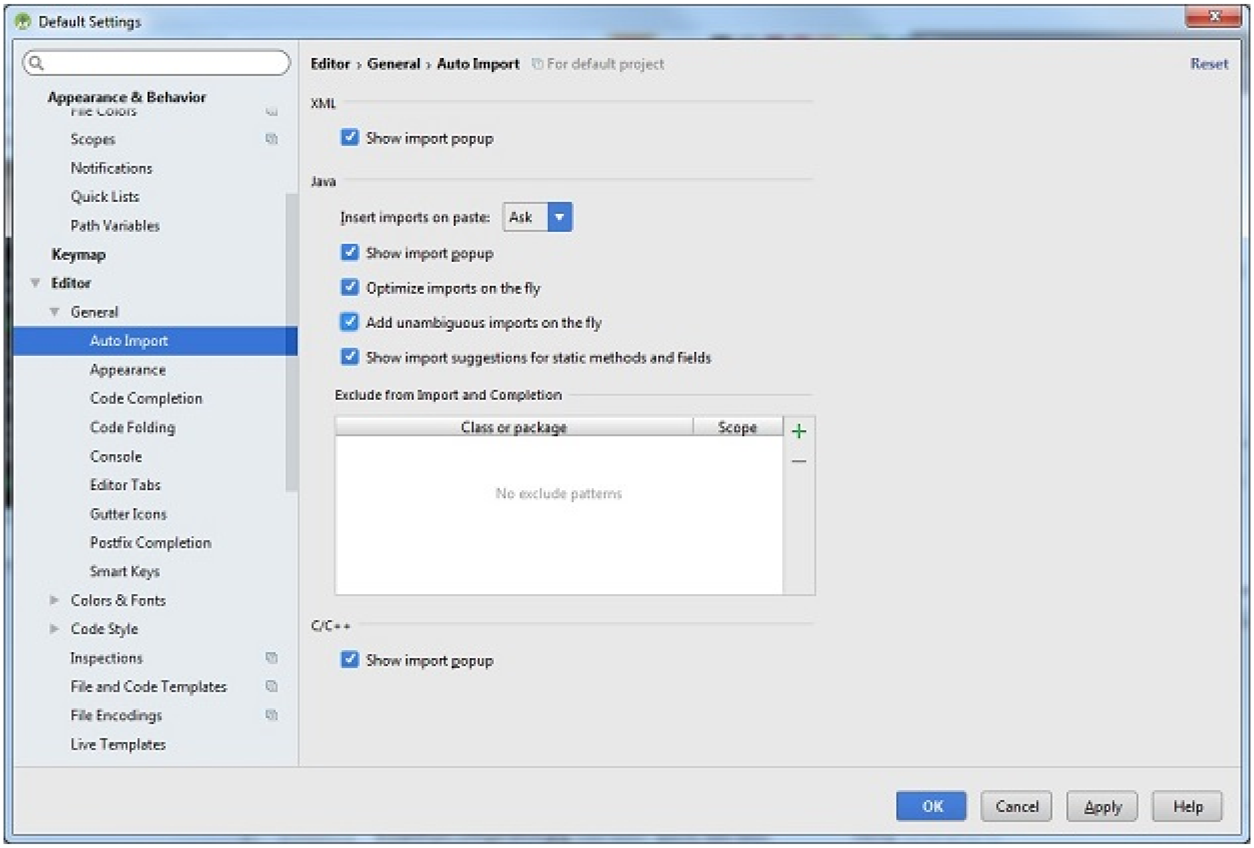
The Javadoc reference documentation for the SDK is available online. Visit the following URL to view the SDK documentation:

- https://javadoc.io/doc/org.firstinspires.ftc

Enabling Auto Import

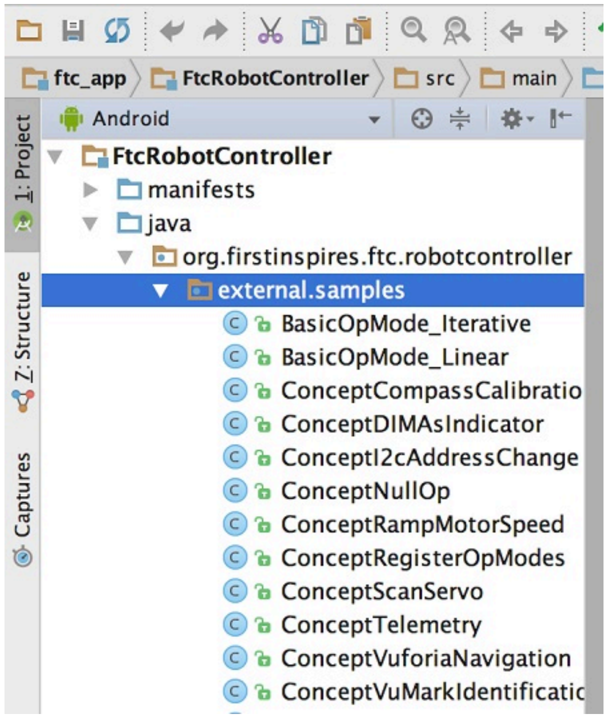
The auto import feature of Android Studio is a convenient function that helps save time as you write your OpMode. If you would like to enable this feature, select the Editor->General->Auto Import item from the Android Studio Settings screen. This will display the editor's auto import settings.

Check the "Add unambiguous imports on the fly" so that Android Studio will automatically add the required import statements for classes that you would like to use in your OpMode.



Sample OpModes

A great way to learn how to program a robot is to examine the sample op modes that are included with the Android Studio project folder. You can locate these files in the FtcRobotController module in the package `org.firstinspires.ftc.robotcontroller.external.samples`.



If you would like to use a sample OpMode, copy it from the `org.firstinspires.ftc.robotcontroller.external.samples` package and move it to the `org.firstinspires.ftc.teamcode` package.

In your newly copied OpMode, look for the following annotation,

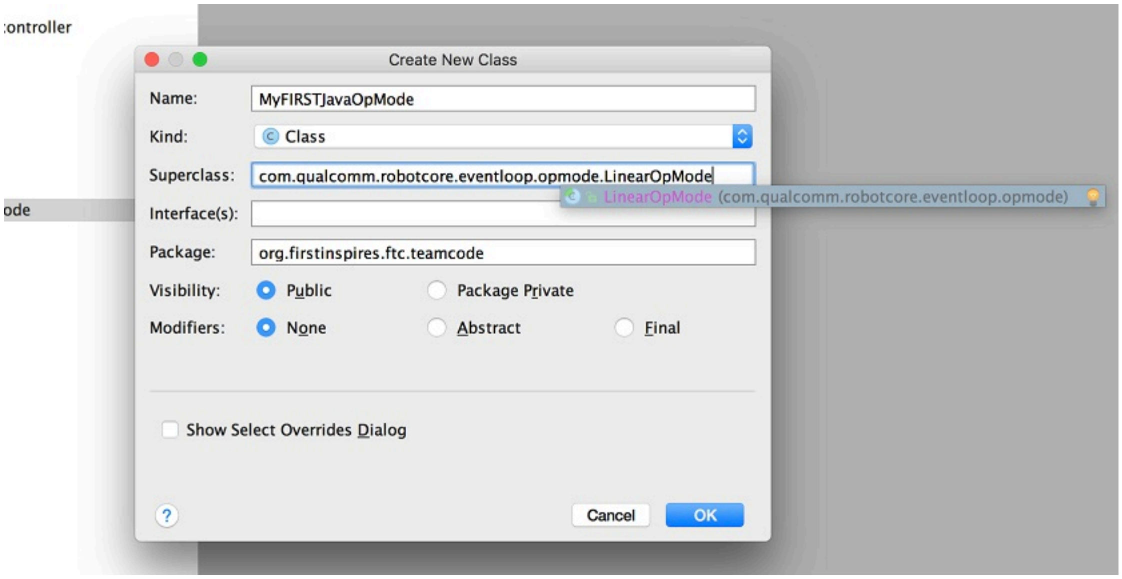
```
@Disabled
```

and comment out this line to enable the OpMode and allow it to be run on the Robot Controller:

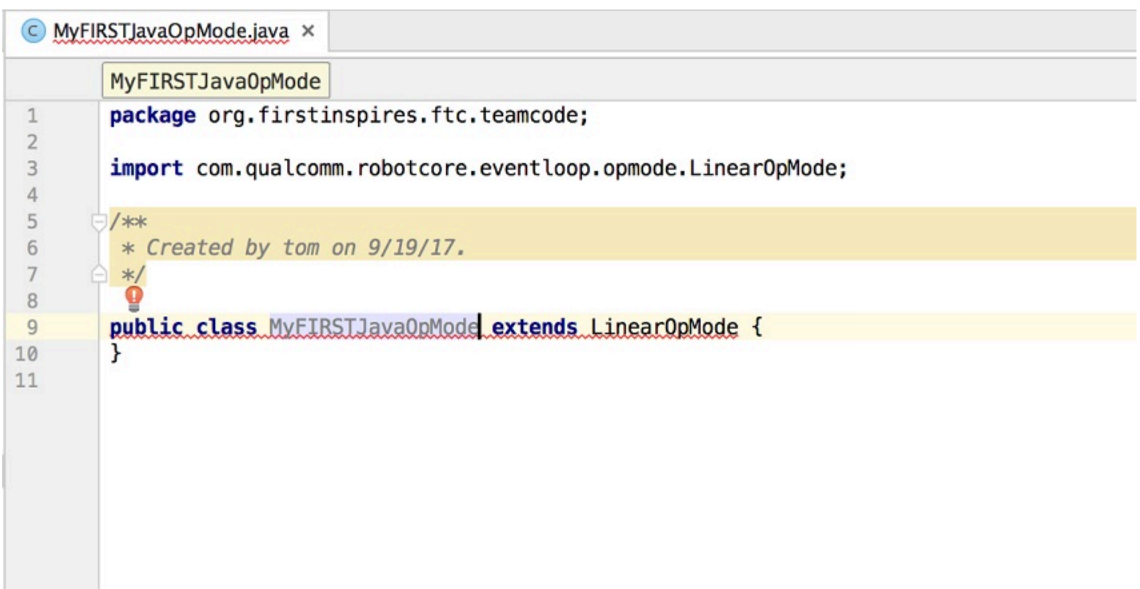
```
//@Disabled
```

Creating Your First OpMode

Right mouse click on the `org.firstinspires.ftc.teamcode` package and select New->Java Class from the pop-up menu. The Create New Class dialog box appear. Specify the name of the new class as `MyFIRSTJavaOpMode` and specify as its superclass the class LinearOpMode which is in the package `com.qualcomm.robotcore.eventloop.opmode`.



Press the OK button to create the new class. The source code for the new class should appear in the editing pane of the Android Studio user interface.



Modify the main portion of your OpMode so that it looks like the following code (note that the package definition and some import statements have been omitted in the following source code):

```
@TeleOp

public class MyFIRSTJavaOpMode extends LinearOpMode {
    private Gyroscope imu;
```

(continues on next page)

(continued from previous page)

```
    private DcMotor motorTest;
    private DigitalChannel digitalTouch;
    private DistanceSensor sensorColorRange;
    private Servo servoTest;


    @Override
    public void runOpMode() {
        imu = hardwareMap.get(Gyroscope.class, "imu");
        motorTest = hardwareMap.get(DcMotor.class, "motorTest");
        digitalTouch = hardwareMap.get(DigitalChannel.class, "digitalTouch");
        sensorColorRange = hardwareMap.get(DistanceSensor.class, "sensorColorRange");
        servoTest = hardwareMap.get(Servo.class, "servoTest");

        telemetry.addData("Status", "Initialized");
        telemetry.update();
        // Wait for the game to start (driver presses PLAY)
        waitForStart();

        // run until the end of the match (driver presses STOP)
        while (opModeIsActive()) {
            telemetry.addData("Status", "Running");
            telemetry.update();

        }
    }
}
```

We will use this source code as the framework for your first OpMode. Note that Android Studio automatically saves your source code as you are editing it.

Congratulations! You've written an OpMode. It does not do much, but we will modify it to make it more useful.

Examining the Structure of Your OpMode

It can be helpful to think of an OpMode as a list of tasks for the Robot Controller to perform. For a linear OpMode, the Robot Controller will process this list of tasks sequentially. Users can also use control loops (such as a while loop) to have the Robot Controller repeat (or iterate) certain tasks within a linear OpMode.

If you think about an OpMode as a list of instructions for the robot, this set of instructions that you created will be executed by the robot whenever a team member selects the OpMode called `MyFIRSTJavaOpMode` from the list of available OpModes for this Robot Controller.

Let's look at the structure of your newly created OpMode. Here's a copy of the OpMode text (minus some comments, the package definition, and some import package statements):

```
@TeleOp

public class MyFIRSTJavaOpMode extends LinearOpMode {
    private Gyroscope imu;
    private DcMotor motorTest;
```

(continues on next page)

(continued from previous page)

```
    private DigitalChannel digitalTouch;
    private DistanceSensor sensorColorRange;
    private Servo servoTest;


    @Override
    public void runOpMode() {
        imu = hardwareMap.get(Gyroscope.class, "imu");
        motorTest = hardwareMap.get(DcMotor.class, "motorTest");
        digitalTouch = hardwareMap.get(DigitalChannel.class, "digitalTouch");
        sensorColorRange = hardwareMap.get(DistanceSensor.class, "sensorColorRange");
        servoTest = hardwareMap.get(Servo.class, "servoTest");

        telemetry.addData("Status", "Initialized");
        telemetry.update();
        // Wait for the game to start (driver presses PLAY)
        waitForStart();

        // run until the end of the match (driver presses STOP)
        while (opModeIsActive()) {
            telemetry.addData("Status", "Running");
            telemetry.update();

        }
    }
}
```

At the start of the OpMode there is an annotation that occurs before the class definition. This annotation states that this is a tele-operated (i.e., driver controlled) OpMode:

`@TeleOp`

If you wanted to change this OpMode to an autonomous OpMode, you would replace the `@TeleOp` with an `@Autonomous` annotation instead.

You can see from the sample code that an OpMode is defined as a Java class. In this example, the OpMode name is called `MyFIRSTJavaOpMode` and it inherits characteristics from the LinearOpMode class.

```
public class MyFIRSTJavaOpMode extends LinearOpMode {
```

You can also see that the OnBot Java editor created five private member variables for this OpMode. These variables will hold references to the five configured devices that the OnBot Java editor detected in the configuration file of your Robot Controller.

```
private Gyroscope imu;
private DcMotor motorTest;
private DigitalChannel digitalTouch;
private DistanceSensor sensorColorRange;
private Servo servoTest;
```

Next, there is an overridden method called `runOpMode`. Every OpMode of type `LinearOpMode` must implement this method. This method gets called when a user selects and runs the OpMode.

```
@Override
public void runOpMode() {
```

At the start of the `runOpMode` method, the OpMode uses an object named `hardwareMap` to get references to the hardware devices that are listed in the Robot Controller's configuration file:

```
imu = hardwareMap.get(Gyroscope.class, "imu");
motorTest = hardwareMap.get(DcMotor.class, "motorTest");
digitalTouch = hardwareMap.get(DigitalChannel.class, "digitalTouch");
sensorColorRange = hardwareMap.get(DistanceSensor.class, "sensorColorRange");
servoTest = hardwareMap.get(Servo.class, "servoTest");
```

The `hardwareMap` object is available to use in the `runOpMode` method. It is an object of type `HardwareMap` class.

Note that when you attempt to retrieve a reference to a specific device in your OpMode, the name that you specify as the second argument of the `HardwareMap.get` method must match the name used to define the device in your configuration file. For example, if you created a configuration file that had a DC motor named `motorTest`, then you must use this same name (it is case sensitive) to retrieve this motor from the `hardwareMap` object. If the names do not match, the OpMode will throw an exception indicating that it cannot find the device.

In the next few statements of the example, the OpMode prompts the user to push the start button to continue. It uses another object that is available in the `runOpMode` method. This object is called telemetry and the OpMode uses the `addData` method to add a message to be sent to the Driver Station. The OpMode then calls the update method to send the message to the Driver Station. Then it calls the `waitForStart` method, to wait until the user pushes the start button on the driver station to begin the OpMode run.

```
telemetry.addData("Status", "Initialized");
telemetry.update();
// Wait for the game to start (driver presses PLAY)
waitForStart();
```

Note that all linear OpModes should have a `waitForStart` statement to ensure that the robot will not begin executing the OpMode until the driver pushes the start button.

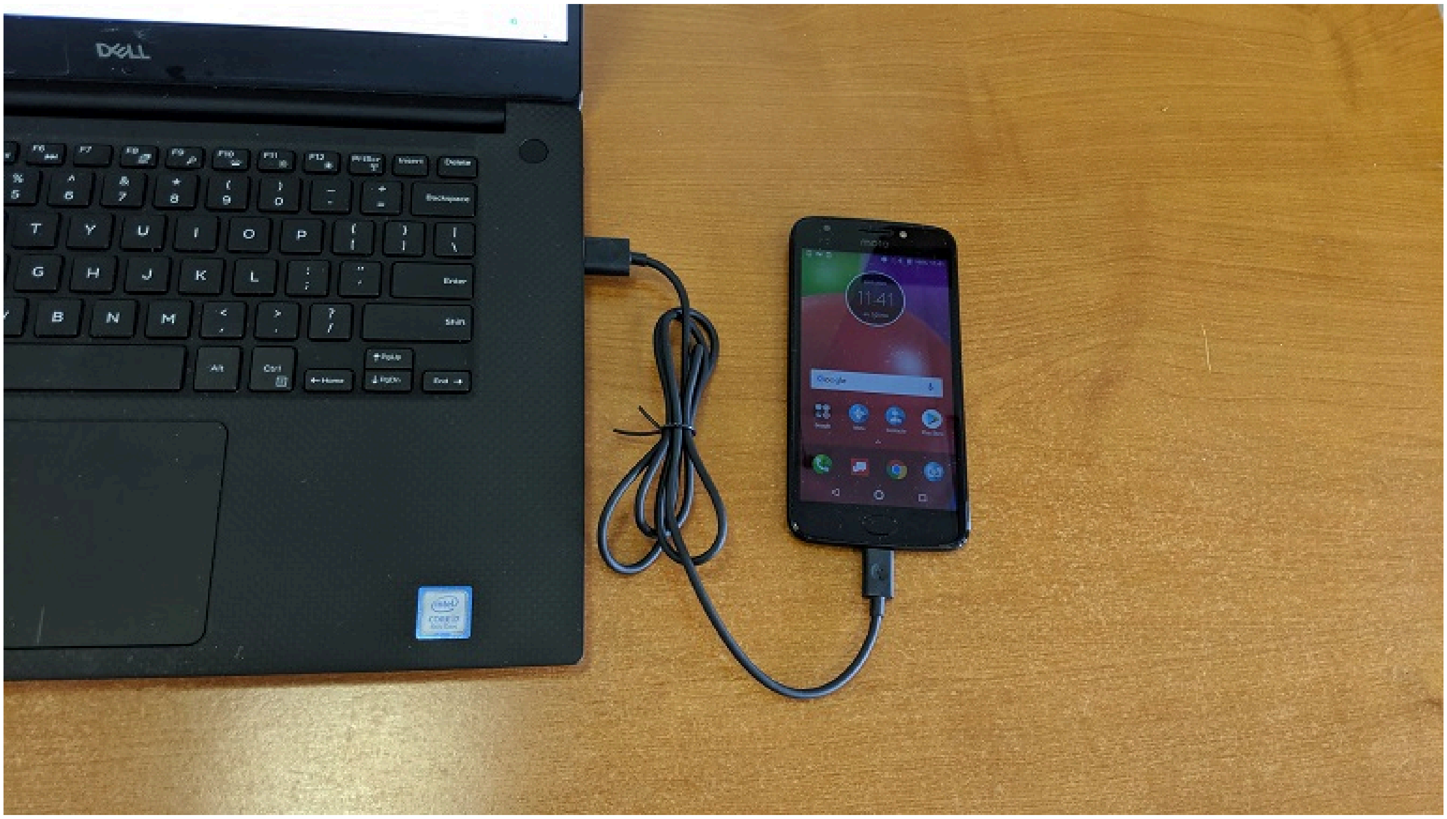
After a start command has been received, the OpMode enters a while loop and keeps iterating in this loop until the OpMode is no longer active (i.e., until the user pushes the stop button on the Driver Station):

```
// run until the end of the match (driver presses STOP)
while (opModeIsActive()) {
    telemetry.addData("Status", "Running");
    telemetry.update();

}
```

As the OpMode iterates in the while loop, it will continue to send telemetry messages with the index of “Status” and the message of “Running” to be displayed on the Driver Station.

Building and Installing Your OpMode

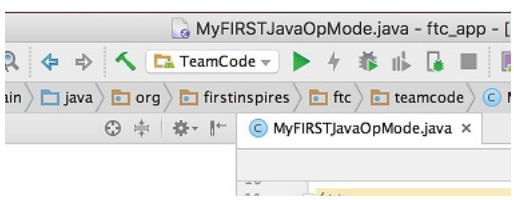
Verify that the Robot Controller phone is connected to your laptop and that the laptop has USB debugging permission for the phone.



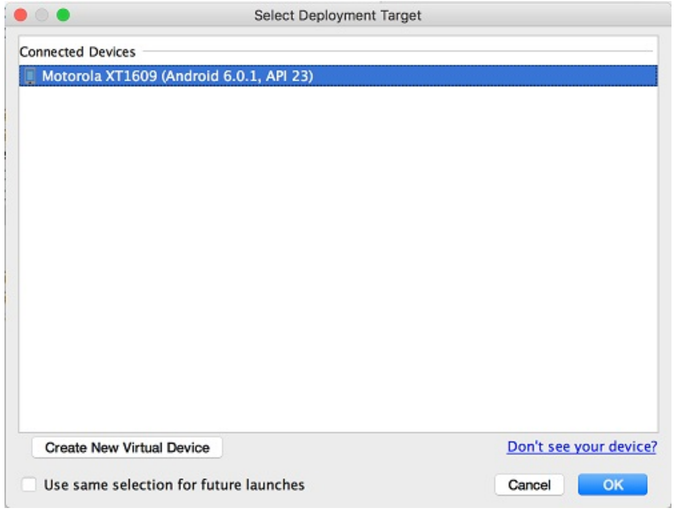
Or, if you are using a Control Hub, verify that the Control Hub is powered by a freshly charged 12V battery, and that it is connected to your laptop through its USB Type C port. Note that the Control Hub should automatically have USB debugging permission enabled.

When using the Control Hub, please make sure you use the Type C port (and not the USB Mini port) to connect the Control Hub to your development laptop.

Look towards the top of the Android Studio user interface and find the little green Play or Run button (which is represented by a green triangle) next to the words `Team Code`. Press this green button to build the Robot Controller app and to install it onto your phone.

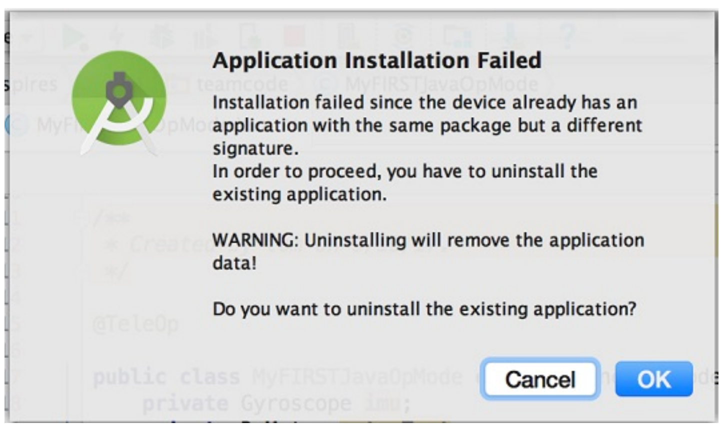


Android Studio should prompt you to select a target device to install the Robot Controller app. Your screen might look something like the image shown below.

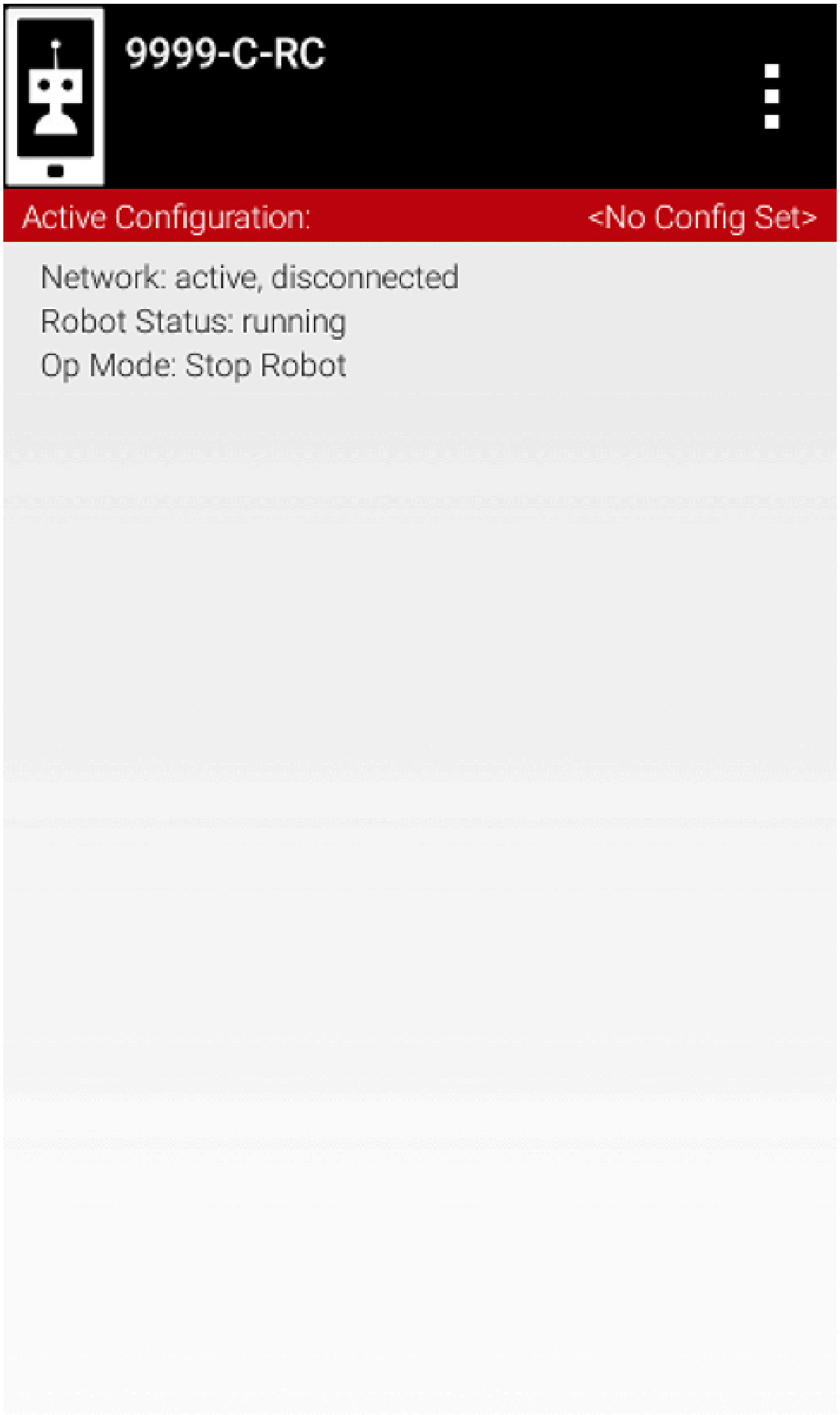


Make sure that you select the correct target device. In the figure above the Motorola phone is selected as the target device. Hit OK to build the APK file and install it on the target device.

Note that if you previously installed a copy of the Robot Controller app from the Google Play store, the installation of your newly built app will fail the first time you attempt to install it. This is because Android Studio detects that the app that you just build has a different digital signature than the official version of the Robot Controller app that was installed from Google Play.



If this happens, Android Studio will prompt you if it's OK to uninstall the previous (official) version of the app from your device and replace it with the updated version of the app. Select OK to uninstall the previous version and to replace it with your newly created Robot Controller App (see image above).



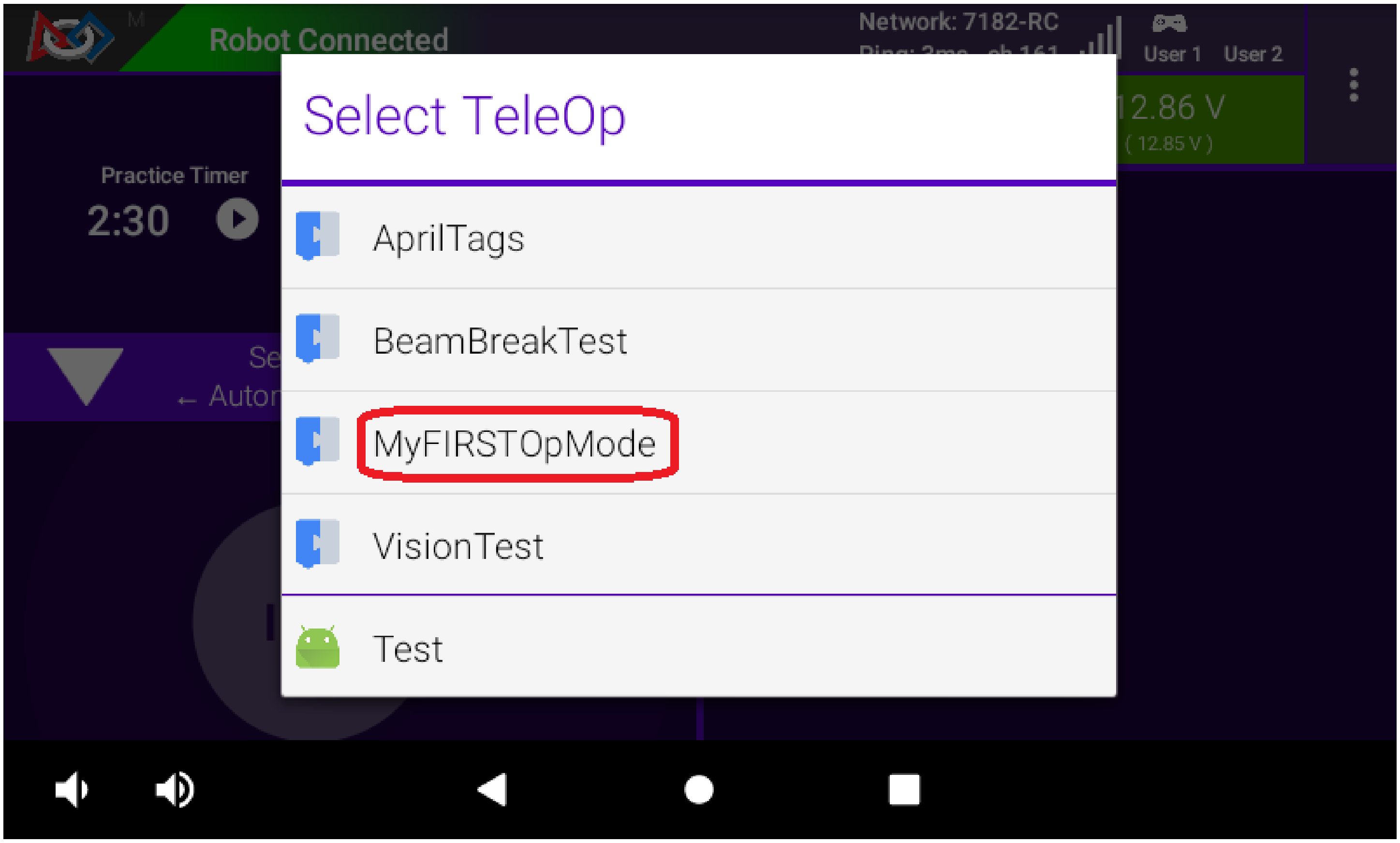
If the installation was successful, the Robot Controller app should be launched on the target Android device. If you are using an Android phone as your Robot Controller, you should see the main Robot Controller app screen displayed on the phone.

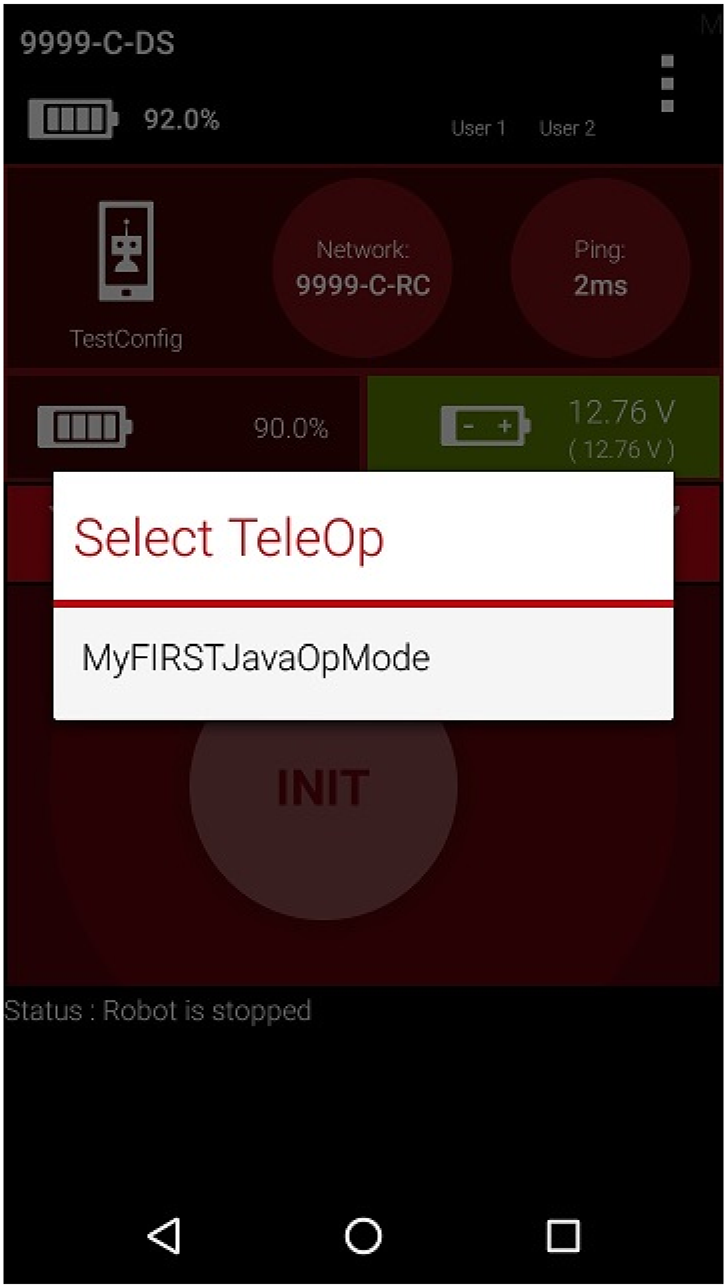
Although the Control Hub lacks a built in screen, if you are Control Hub user, you can verify that the app was installed onto your Control Hub properly by looking at your Driver Station. If the Driver Station indicates that it is successfully connected to the Control Hub (after momentarily disconnecting while the update was occurring) then the app was successfully updated.

Running Your OpMode

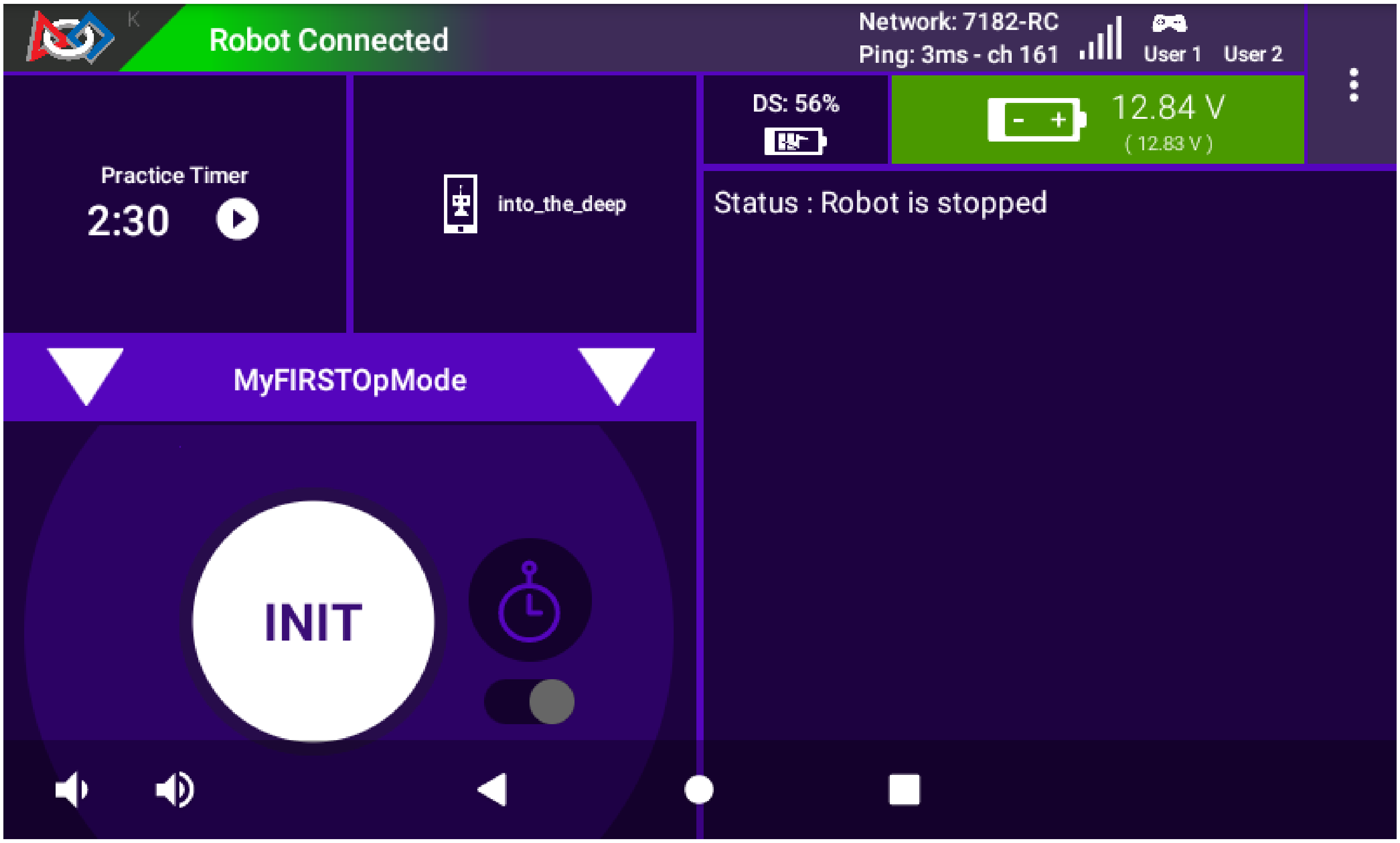
If you successfully built and installed your updated Android app with your new OpMode, then you are ready to run the OpMode. Verify that the Driver Station is still connected to the Robot Controller. Since you designated that your example OpMode is a tele-operated OpMode, it will be listed as a `TeleOp` OpMode.

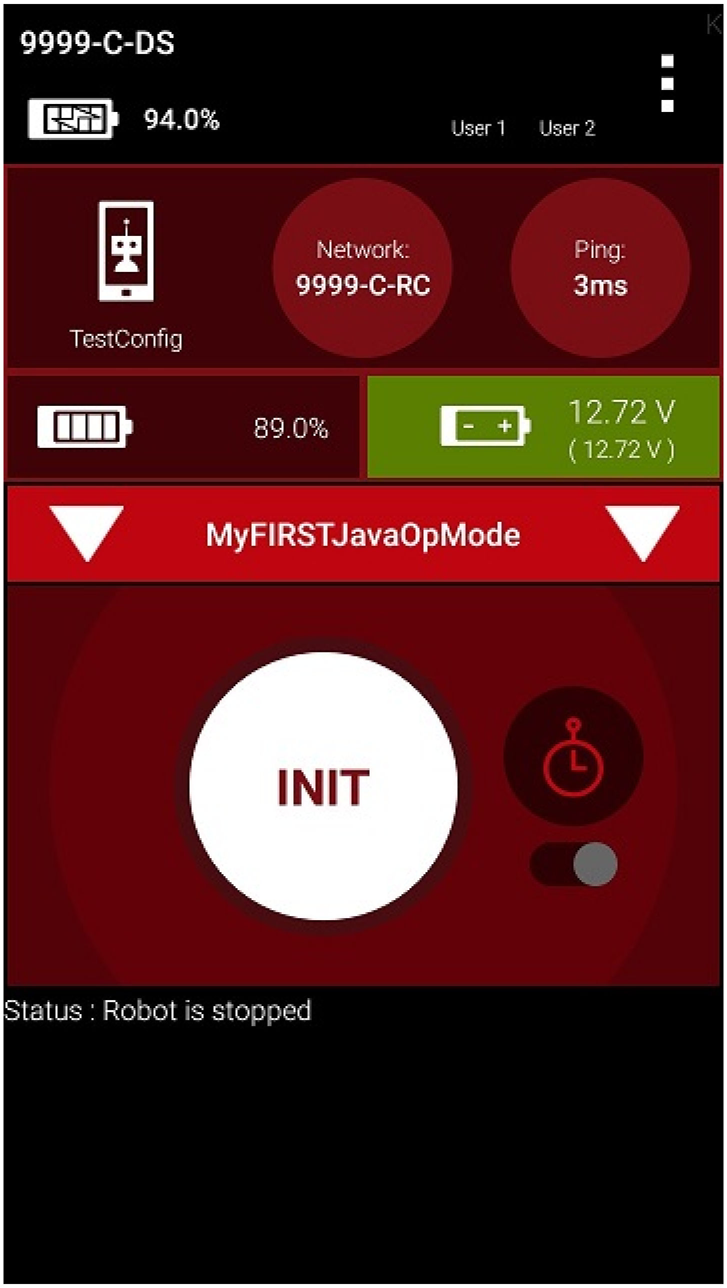
On the Driver Station, use the `TeleOp` dropdown list control to display the list of available OpModes. Select your OpMode ("MyFIRSTJavaOpMode") from the list.



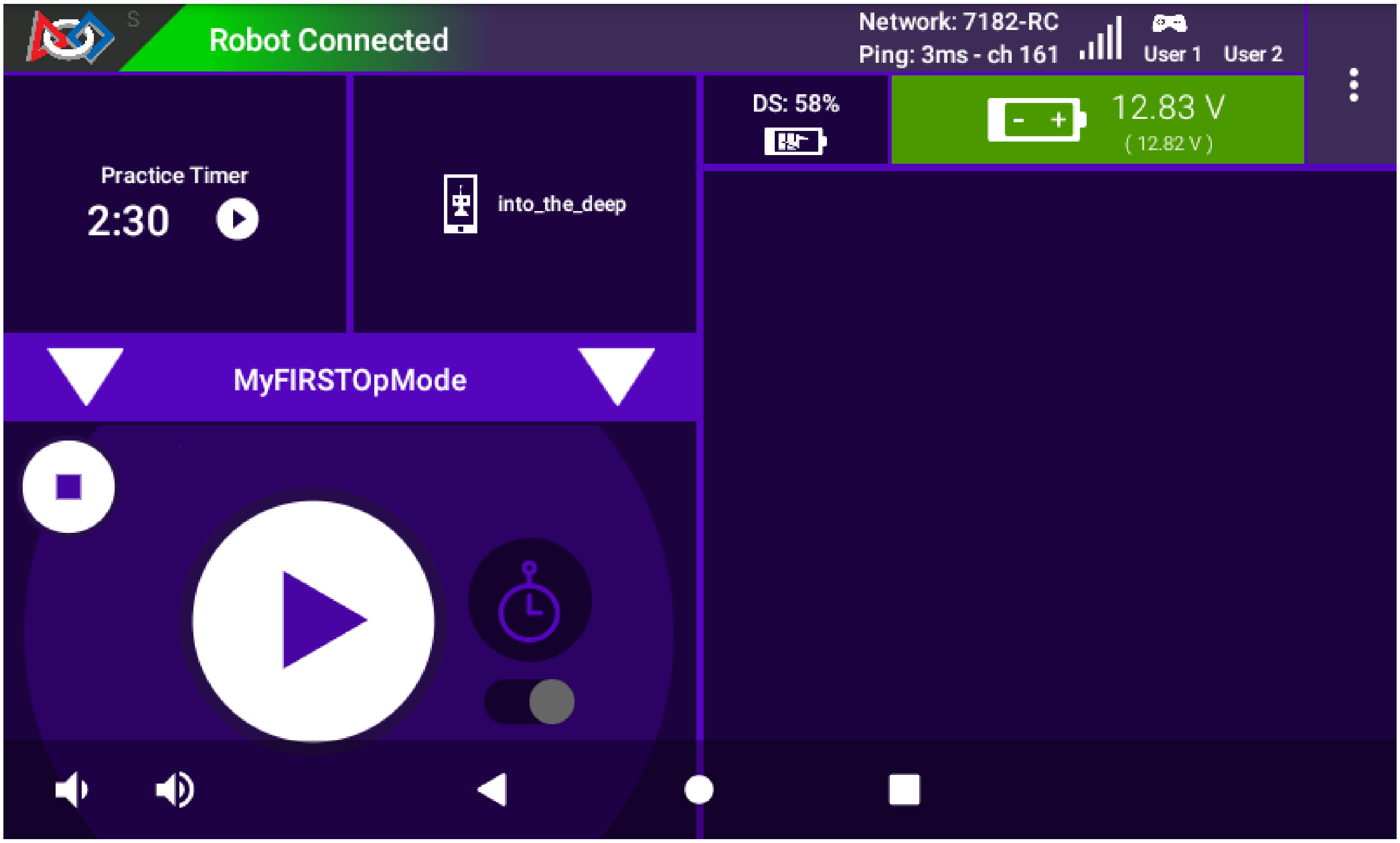


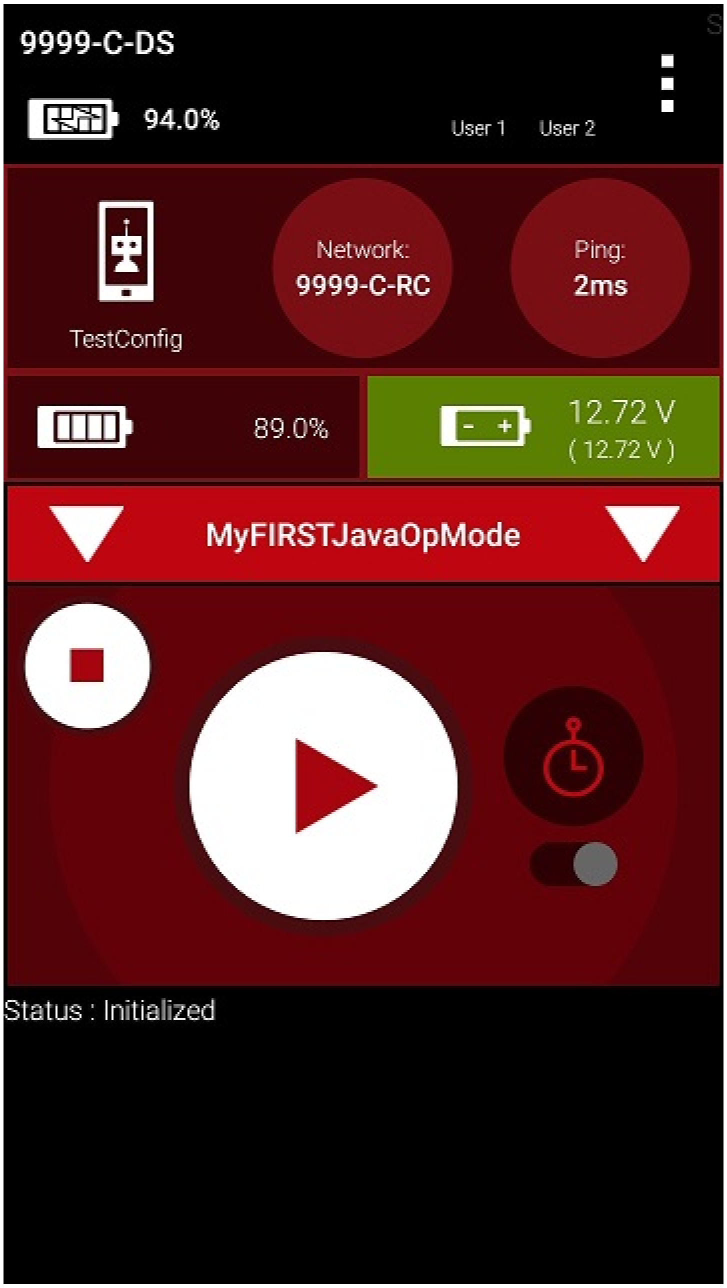
Press the "INIT" button to initialize the OpMode.


Robot Connected
Network: 7182-RC
Ping: 3ms - ch 161
User 1
User 2
DS: 56%
12.84 V
(12.83 V)
Practice Timer
2:30
into_the_deep
Status : Robot is stopped
MyFIRSTOpMode
INIT

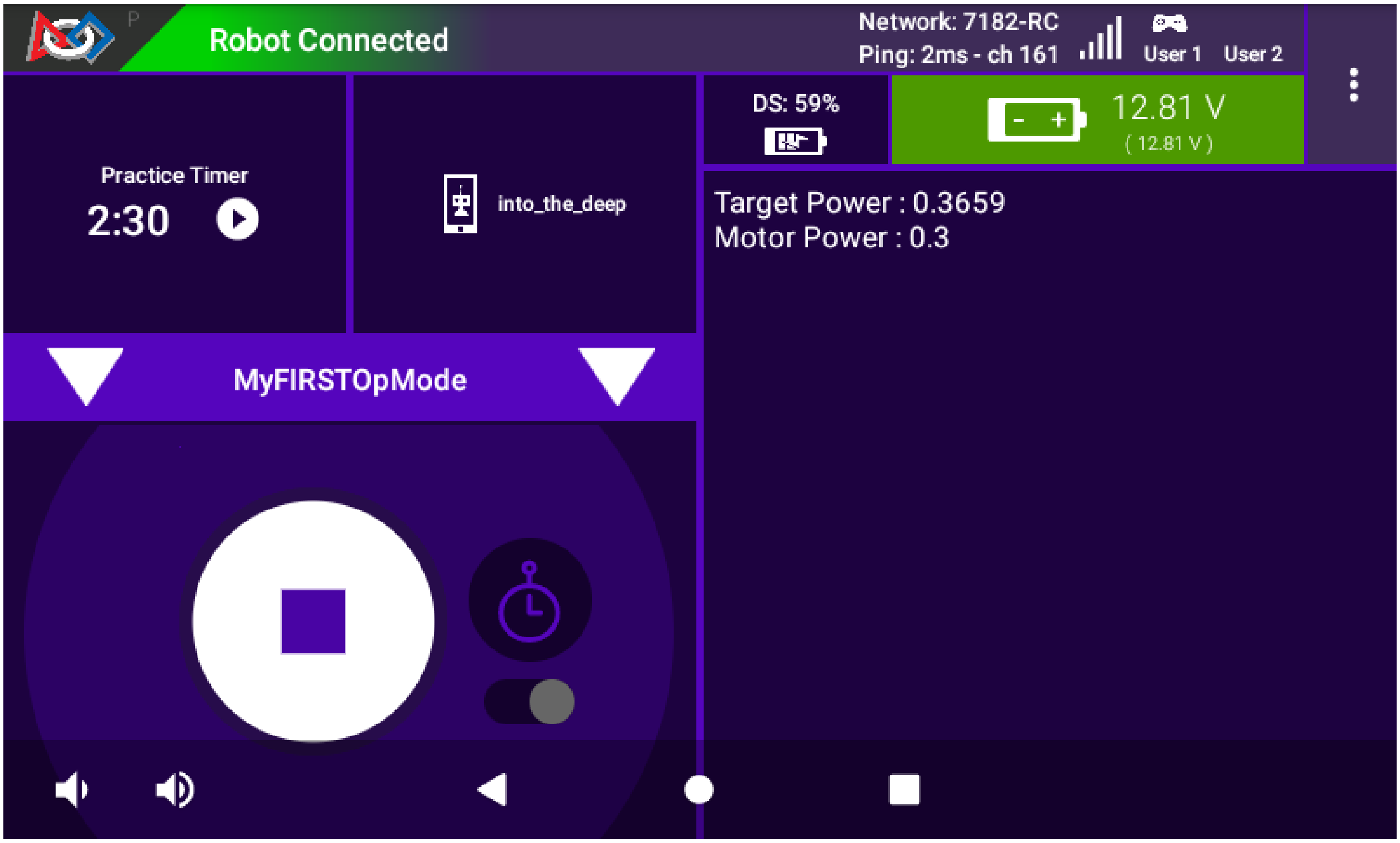


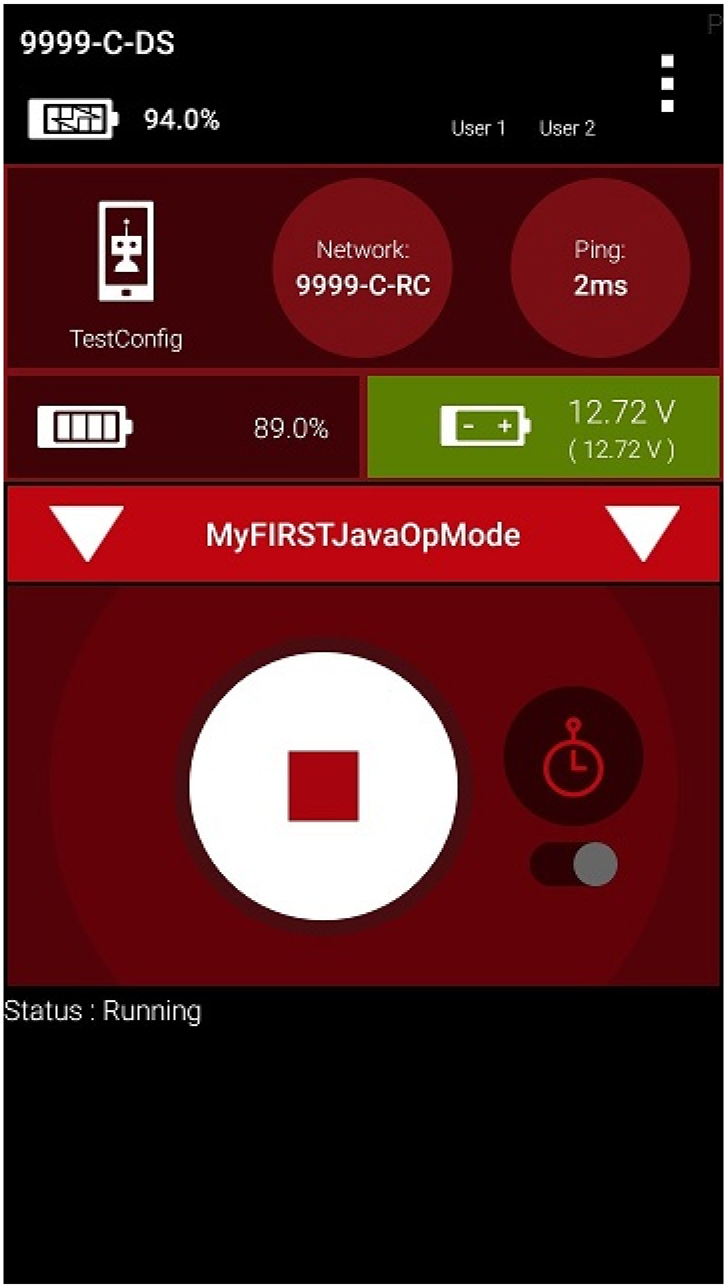
The OpMode will execute the statements in the runOpMode method up to the waitForStart statement. It will then wait until you press the start button (which is represented by the triangular shaped symbol) to continue.


Robot Connected
Network: 7182-RC
Ping: 3ms - ch 161
User 1
User 2
DS: 58%
12.83 V
(12.82 V)
Practice Timer
2:30
into_the_deep
MyFIRSTOpMode



Once you press the start button, the OpMode will continue to iterate and send the "Status: Running" message to the Driver Station. To stop the OpMode, press the square-shaped stop button.


Robot Connected
Network: 7182-RC
Ping: 2ms - ch 161
User 1
User 2
DS: 59%
12.81 V
(12.81 V)
Practice Timer
2:30
into_the_deep
Target Power : 0.3659
Motor Power : 0.3
MyFIRSTOpMode



Congratulations! You ran your first Java OpMode!

Modifying Your OpMode to Control a Motor

Let's modify your OpMode to control the DC motor that you connected and configured for your REV Robotics Control Hub or REV Robotics Expansion Hub. Modify the code for the program loop so that it looks like the following:

```
// run until the end of the match (driver presses STOP)
double tgtPower = 0;
while (opModeIsActive()) {
    tgtPower = -this.gamepad1.left_stick_y;
    motorTest.setPower(tgtPower);
    telemetry.addData("Target Power", tgtPower);
    telemetry.addData("Motor Power", motorTest.getPower());
    telemetry.addData("Status", "Running");
    telemetry.update();

}
```

If you look at the code that was added, you will see that we defined a new variable called target power before we enter the while loop.

```
double tgtPower = 0;
```

At the start of the while loop we set the variable tgtPower equal to the negative value of the gamepad1's left joystick:

```
tgtPower = -this.gamepad1.left_stick_y;
```

The object `gamepad1` is available for you to access in the `runOpMode` method. It represents the state of gamepad #1 on your Driver Station. Note that for the F310 gamepads that are used during the competition, the Y value of a joystick ranges from -1, when a joystick is in its topmost position, to +1, when a joystick is in its bottommost position. In the example code above, you negate the `left_stick_y` value so that pushing the left joystick forward will result in a positive power being applied to the motor. Note that in this example, the notion of forwards and backwards for the motor is arbitrary. However, the concept of negating the joystick y value can be very useful in practice.

The next set of statements sets the power of motorTest to the value represented by the variable tgtPower. The values for target power and actual motor power are then added to the set of data that will be sent via the telemetry mechanism to the Driver Station.

```
tgtPower = -this.gamepad1.left_stick_y;
motorTest.setPower(tgtPower);
telemetry.addData("Target Power", tgtPower);
telemetry.addData("Motor Power", motorTest.getPower());
```

After you have modified your OpMode to include these new statements, press the build button and verify that the OpMode was built successfully.

Running Your OpMode with a Gamepad Connected

Your OpMode takes input from a gamepad and uses this input to control a DC motor. To run your OpMode, you will need to connect a Logitech F310 or other approved gamepad to the Driver Station.

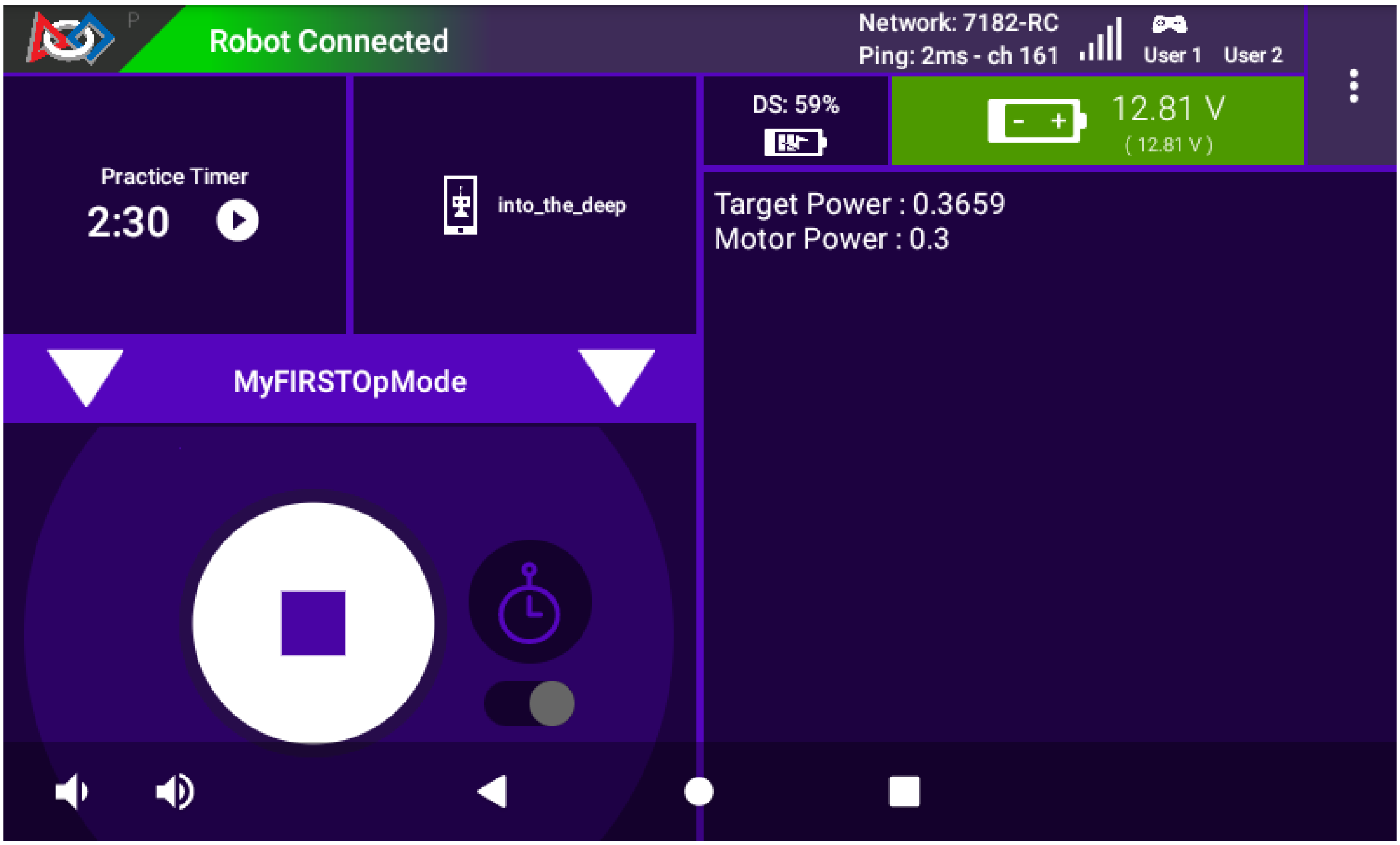
Connect the gamepad to the Driver Station. If using a REV Robotics Driver Hub, you can directly connect the gamepad to one of the USB-A ports. On a DRIVER STATION phone, you will need a Micro USB OTG adapter cable.

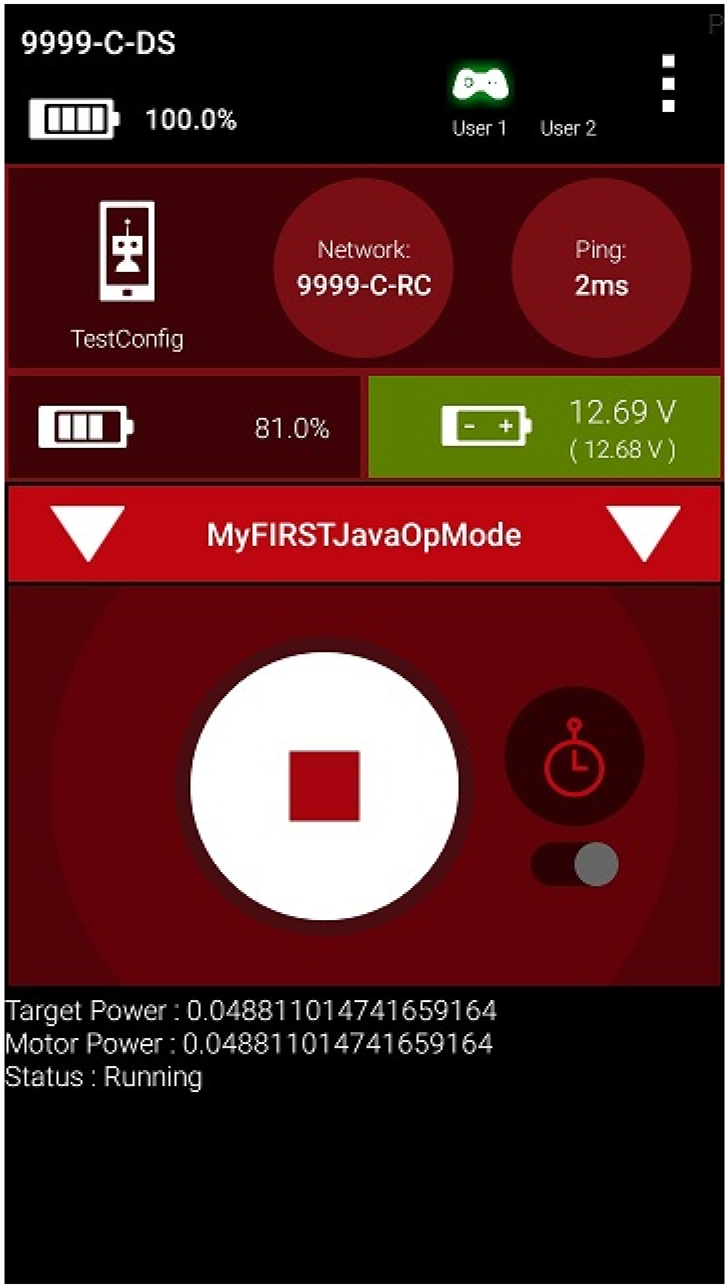
Your example OpMode is looking for input from the gamepad designated as the user or driver #1. Press the Start button and the A button simultaneously on the Logitech F310 controller to designate your gamepad as user #1. Note that pushing the Start button and the B button simultaneously would designate the gamepad as user #2. On a PS4-style gamepad, use the Options button and Cross for user #1, or Options and Circle for user #2.

If you successfully designated the gamepad to be user #1, you should see a little gamepad icon above the text "User 1" in the upper right hand corner of the Driver Station Screen. Whenever there is activity on gamepad #1, the little icon should be highlighted in green. If the icon is missing or if it does not highlight in green when you use your gamepad, then there is a problem with the connection to the gamepad.

Select, initialize and run your `MyFIRSTJavaOpMode` OpMode.

If you configured your gamepad properly, then the left joystick should control the motion of the motor. As you run your OpMode, be careful and make sure you do not get anything caught in the turning motor. Note that the User #1 gamepad icon should highlight green each time you move the joystick. Also note that the target power and actual motor power values should be displayed in the telemetry area on the Driver Station.




9999-C-DS
100.0%
User 1
User 2
TestConfig
Network:
9999-C-RC
Ping:
2ms
81.0%
12.69 V
(12.68 V)
MyFIRSTJavaOpMode
Target Power : 0.048811014741659164
Motor Power : 0.048811014741659164
Status : Running

Controlling a Servo AS

In this section, you will modify your op mode to control a servo motor with the buttons of the gamepad.

What is a Servo Motor?

A servo motor is a special type of motor. A servo motor is designed for precise motion. A typical servo motor has a limited range of motion.

In the figure below, "standard scale" 180-degree servo is shown. This type of servo is popular with hobbyists and with FIRST Tech Challenge teams. This servo motor can rotate its shaft through a range of 180 degrees. Using an electronic module known as a servo controller you can write an op mode that will move a servo motor to a specific position. Once the motor reaches this target position, it will hold the position, even if external forces are applied to the shaft of the servo.

Servo motors are useful when you want to do precise movements (for example, sweep an area with a sensor to look for a target or move the control surfaces on a remotely controlled airplane).

Modifying Your Op Mode to Control a Servo

Let's modify your op mode to add the logic required to control a servo motor. For this example, you will use the buttons on the Logitech F310 gamepad to control the position of the servo motor.

With a typical servo, you can specify a target position for the servo. The servo will turn its motor shaft to move to the target position, and then maintain that position, even if moderate forces are applied to try and disturb its position.

For the FIRST Tech Challenge control system, you can specify a target position that ranges from 0 to 1 for a servo. A target position of 0 corresponds to zero degrees of rotation and a target position of 1 corresponds to 180 degrees of rotation for a typical servo motor.

In this example, you will use the colored buttons on the right side of the F310 controller to control the position of the servo. Initially, the op mode will move the servo to the midway position (90 degrees of its 180-degree range). Pushing the yellow "Y" button will move the servo to the zero-degree position. Pushing the blue "X" button or the red "B" button will move the servo to the 90-degree position. Pushing the green "A" button will move the servo to the 180-degree position.

Modify your op mode to add the following code:

```
// run until the end of the match (driver presses STOP)
double tgtPower = 0;
while (opModeIsActive()) {
    tgtPower = -this.gamepad1.left_stick_y;
    motorTest.setPower(tgtPower);
    // check to see if we need to move the servo.
    if(gamepad1.y) {
        // move to 0 degrees.
        servoTest.setPosition(0);
    } else if (gamepad1.x || gamepad1.b) {
        // move to 90 degrees.
        servoTest.setPosition(0.5);
    } else if (gamepad1.a) {
        // move to 180 degrees.
        servoTest.setPosition(1);
    }
    telemetry.addData("Servo Position", servoTest.getPosition());
    telemetry.addData("Target Power", tgtPower);
```

(continues on next page)

(continued from previous page)

```
    telemetry.addData("Motor Power", motorTest.getPower());
    telemetry.addData("Status", "Running");
    telemetry.update();

}
```

This added code will check to see if any of the colored buttons on the F310 gamepad are pressed. If the Y button is pressed, it will move the servo to the 0-degree position. If either the X button or B button is pressed, it will move the servo to the 90-degree position. If the A button is pressed, it will move the servo to the 180-degree position. The op mode will also send telemetry data on the servo position to the Driver Station.

After you have modified your op mode, you can build it and then run it. Verify that gamepad #1 is still configured and then use the colored buttons to move the position of the servo.

Using Sensors AS

Color-Distance Sensor

A sensor is a device that lets the Robot Controller get information about its environment. In this example, you will use a REV Robotics Color-Distance sensor to display range (distance from an object) info to the driver station.

The Color-Range sensor uses reflected light to determine the distance from the sensor to the target object. It can be used to measure close distances (up 5" or more) with reasonable accuracy. Note that at the time this document was most recently edited, the REV Color-Range sensor saturates around 2" (5cm). This means that for distances less than or equal to 2", the sensor returns a measured distance equal to 2" or so.

Modify your op mode to add a telemetry statement that will send the distance information (in centimeters) to the Driver Station.

```
telemetry.addData("Servo Position", servoTest.getPosition());
telemetry.addData("Target Power", tgtPower);
telemetry.addData("Motor Power", motorTest.getPower());
telemetry.addData("Distance (cm)", sensorColorRange.getDistance(DistanceUnit.CM));
telemetry.addData("Status", "Running");
telemetry.update();
```

After you have modified your op mode, build and install the updated Robot Controller app, then run the op mode to verify that it now displays distance on your Driver Station. Note that if the distance reads "NaN" (short for "Not a Number") it probably means that your sensor is too far from the target (zero reflection). Also note that the sensor saturates at around 5 cm.

Touch Sensor

The REV Robotics Touch Sensor can be connected to a digital port on the Control Hub or Expansion Hub. The Touch Sensor is HIGH (returns TRUE) when it is not pressed. It is pulled LOW (returns FALSE) when it is pressed.

The Control Hub or Expansion Hub digital ports contain two digital pins per port. When you use a 4-wire JST cable to connect a REV Robotics Touch sensor to a Control Hub or Expansion Hub digital port, the Touch Sensor is wired to the second of the two digital pins within the port. The first digital pin of the 4-wire cable remains disconnected.

For example, if you connect a Touch Sensor to the "0,1" digital port of the Control Hub or Expansion Hub, the Touch Sensor will be connected to the second pin (labeled "1") of the port. The first pin (labeled "0") will stay disconnected.

Modify the code in your op mode that occurs before the waitForStart command to set the digital channel for input mode.

```
// set digital channel to input mode.
digitalTouch.setMode(DigitalChannel.Mode.INPUT);

telemetry.addData("Status", "Initialized");
telemetry.update();
// Wait for the game to start (driver presses PLAY)
waitForStart();
```

Also, modify the code in your while loop to add an if-else statement that checks the state of the digital input channel. If the channel is LOW (false), the touch sensor button is pressed and being pulled LOW to ground. Otherwise, the touch sensor button is not pressed.

```
// is button pressed?
if (digitalTouch.getState() == false) {
    // button is pressed.
    telemetry.addData("Button", "PRESSED");
} else {
    // button is not pressed.
    telemetry.addData("Button", "NOT PRESSED");
}

telemetry.addData("Status", "Running");
telemetry.update();
```

Build and install the updated Robot Controller app, then reinitialize and restart your op mode. The op mode should now display the state of the button ("PRESSED" or "NOT PRESSED").

21.2 Supporting Documentation

Control System Supporting Documentation

- *Control System Introduction*
- *Required Materials*
- *Using Your Android Device*
- *Phone Pairing*
- *Configuring Your Android Devices*
- *Connecting Devices to a Control or Expansion Hub*
- *Configuring Your Hardware*
- *Connecting a Laptop to a Program & Manage Wi-Fi Network*
- *Installing a Javascript Enabled Browser*
- *Managing a Control Hub*
- *Managing a Smartphone Driver Station (DS)*
- *Managing a Smartphone Robot Controller (RC)*

21.2.1 Phone Pairing

Introduction

The recent generation of apps (8.0, 8.1.1, and newer) are extremely reliable for pairing, including between **all models of legal phones**.

When the Android phones have been suitably prepared, pairing via Wi-Fi Direct is **fast** and usually **automatic**. Here is a procedure that addresses various **pre-existing conditions** that can impede pairing.

This article does not cover the REV Control Hub or REV Driver Hub.

Legal Phones

As of CENTERSTAGE presented by Raytheon Technologies in 2023-2024, these are the legal phones: - Motorola Moto G4 Play (XT1607, XT1609) - Motorola Moto G5 - Motorola Moto G5 Plus - Motorola Moto E4 (XT1765, XT1765PP, XT1766, XT1767) - Motorola Moto E5 (XT1920) - Motorola Moto E5 Play (XT1921)

Note that only Motorola Moto G4 Play models that have been updated to Android 7 (Nougat) are legal to use, as the minimum Android version is Android 7.0 - unfortunately Motorola no longer supports Over-The-Air updates for the Motorola Moto G4 Play, so there is no automatic way to update the smartphone. Depending on the exact model of the phone, the Motorola Rescue and Smart Assistant Tool **may** be able to update your device, however there are no guarantees.

Phone Cleanup and Prep

1. On RC phone: if needed, select Settings/Accounts/Google/select/3 dots/Remove account/confirm. Repeat for any other accounts. Also remove any non-*FIRST* Tech Challenge apps/games that might run in the background or attempt updates.
2. On RC phone: force quit (swipe away) all apps, including the RC app.
3. RC phone, Apps/Settings/Wi-Fi. Manually select and Forget any saved Networks.
4. RC phone, still in WiFi menu: navigate to Wi-Fi Direct menu (via More Settings or Advanced).

- Select and forget/disconnect any connections with Peer Devices, including the current phone pairing. This may take a few tries; OK to give up if disconnect not acknowledged.
  - If the top item shows 'Created Group', Disconnect it.
  - If you inadvertently create an Invitation pop-up on the other phone, Decline on the other phone and Cancel on this phone. In rare cases, the Invite prompt is underneath any open windows on the RC phone.
  - Pairing will be done later in the apps; see below.
- Select and Forget all Remembered Groups, including ANY phone pairings. (This can also be done from Advanced RC Settings from either app.) Your goal after steps d1 and d2: 'Not visible', no 'Peer devices', no 'Remembered groups'.
- If needed, Rename/Configure phone now to legal name, e.g. 12345-A-RC or 12345-RC. (This can also be done from Settings in each app.)
- Optional for Moto phones only: Configure device/Limit 2 devices, 'Inactivity timeout' Never, check box 'Auto connect remembered groups'. (Note: timeout is not persistent, re-check occasionally.)

5. Force quit to device home screen. Swipe down twice from top, do this in order:

- Airplane Mode ON
- Wi-Fi ON (usually toggles off when Airplane Mode is turned on), then Done
- Bluetooth OFF
- Location OFF, only for Android 7.x

6. repeat above steps on DS phone.

Pairing

1. On RC phone: open the current season's RC app. Check Self Inspect for any RC issues.
2. On DS phone: open the current season's DS app. Check Self Inspect for any DS issues.
3. On DS phone: Menu (3 dots)/Settings. Confirm 'Pairing Method' is Wi-Fi Direct. Open 'Pair with Robot Controller'. (Do not pair using phone/Android menu.)
4. Filter can remain on, be patient and wait for the app to find the matching device. Or turn off Filter to see all devices within a few seconds. Choose the corresponding RC phone, touch Back, and Back again to return to the DS home screen.
5. Look at RC phone, accept the Invitation there. In rare cases, the Invite prompt is underneath any open windows on the RC phone. Pairing will happen within seconds.

Summary

The above procedure may seem long, but it covers conditions that should not have been present in the first place. Going forward, pairing will be **fast and reliable – usually automatic**.

Questions, comments and corrections to westsiderobotics@verizon.net

21.2.2 Managing a Control Hub

Changing the Name

By default, the Control Hub has a name that begins with the phrase "FTC-" and ends with four characters that are assigned at the factory. In order to comply with game manual rule <RS01>, the name should be changed.

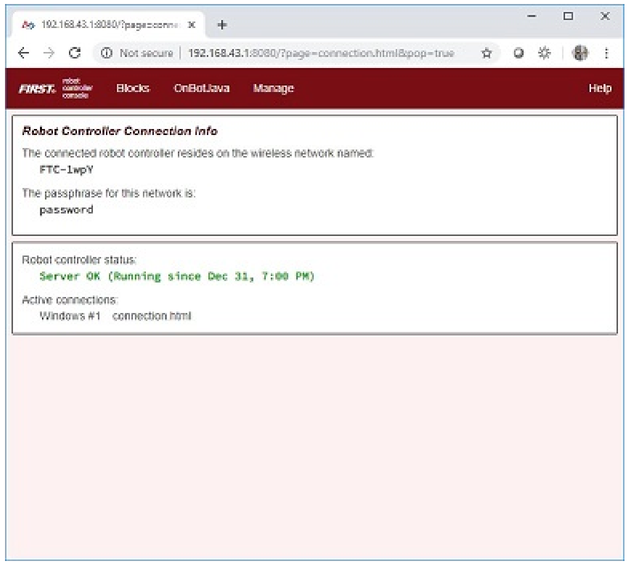
The name of a Control Hub (or Robot Controller phone) can be changed from a paired DS app, as shown in *Changing the Name*.

As an alternate, you can change the name of a Control Hub at the *Manage* page from a connected Driver Station or laptop, as described below. Click `Apply Wi-Fi Settings` when done.

Important: Changing the name of a Control Hub changes the name of the Hub's wireless network. Once the name is changed, you will have to connect your devices (Driver Station and programming laptop) to the new network.

Changing the Name of a Control Hub

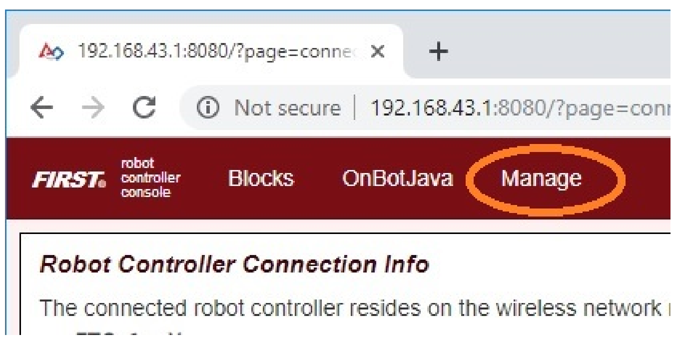
1. Verify that your laptop or Chromebook is connected to the Program & Manage wireless network of the Control Hub. If you are connected to the network, you should be able to see the *Robot Controller Connection Info* page when you navigate to address "192.168.43.1:8080":



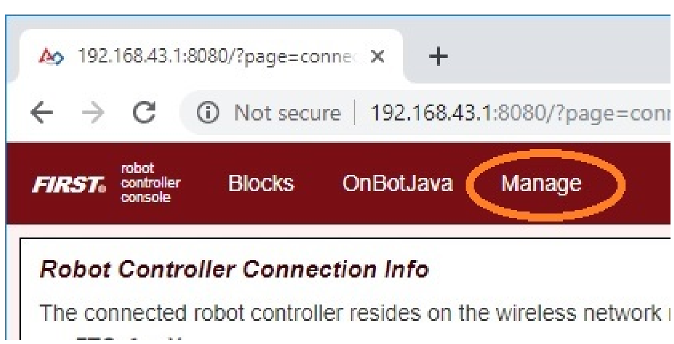
If your laptop or Chromebook is not connected and you are unable to access the *Robot Controller Connection Info* page, then read the instructions in the following tutorial to learn how to connect to the Program & Manage network.

Connecting a laptop to the Program & Manage Network

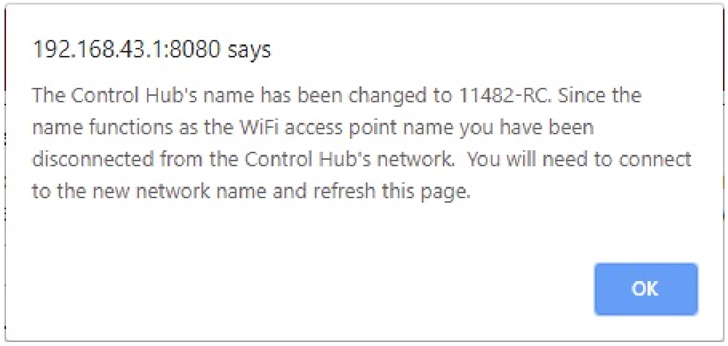
2. Click on the *Manage* link towards the top of the *Robot Controller Connection Info* page to navigate to the Manage page.



3. Change the name in the *Robot Controller Name* field and click on the *Change Name* button to change the Control Hub's name.



4. After you press the *Change Name* button, a dialog box will appear, indicating that the name has been changed and that you will need to connect to the new wireless network and refresh the current page.



Changing the Password

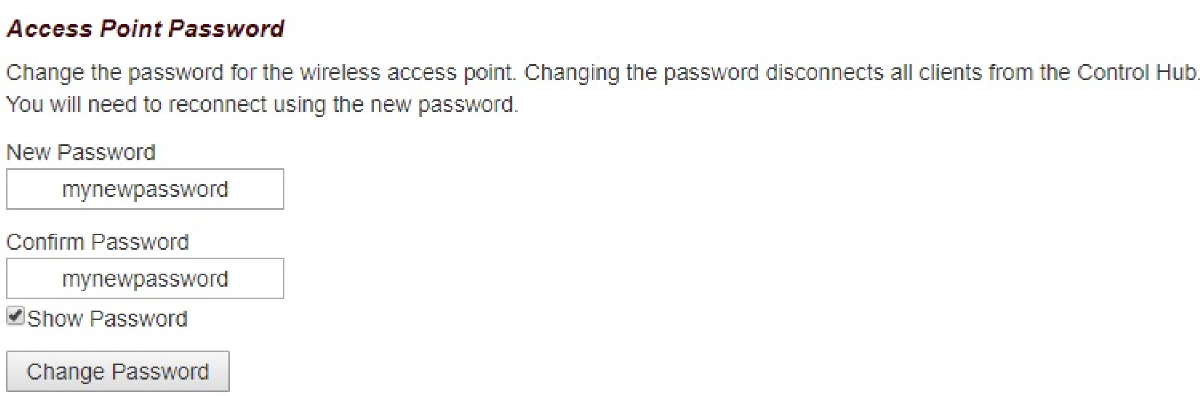
By default, the Control Hub has its password set to password at the factory. It is a good idea to change the password from its default value before you begin using your Control Hub.

You can change the password of a Control Hub using a laptop or Chromebook that is connected to the Hub's Program & Management page.

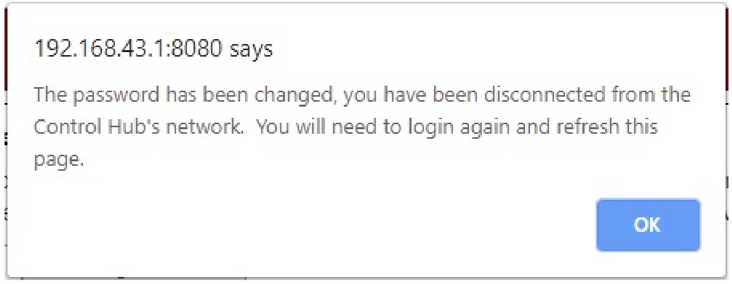
Warning: Commit your new password to memory or store it in a secure location so you will not forget it. You will need this password to manage and operate your Control Hub. Also note, once the password has been changed, you will have to reconnect your devices (Driver Station and programming laptop/Chromebook) to the network using the new password.

Changing the Password of a Control Hub

1. On the *Manage* page of the Control Hub user interface, find specify your new password and then confirm this new password in the *Access Point Password* section of the page. Press the *Change Password* to change the password.



2. After you press the *Change Password* button, a dialog box will appear, indicating that the password has been changed and that you will need to reconnect to the wireless network using the new password.



Resetting a Control Hub

If you forget the network name or password for a Control Hub, you can reset the Hub's name and password back to their factory default values.

Important: Resetting a Control Hub will restore its default network name and password. However, existing configuration files and op modes should not be affected by the reset. This includes op modes that were created using the Blocks, OnBot Java and Android Studio tools.

Resetting Instructions

1. Turn off the power to your Control Hub for 5 seconds.
2. Press and hold the button on the Control Hub (see image below).

3. Power on the Control Hub while continuing to hold the button. Monitor the LED while the Control Hub is rebooting. Eventually, the LED will switch from being solid blue, to a multi-color blink pattern.

   When the reset has started, the LED should blink purple, yellow, blue, and then red. This pattern should occur five times in rapid succession.

   Once the multi-colored blink pattern is complete, you can release the button. The Control Hub's network name and password should be restored to their factory values. Note that the reboot and reset process should take approximately 30 seconds to complete.

Changing the WiFi Channel

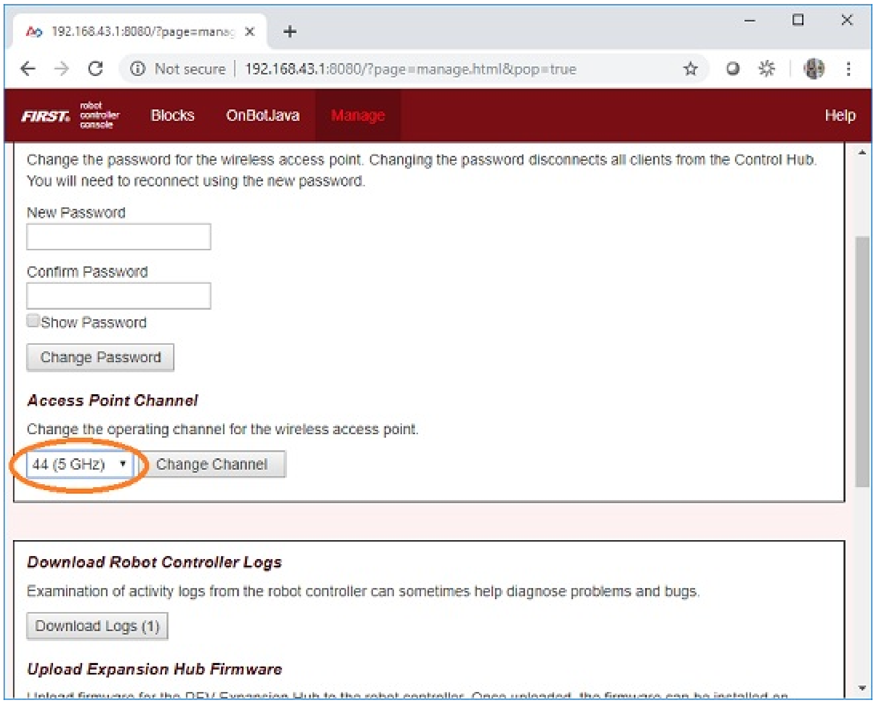
The Control Hub acts as a wireless access point for the Driver Station and for the programming laptop or Chromebook. By default the Control Hub automatically picks an operating WiFi channel. However, it is sometimes necessary to specify the operating channel for the Hub.

For example, at a large competition an FTA might ask that you switch to a designated channel to avoid wireless interference that is present in the venue. Similarly, an FTA might ask you to switch to a specific channel because the FTA is monitoring that designated channel for interference or other wireless disruptions.

You can select the operating channel for the Control Hub from the *Manage* page.

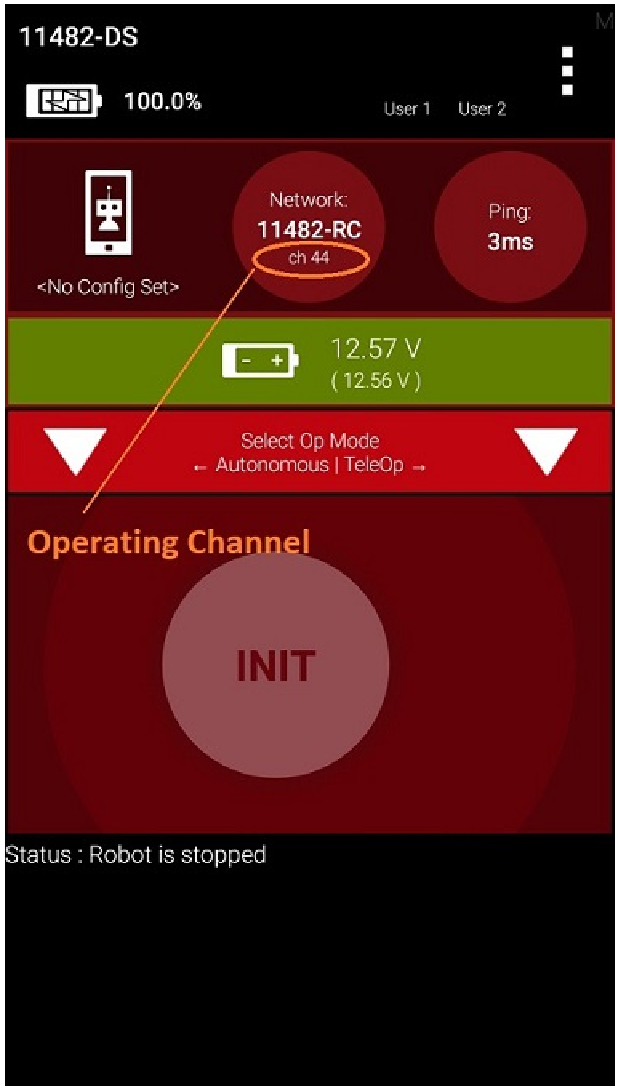
Changing the WiFi Channel Instructions

1. On the *Manage* page of the Control Hub user interface, use the drop down selector to select the desired operating channel. Note that the Control Hub supports both the 2.4 GHz and 5 GHz bands.



2. Press the *Change Channel* button to change to the new channel. Note that when the channel change occurs, the Driver Station might momentarily disconnect from the Control Hub. It should eventually, however, reconnect to Control Hub's wireless network.

3. Verify on the Driver Station that the Control Hub is operating on the desired WiFi channel. The operating channel should be displayed under the network name in the *Network* section of the main Driver Station screen.

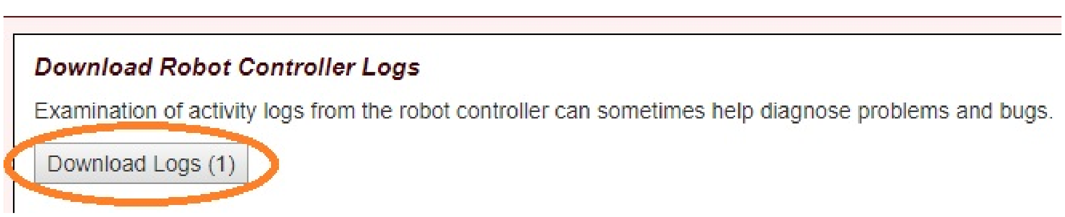


Downloading the Log File

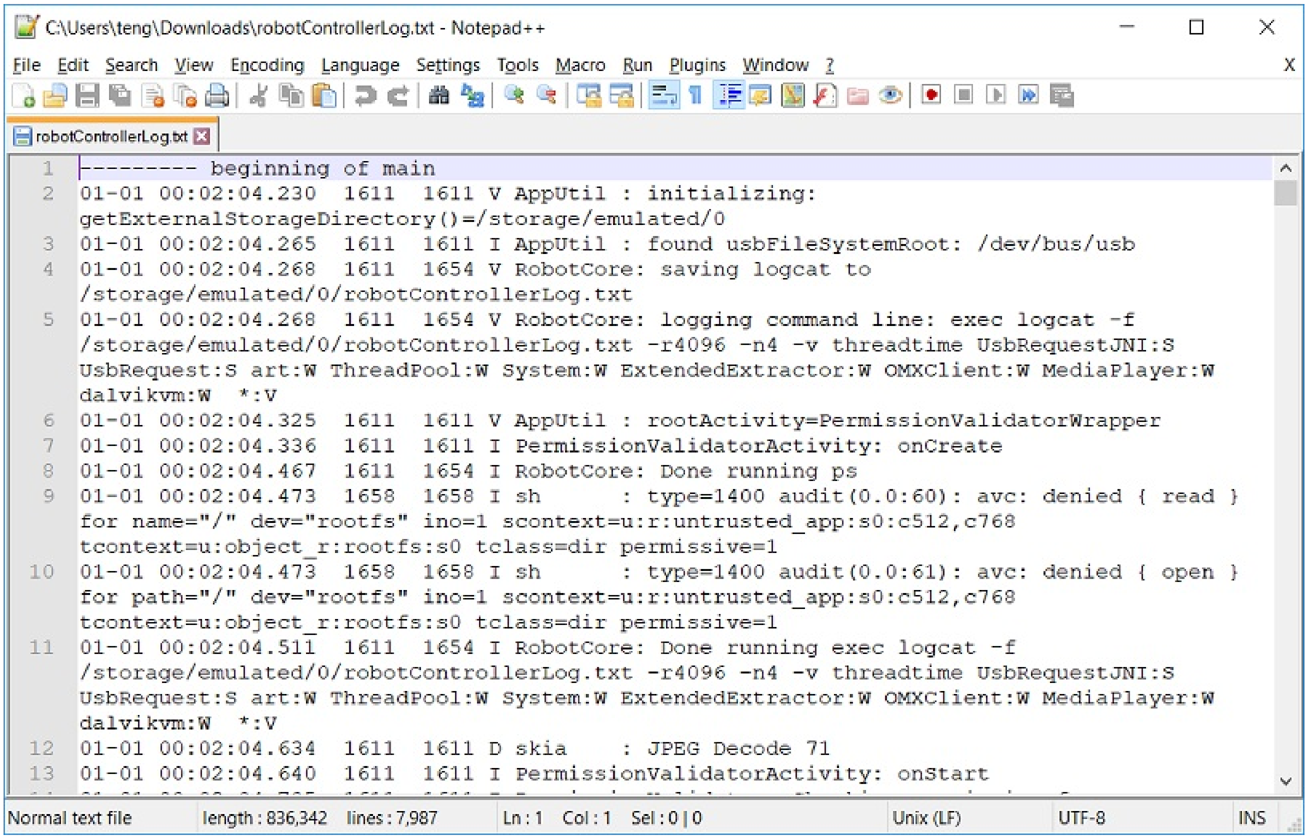
It's often helpful when troubleshooting problems with the Control System to download the log file from the Control Hub. This can be done from the *Manage* page. Note that the log file name is *robotControllerLog.txt* by default.

Downloading the Log File Instructions

1. On the *Manage* page of the Control Hub user interface, press the *Download Logs* button to download the Robot Controller log file.



2. Verify that the Robot Controller log file was downloaded to the Downloads directory of your computer.
3. Use a text editor such as Notepad++ or Microsoft's WordPad to open and view the contents of the log file. Note that the Windows app, Notepad, will not properly display the contents of the log file.



Updating the Expansion Hub Firmware

The Control Hub has its own built-in REV Robotics Expansion Hub. The purpose of the Expansion Hub board is to facilitate communication between the Control Hub's Android controller and the motors, servos, and sensors of the robot. Periodically, REV Robotics will release new versions of the firmware which contains fixes and improvements for the Expansion Hub. The firmware releases are in the form of a binary (.bin) file.

The REV Hardware Client software can update the firmware for the Control Hub's embedded Expansion Hub.

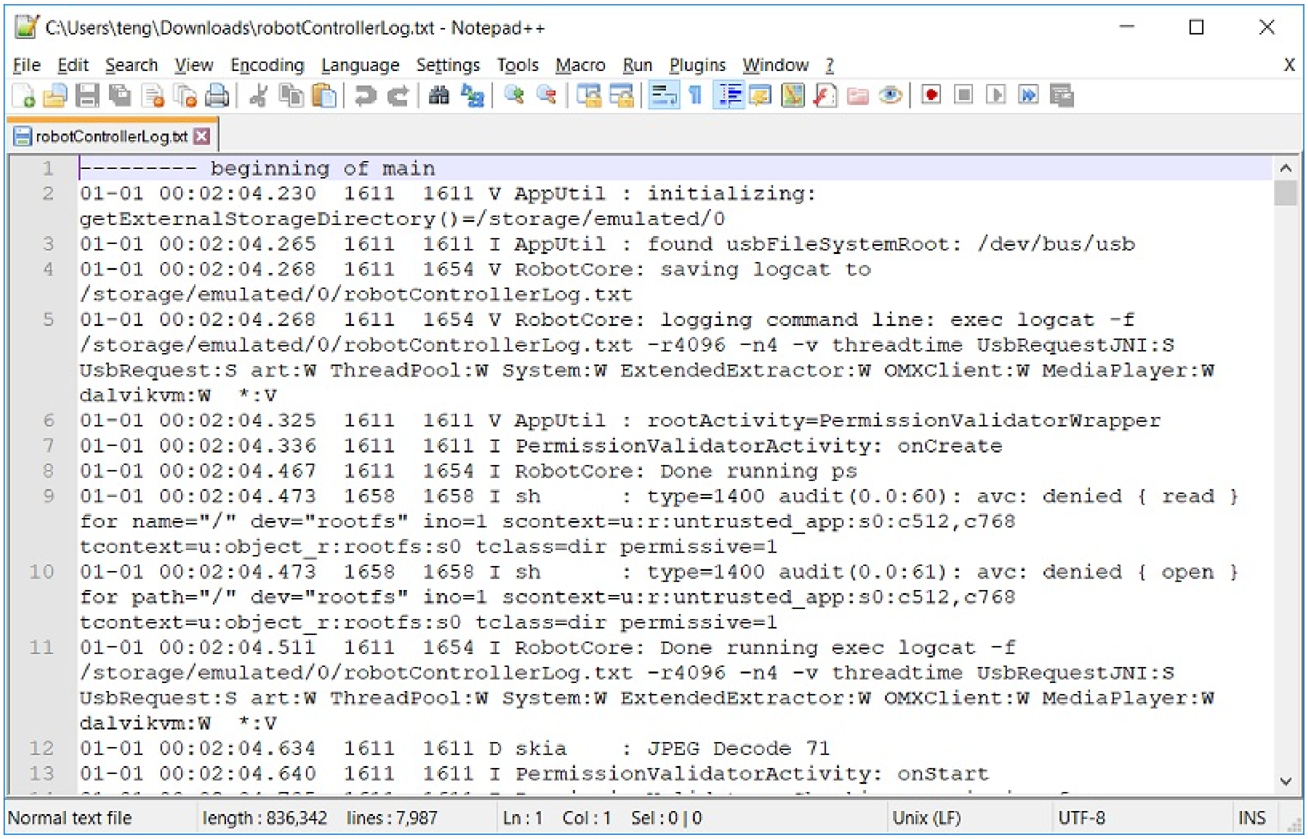
As an alternate, you can use the *Manage* interface from a connected laptop or Driver Station (DS) app to upload a Control Hub's firmware, or to update it using the included or uploaded version. New firmware images can be obtained from the REV Robotics website.

Also, included or uploaded Control Hub firmware can be updated in Robot Controller Advanced Settings, from a paired Driver Station (DS) app as shown below.

These three methods do not apply to updating the firmware of an Expansion Hub connected to a Control Hub via RS485 data wire. Standalone Expansion Hubs must be updated by direct USB plug-in to a laptop running the REV Hardware Client or to a Robot Controller phone.

Uploading and Updating the Expansion Hub Firmware

1. On the *Manage* page of the Control Hub user interface, press the *Select Firmware* button to to select the firmware file that you would like to upload.

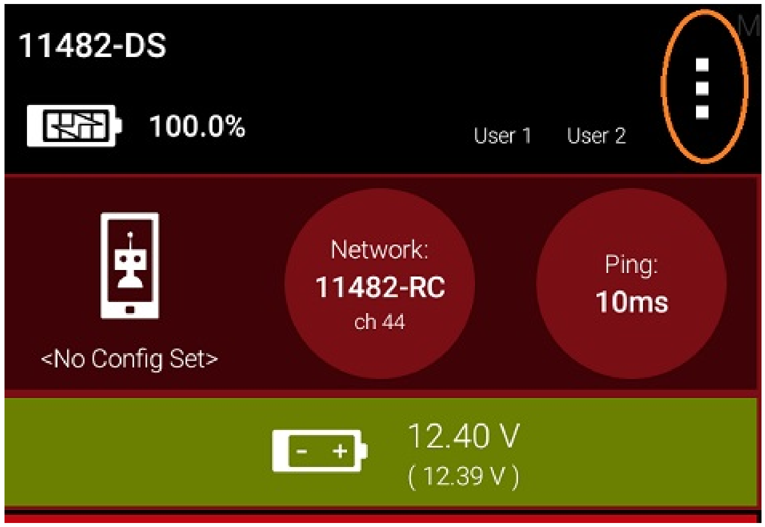


An *Upload* button should appear after you successfully selected a file.

2. Press the *Upload* button to upload the firmware file from your computer to the Control Hub.

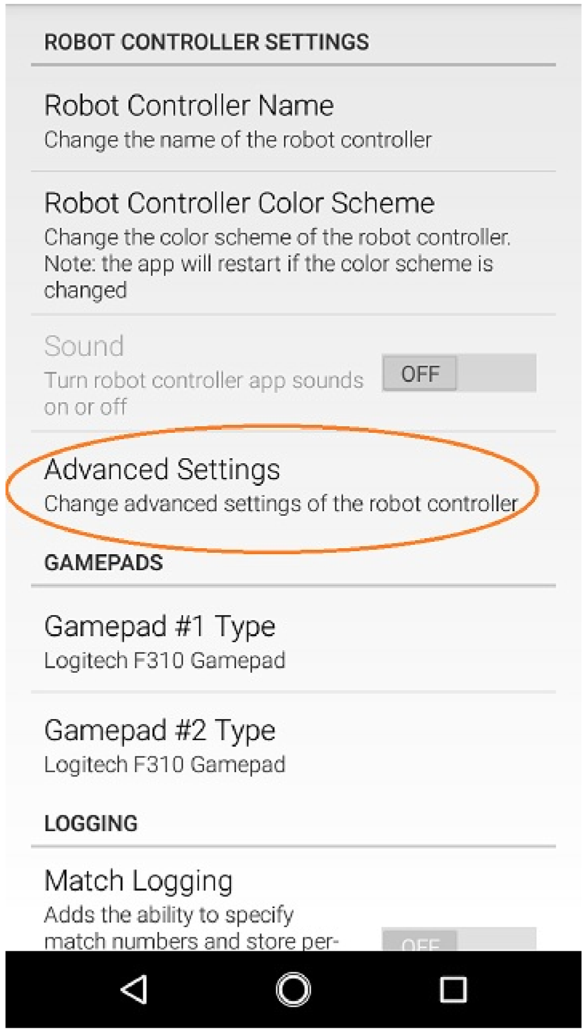
The words "Firmware upload complete" should appear once the file has been uploaded successfully.

3. On the Driver Station, touch the three dots in the upper right hand corner to display a pop-up menu.

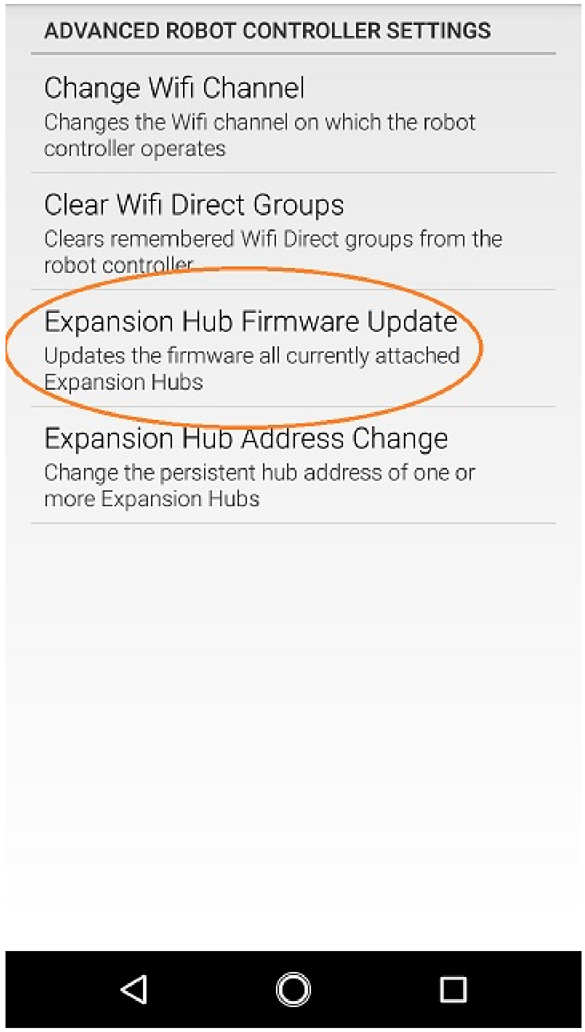


4. Select *Settings* from the pop-up menu to display the Settings activity.

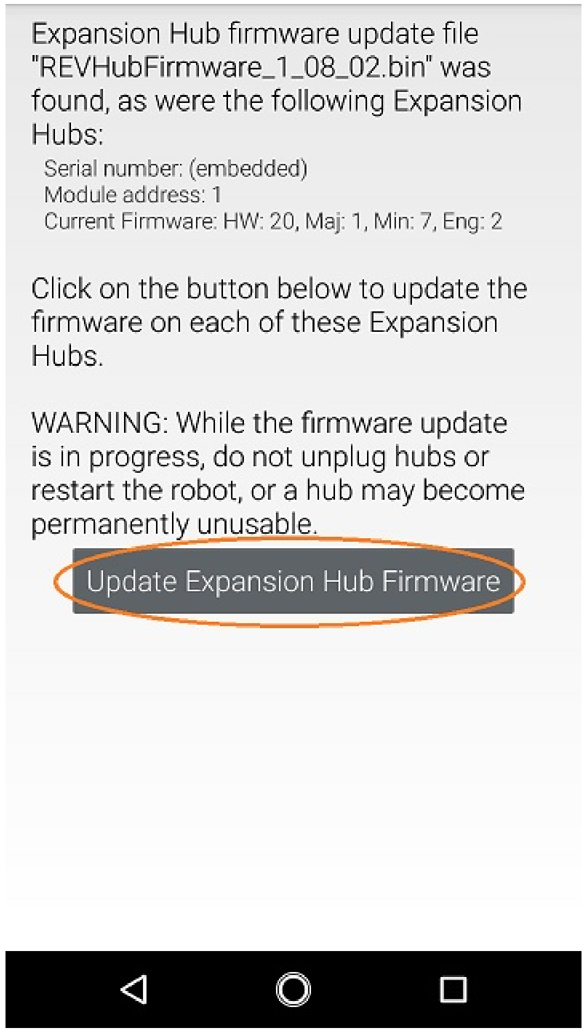
5. On the Driver Station, scroll down and select the *Advanced Settings* item (under the *ROBOT CONTROLLER SETTINGS* category).



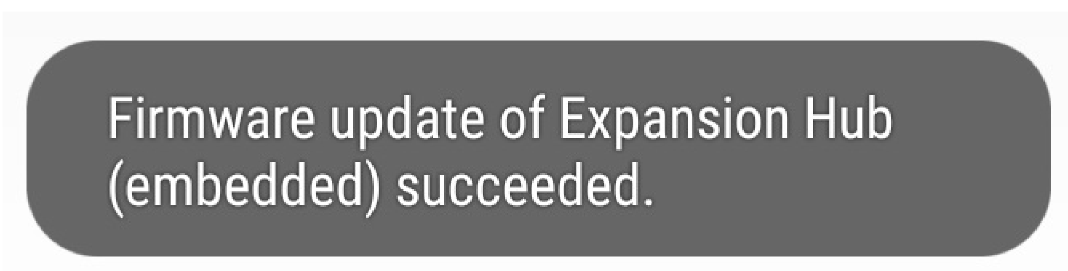
6. Select the *Expansion Hub Firmware Update* item on the *ADVANCED ROBOT CONTROLLER SETTINGS* activity.



7. If a firmware file that is different from the version currently installed on the Expansion Hub was successfully uploaded, the Driver Station should display some information about the current firmware version and the new firmware version. Press the *Update Expansion Hub Firmware* button to start the update process.



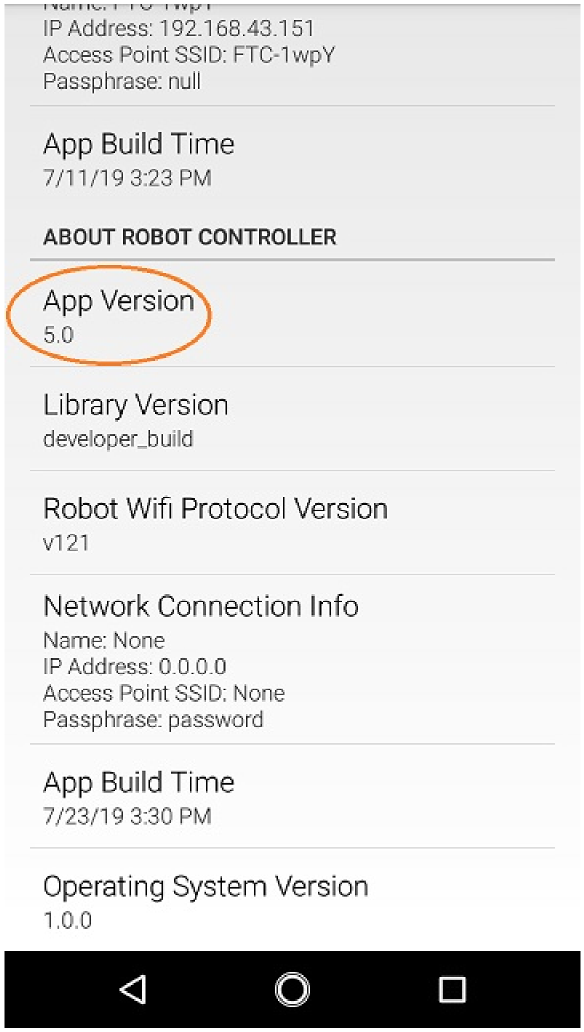
8. A progress bar will display while the firmware is being updated. Do not power off the Control Hub/Expansion Hub during this process. The Driver Station will display a message when the update process is complete.



Updating the Robot Controller App

It is important to know how to update the Robot Controller app that is installed on a Control Hub. FIRST periodically releases new versions of this app, which contain improvements and fixes, as well as season-specific data and features.

Note that you can see the Robot Controller app version number through the Driver Station user interface. Select the *About* menu option on the Driver Station and note the App Version number under the *ABOUT ROBOT CONTROLLER* section.



The REV Hardware Client software can update the Robot Controller (RC) app on the Control Hub.

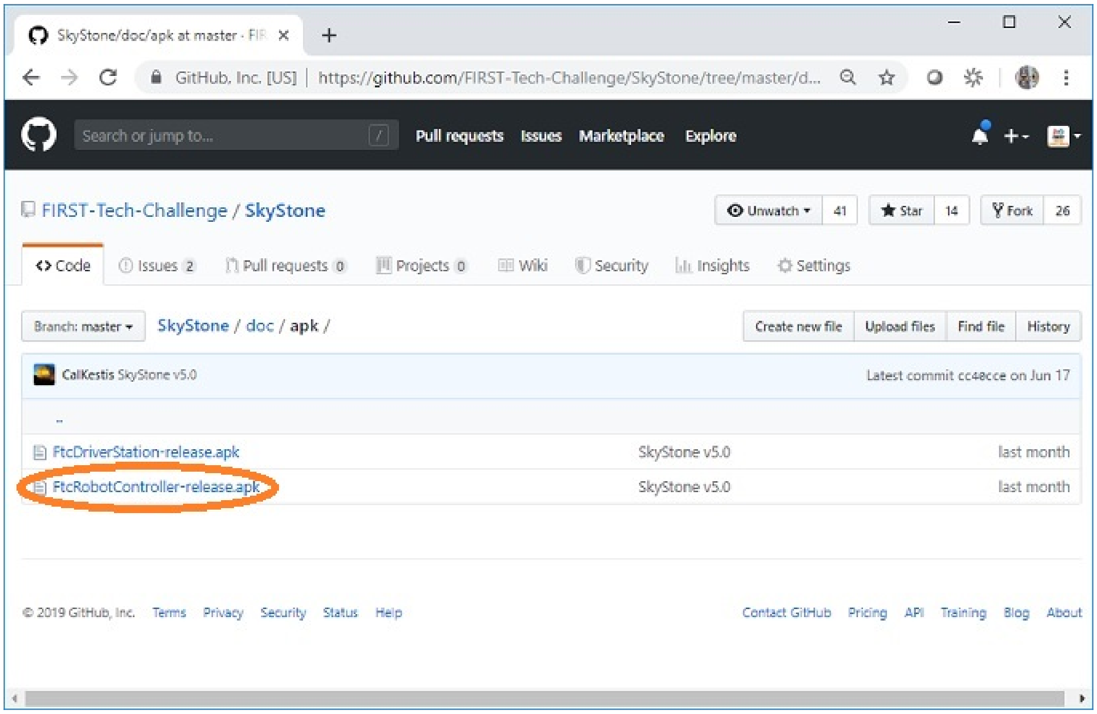
As an alternate, Control Hub users can download the RC app from the FIRST Tech Challenge Github repository and use the *Manage* page to complete the update.

Note that if you are an Android Studio user, then by updating to the newest version of the Android Studio project folder you will update the Robot Controller app when you build the project and install it on your Control Hub.

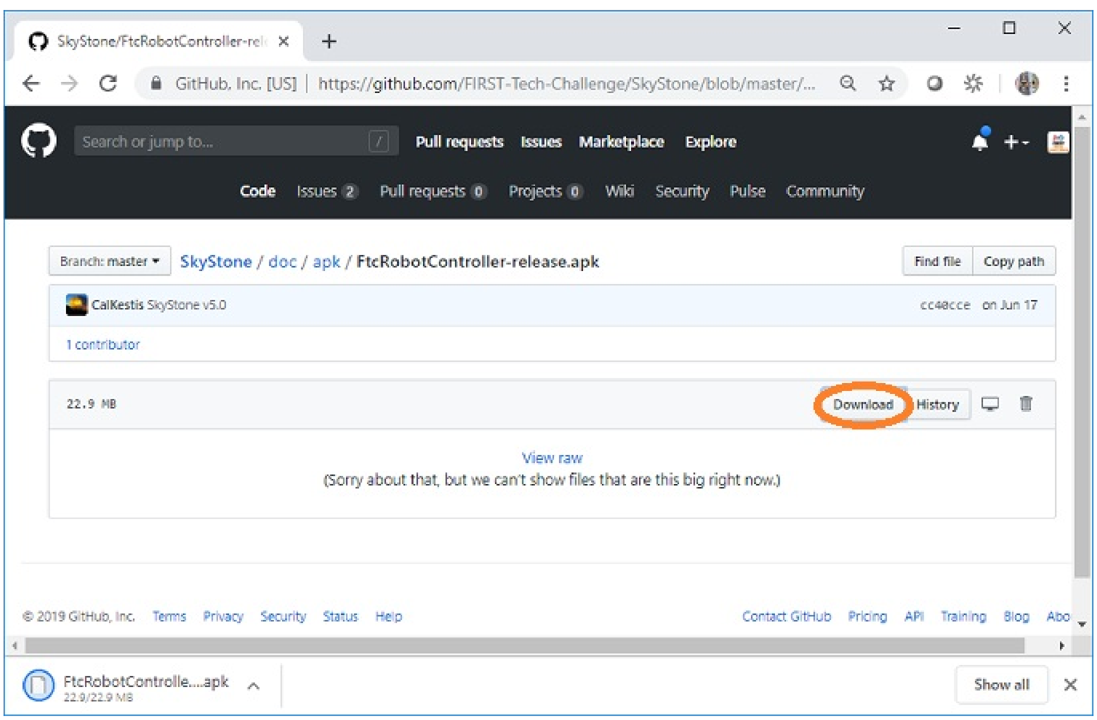
Tip: If you update your Robot Controller, then you should also update your Driver Station software to the same version number.

Updating the Robot Controller App Instructions

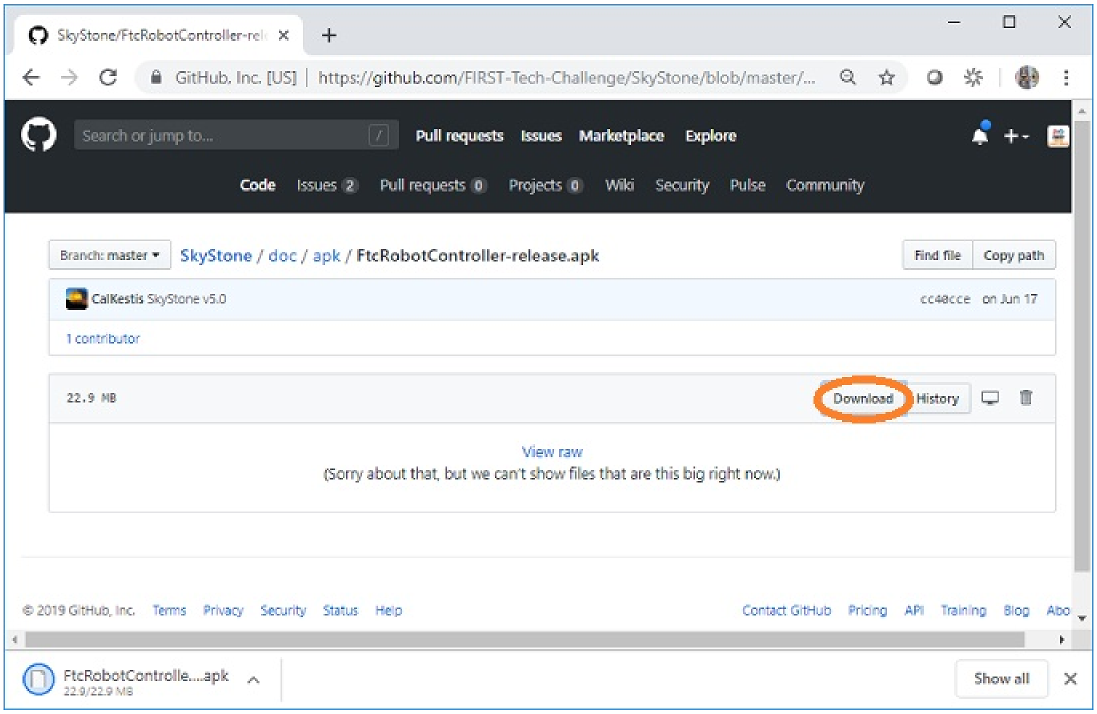
1. Go to the GitHub repository.
2. Locate the *FtcRobotController-release.apk* file.



3. Click on the filename (or *Download* button) to download the Robot Controller app as an APK file to your computer.

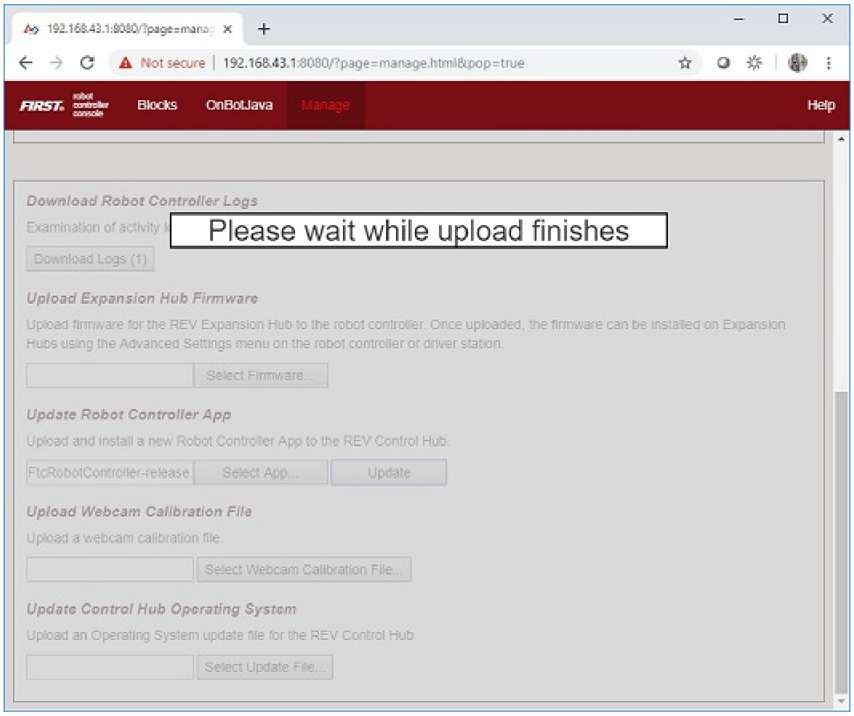


4. On the *Manage* page, click on the *Select App* button to select the Robot Controller app that you would like to upload to the Control Hub.



An *Update* button should appear if an APK file was successfully selected.

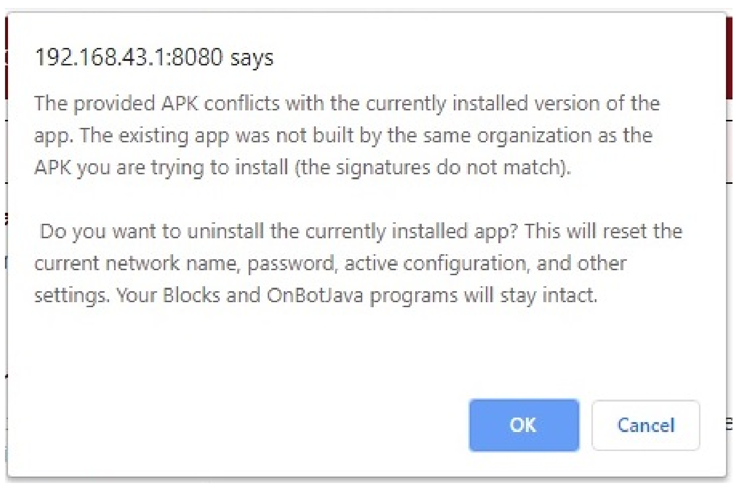
5. Click on the *Update* button to begin the update process.



6. During the update process, if the Control Hub detects that the digital signature of the APK that is being installed is different from the digital signature of the APK that is already installed, the Hub might prompt you to ask if it is OK to uninstall the current app and replace it with the new one.

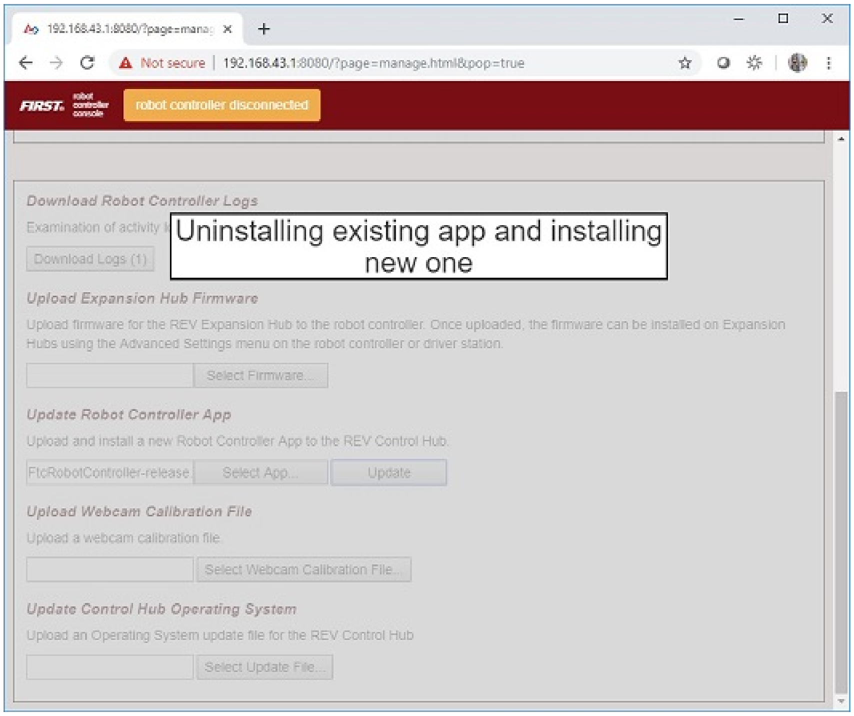
   This difference in digital signatures can occur, for example, if the previous version of the app was built and installed using Android Studio, but the newer app was downloaded from the GitHub repository.

   Press *OK* to uninstall the old app and continue with the update process.

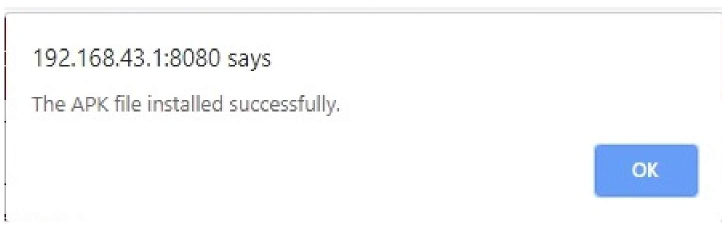


7. If the update process had to uninstall the previous version of the Robot Controller app, the network name and password for the Control Hub will be reset back to their factory values. If this happens, then you will need to reconnect your

computer to the Control Hub using the factory default values.



8. When the update process is complete and you have successfully reconnected your computer to the Control Hub's network, you should see an *installed successfully* message on the *Manage* web page.



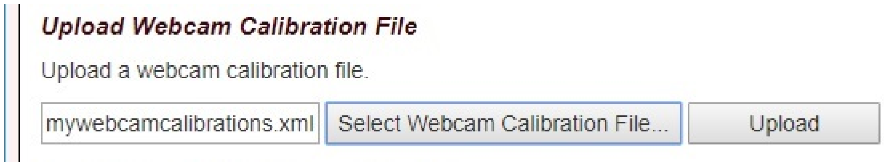
Uploading a Custom Webcam Calibration File

The Robot Controller app has built-in calibration information for a variety of commonly available webcams. Users can also create their own custom calibration files and then upload these files to a Control Hub.

A commented example of what the contents of a calibration file should look like can be found in a file called *teamwebcam-calibrations.xml*, which is included with the Android Studio project folder. This example calibration file can be found here.

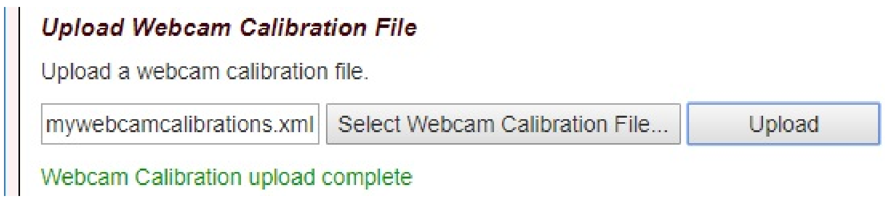
Uploading a Custom Webcam Calibration File Instructions

1. On the *Manage* page, click on the *Select Webcam Calibration File* button to select the calibration file.



   An *Upload* button should appear if a file was successfully selected.

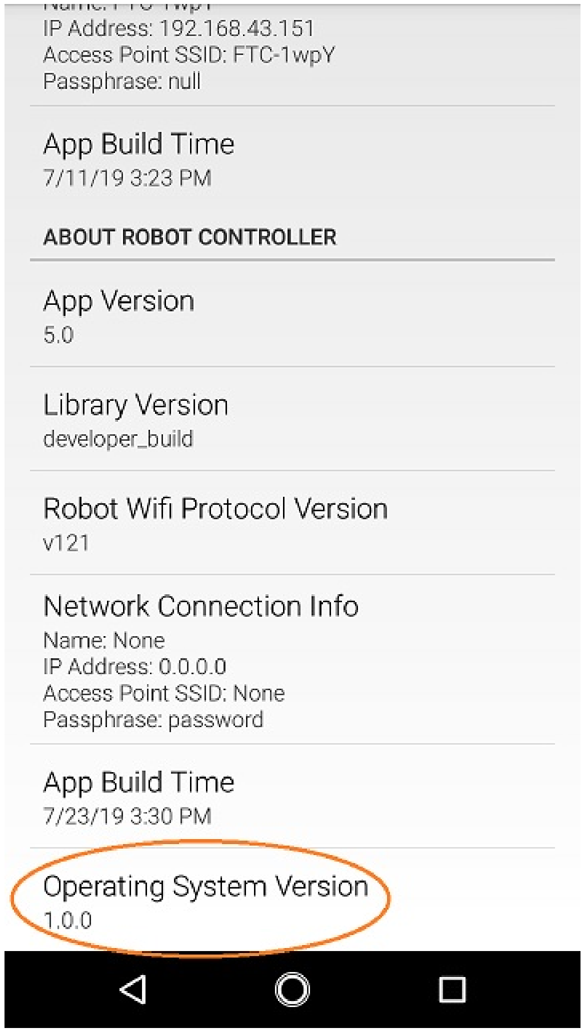
2. Click on the *Upload* button to upload the selected file. If the upload was successful, then the *Manage* page will display a message indicating that the upload has completed.



Updating the Control Hub OS

REV Robotics periodically releases new versions of the Control Hub operating system (OS). These new versions incorporate fixes, improvements, and new features.

Note that you can see the Control Hub OS version number through the Driver Station user interface. Select the *About* menu option on the Driver Station and note the Operating System Version number under the *ABOUT ROBOT CONTROLLER* section.

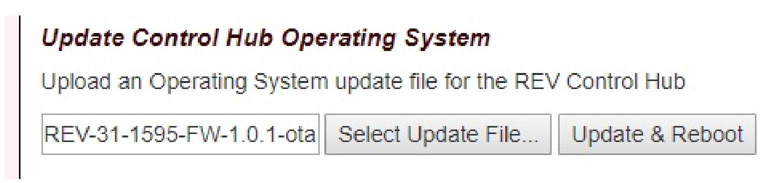


The REV Hardware Client software can update the Control Hub operating system.

As an alternate, Control Hub users can download a new Control Hub OS file from the REV Robotics website and use the *Manage* page to complete the update of the OS.

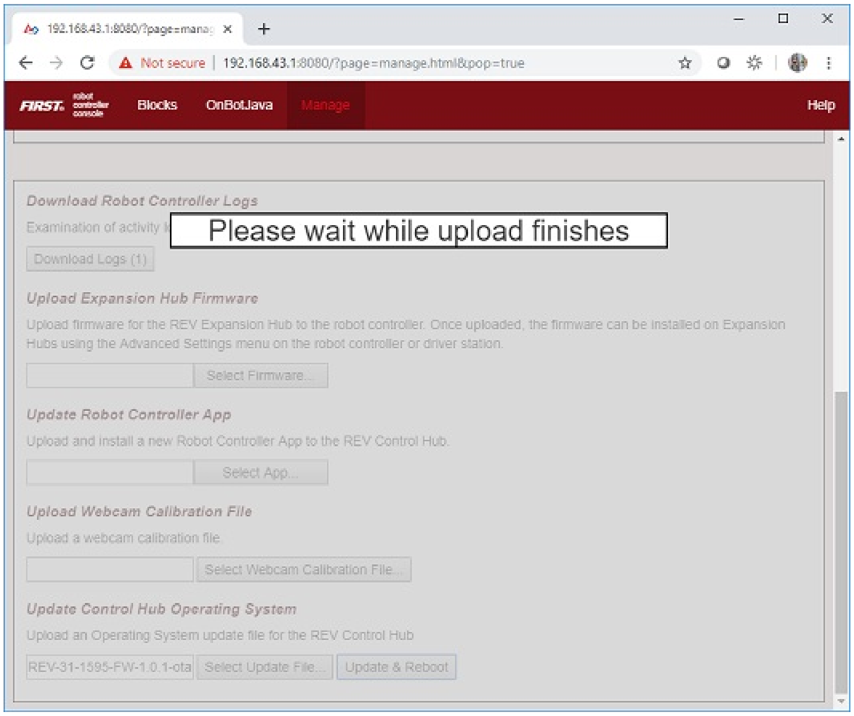
Updating the Control Hub OS Instructions

1. Download the new Control Hub OS update file from the REV Robotics website.
2. On the *Manage* page, click on the *Select Update File* button to select the OS update file that you would like to upload.

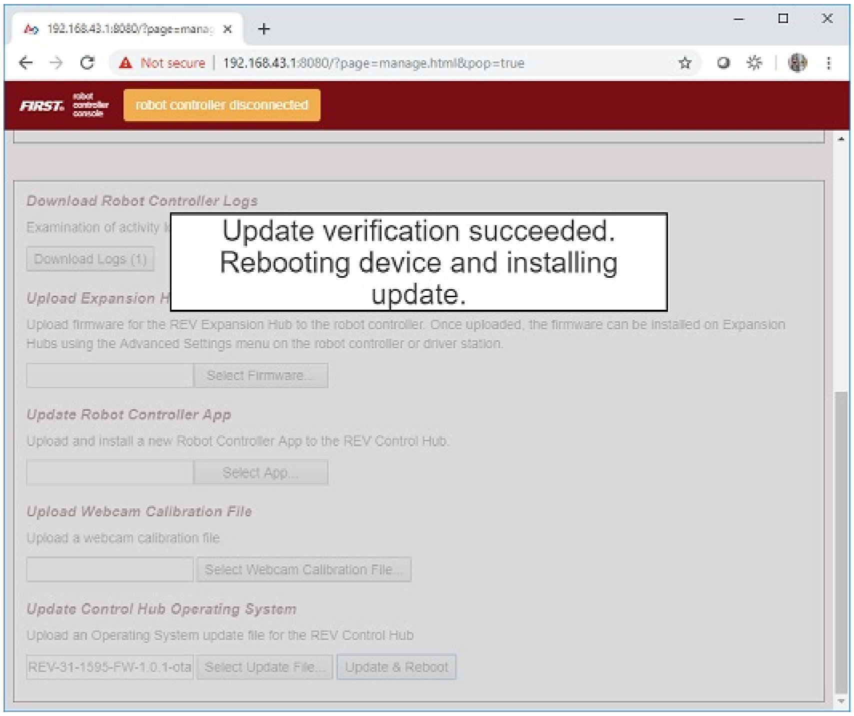


An *Update & Reboot* button should appear if an update file was successfully selected.

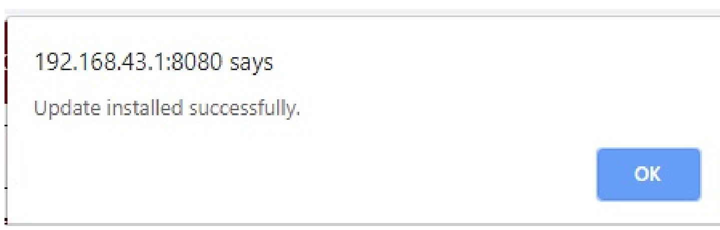
3. Click on the *Update & Reboot* button to start the update process. Please wait while the OS file gets uploaded to the Control Hub. Note that since the file is relatively large, it might take several minutes before the upload is complete. Do not turn off the Control Hub while the process is underway.



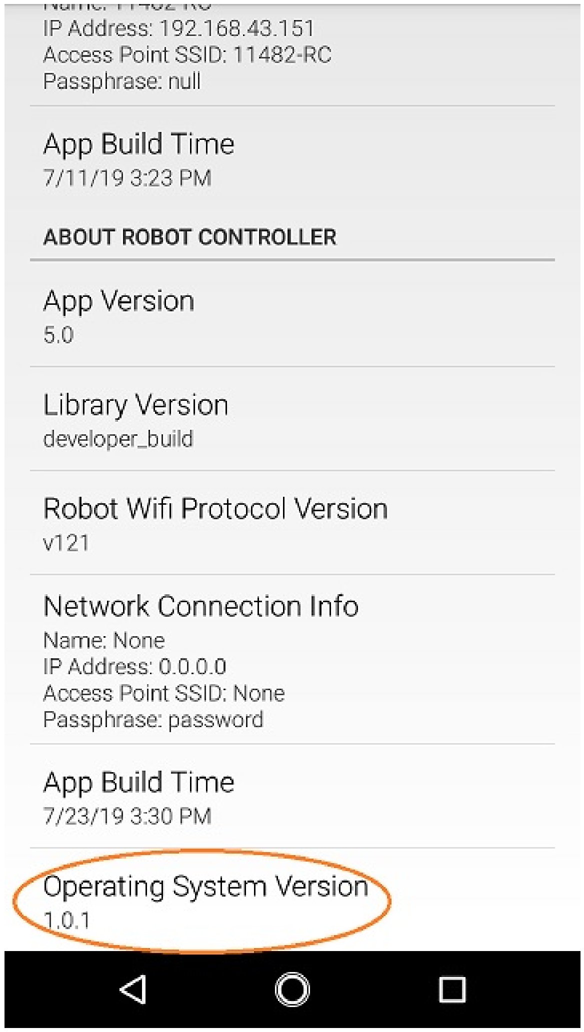
4. If the upload was successful, the *Manage* page will display a message indicating that the device is being rebooted and the update is being installed.



5. When the OS update has completed, the Control Hub LED should switch from blue, back to its normal blink pattern (green, then it will blink blue once to indicate the Hub's serial address number, then the pattern repeats itself). Reconnect your computer to the Control Hub network and verify that the update was a success.



Note that you can also check in the About page (through the Driver Station app) to verify the updated version number of the Control Hub OS.



Connecting to the Control Hub Using Wireless ADB

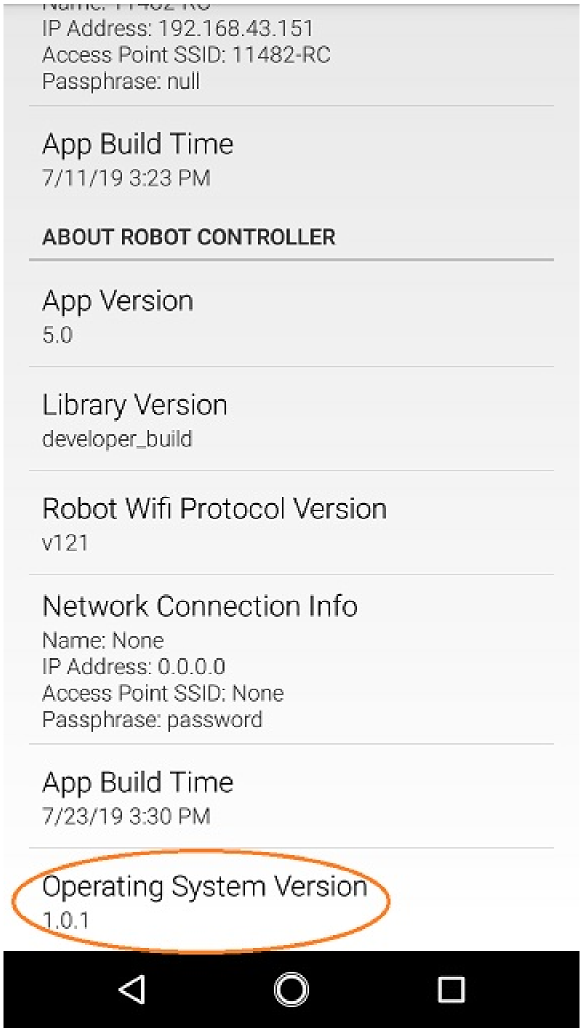
Advanced users who use Android Studio to build and install the Robot Controller app onto their Control Hub should be familiar with the Android Debug Bridge (adb) utility. adb is included with the Android development platform tools. It can be used to communicate with an Android device such as the Control Hub.

Traditionally, programmers use a hard-wired USB connection to communicate using adb to their Android device. adb also supports a mode where commands are sent back and forth through a wireless connection.

The Control Hub is configured so that it automatically will support an adb wireless connection request on port 5555.

Connecting to the Control Hub Using Wireless ADB Instructions

1. Verify that your laptop is connected to the Program & Manage wireless network of the Control Hub. If you are connected to the network, you should be able to see the *Robot Controller Connection Info* page when you navigate to address "192.168.43.1:8080":



If your laptop is not connected and you are unable to access the *Robot Controller Connection Info* page, then read the instructions in the following tutorial (Connecting-a-Laptop-to-the-Program-&-Manage-Network) to learn how to connect to the Program & Manage network.

Connecting a laptop to the Program & Manage Network

2. Verify that the PATH environment variable for your Windows computer includes the path to the adb.exe executable file. The Android Developer website tells you where in your Android SDK installation you can find the adb.exe file. This post from HelpDeskGeek.com shows how to add a directory to your Windows PATH environment variable.

3. Open a Windows Command Prompt and type in "adb.exe connect 192.168.43.1:5555". This should connect your adb server to the Control Hub over the wireless connection.

21.2.3 Managing a Smartphone Driver Station

REV Driver Hub

The REV Driver Hub is preloaded with the Driver Station (DS) app. The procedures described below for a DS phone, also apply to a REV Driver Hub.

Changing the Name

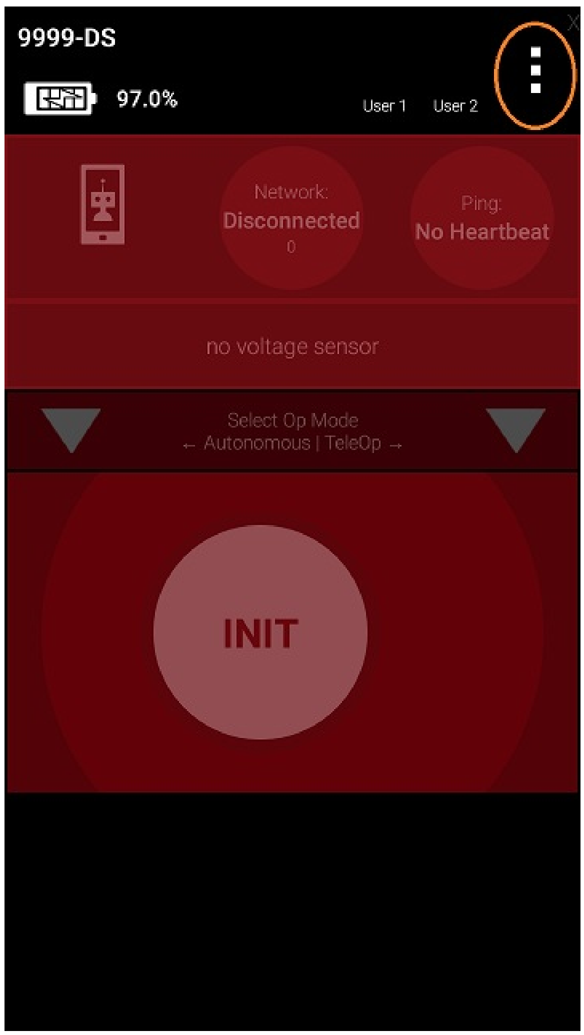
In order to comply with game manual rule <RS01>, the name of the Driver Station (DS) smartphone should be changed.

This can be done in the DS app, as described below.

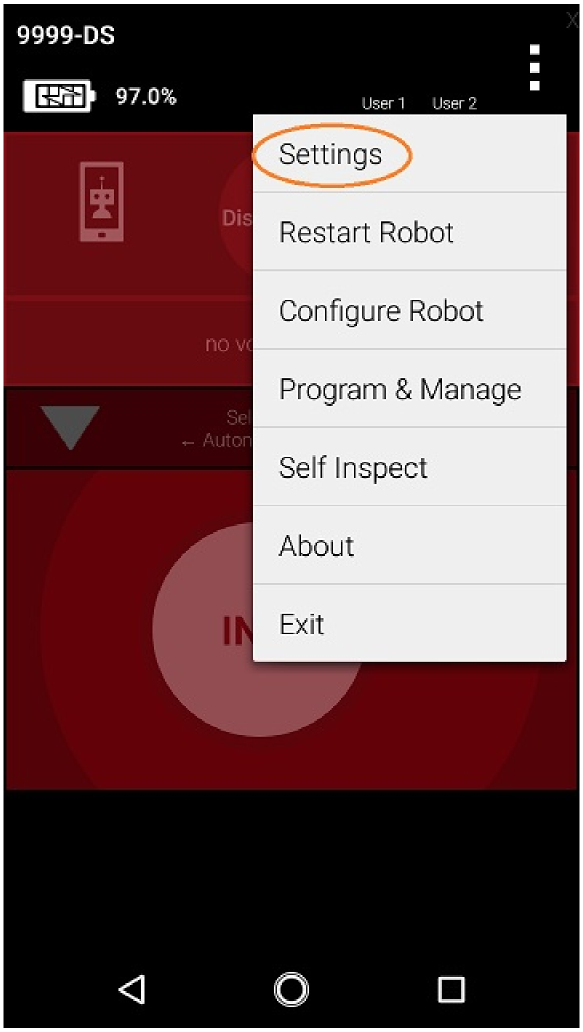
As an alternate, *found here* show how to rename a smartphone using the Android Settings activity of the phone.

Changing the Name of a Driver Station Instructions

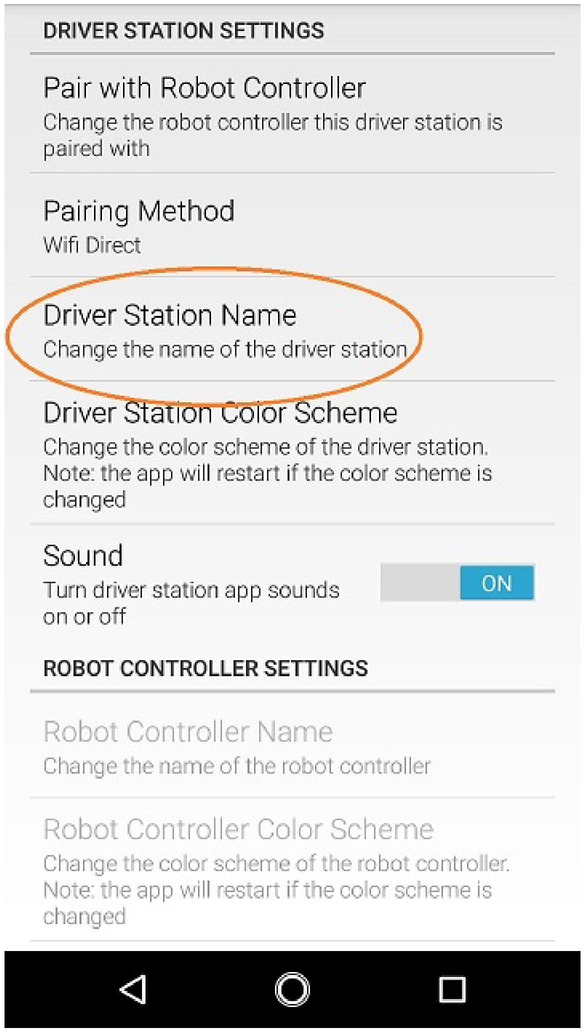
1. On the Driver Station phone, touch the three dots in the upper right hand corner to display a pop-up menu.



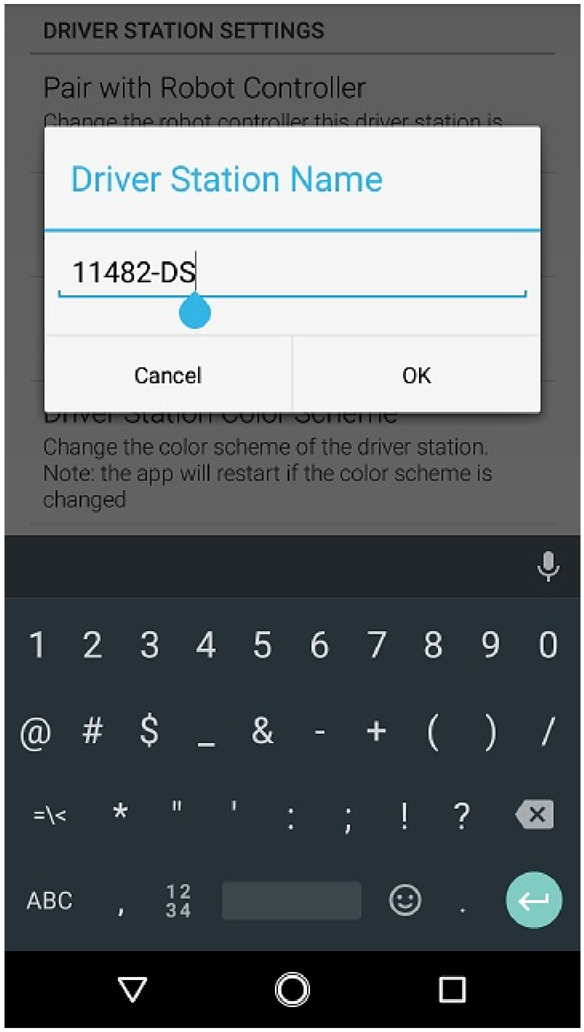
2. Select the *Settings* menu item from the pop-up menu.



3. Click on *Driver Station Name* on the *DRIVER STATION SETTINGS* page.



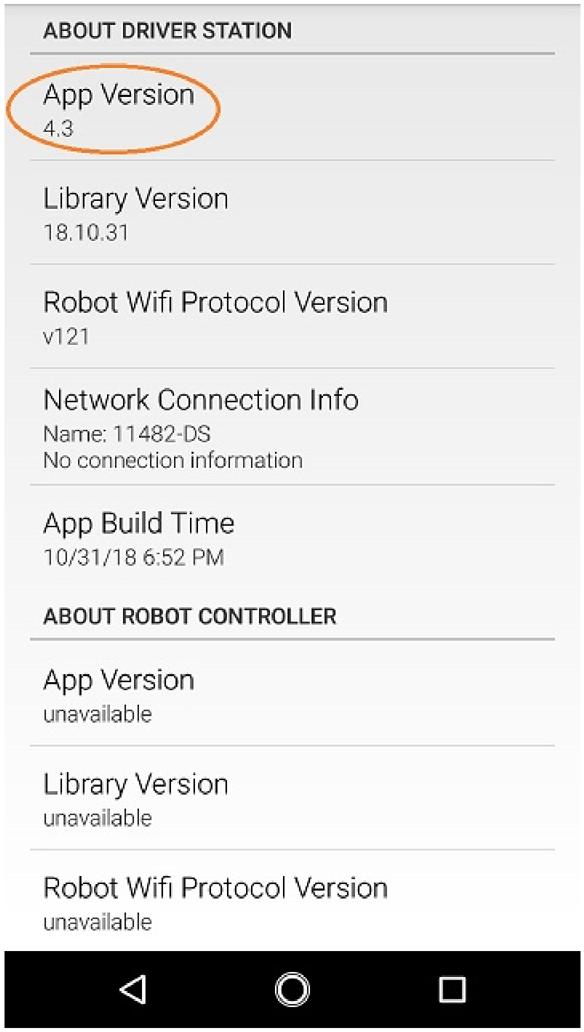
4. Specify the new Driver Station Name and press *OK* to accept the changes.



Updating the Driver Station App

It is important to know how to update the Driver Station app that is installed on your smartphone. FIRST periodically releases new versions of this app, which contain improvements and fixes, as well as season-specific data and features.

Note that you can see the Driver Station app version number through the Driver Station user interface. Select the *About* menu option on the Driver Station and note the App Version number under the *ABOUT DRIVER STATION* section.



As of 2021, all apps (v 6.1 and higher) are no longer available on Google Play.

The REV Hardware Client software will allow you to download the apps to approved devices: REV Control Hub, REV Expansion Hub, REV Driver Hub, and approved Android devices. Here are some of the benefits:

- Connect a REV Control Hub via WiFi.
- One Click update of all software on connected devices.
- Pre-download software updates without a connected device.
- Back up and restore user data from Control Hub.
- Install and switch between DS and RC applications on Android Devices.
- Access the Robot Control Console on the Control Hub.

All teams using Blocks, OnBot Java or Android Studio can use the REV Hardware Client to update the Driver Station (DS) app on a DS phone.

NOTE: it will take an estimated 7.5 minutes per device to complete this task.

As an alternate, the app releases are available on the FTCRobotController Github. Download the Driver Station APK file to a computer, transfer it to the DS phone's Downloads folder, then open that file to install the DS app. This process is called "side-loading".

Important: If you update the Driver Station (DS) app, you should also update the Robot Controller (RC) app to the same version number. That process is different for Android Studio teams; updating RC phones is described *found here*

21.2.4 Managing a Smartphone Robot Controller

Changing the Name

In order to comply with game manual rule <RS01>, the name of the Robot Controller (RC) smartphone should be changed.

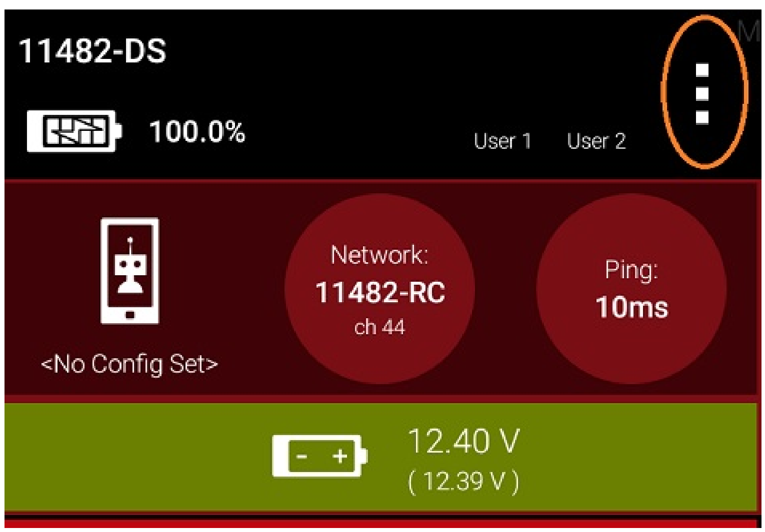
This can be done in the RC app or in a paired DS app, as described below. (These steps also work for changing the name of a Control Hub, from a paired DS app.)

As an alternate, *Renaming Devices* show how to rename a smartphone using the Android Settings activity of the phone.

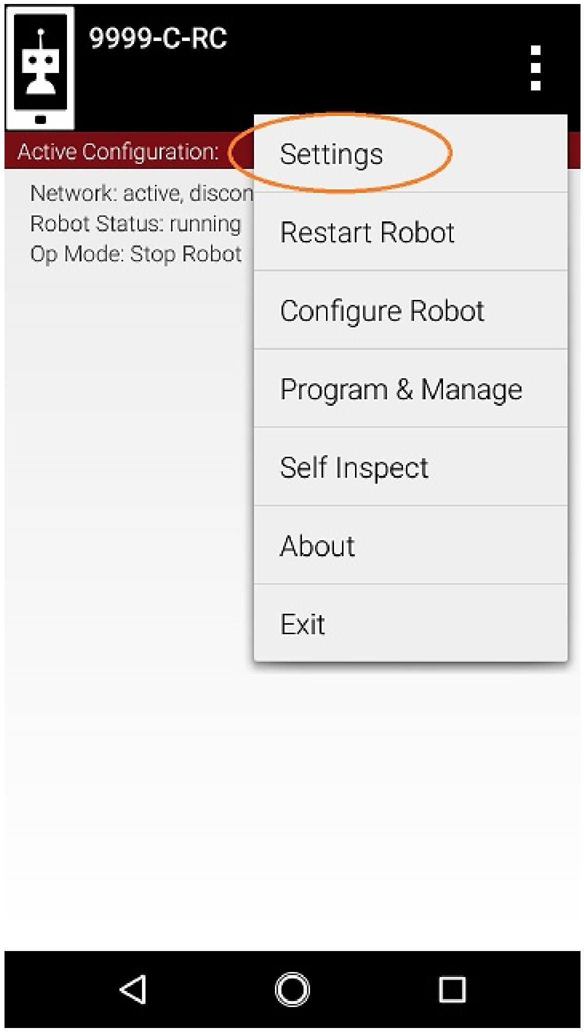
Important: Once the name of your Robot Controller is changed, you might need to reconnect your devices (Driver Station and programming laptop) to the newly changed network.

Changing the Name of a Robot Controller

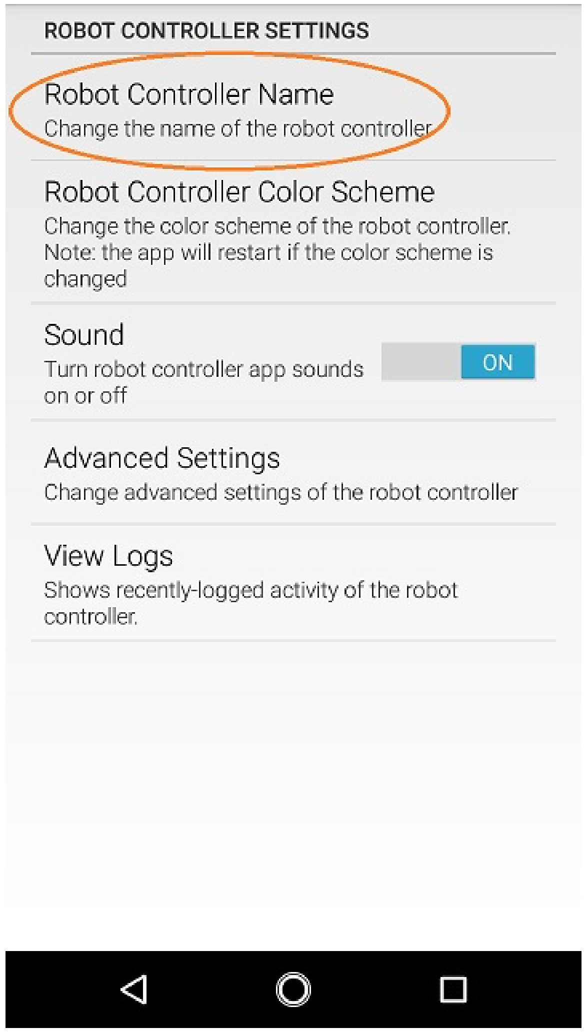
1. On the Robot Controller phone or paired Driver Station phone, touch the three dots in the upper right hand corner to display a pop-up menu.



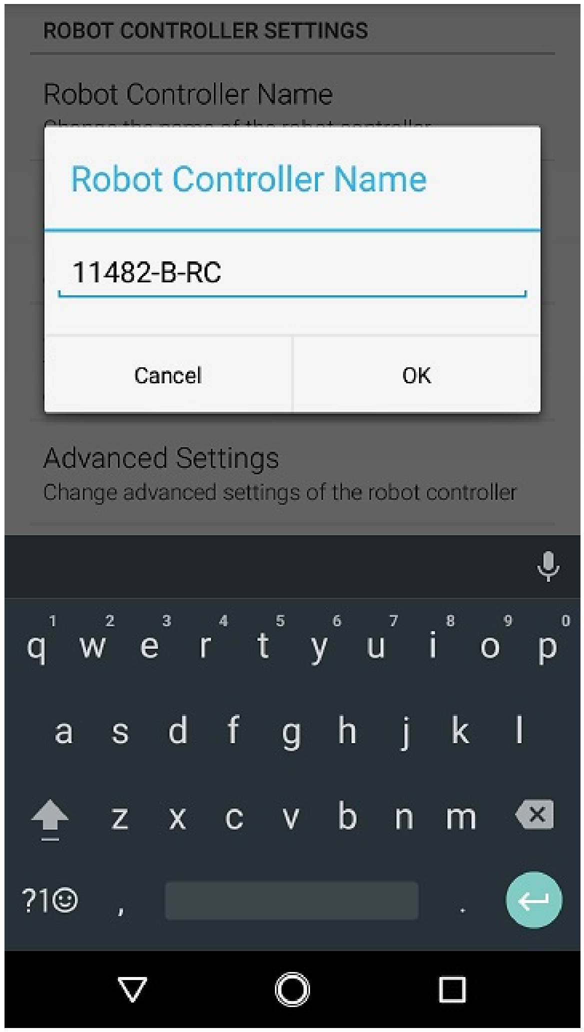
2. Select the *Settings* menu item from the pop-up menu.



3. Click on *Robot Controller Name* on the *ROBOT CONTROLLER SETTINGS* page.



4.Specify the new Robot Controller Name and press *OK* to accept the changes.



Changing the WiFi Channel

By default the smartphone Robot Controller automatically picks its own operating WiFi channel. However, it is sometimes necessary to specify the operating channel for the device.

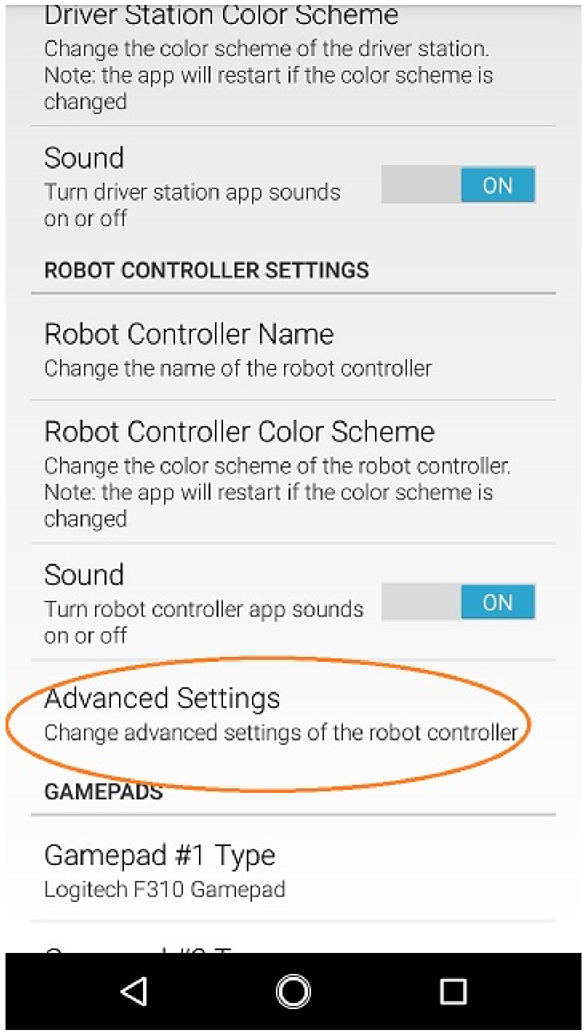
For example, at a large competition an FTA might ask that you switch to a designated channel to avoid wireless interference that is present in the venue. Similarly, an FTA might ask you to switch to a specific channel because the FTA is monitoring that designated channel for interference or other wireless disruptions.

You can change the operating channel using the Advanced Settings menu on the Robot Controller or Driver Station.

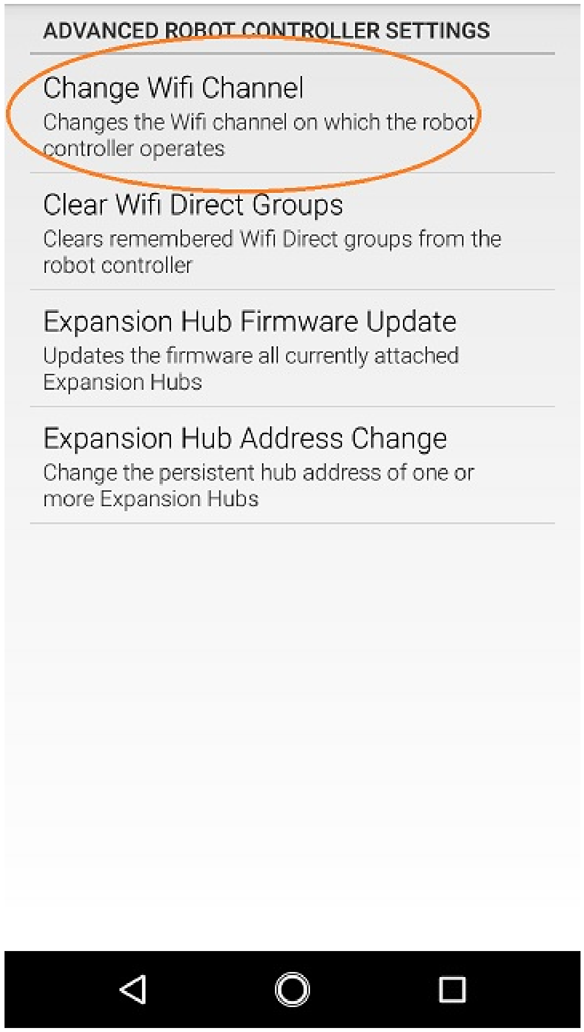
> **Warning:** Not every Android phone supports channel changing through the software. Refer to rule <RE06> in the game manual for a list of *FIRST*-approved phones that support channel changing through the software.

Changing the WiFi Channel Instructions

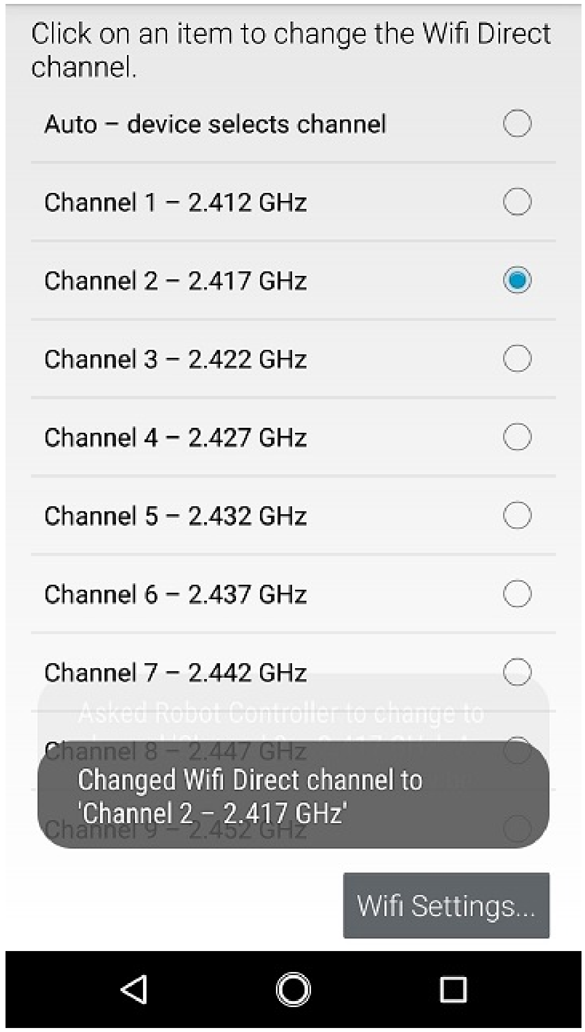
1. Verify that the Driver Station is connected to your Robot Controller.
2. Tap the three dots in the upper right hand corner of the Driver Station's main screen to display the pop-up menu and select *Settings* from the menu.
3. Scroll down to the *ROBOT CONTROLLER SETTINGS* section of the *Settings* screen and click on the words *Advanced Settings* to display the *ADVANCED ROBOT CONTROLLER SETTINGS* activity.



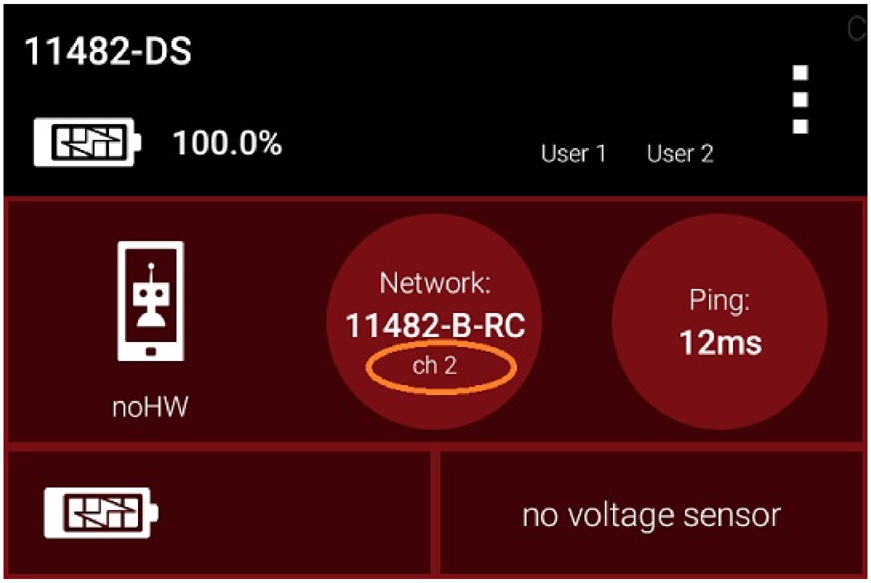
4. Click on the *Change Wifi Channel* link to display a list of available channels.



5. Select the desired operating channel. The phone should display a toast message if the channel change was successful.



6. Use the Android back arrow to return to the main Driver Station screen. The new operating channel should be displayed in the Network: section under the Robot Controller's name

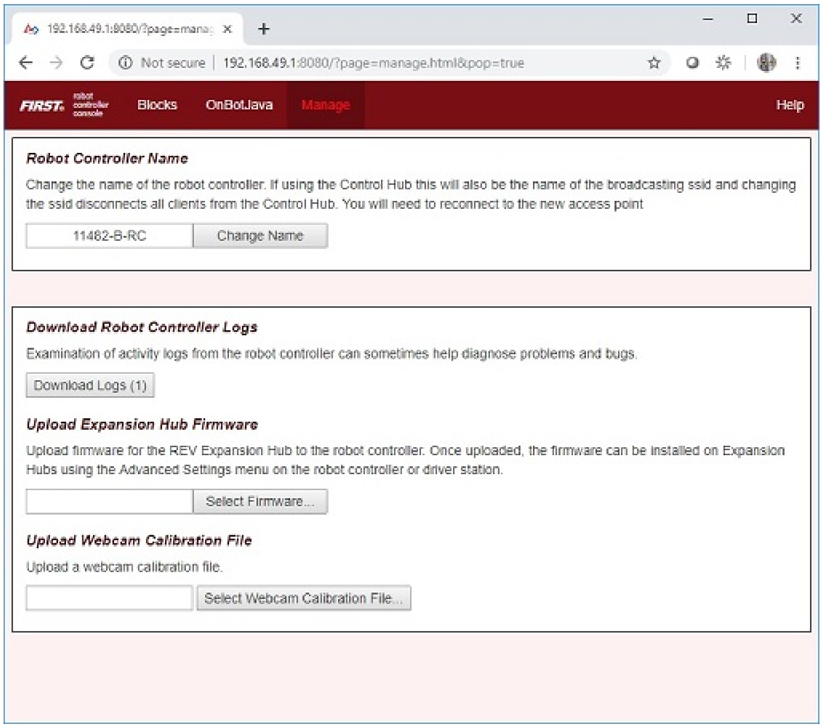


Downloading the Log File

It's often helpful when troubleshooting problems with the Control System to download the log file from the Robot Controller. This can be done from the *Manage* page. Note that the log file name is *robotControllerLog.txt* by default.

Downloading the Log File Instructions

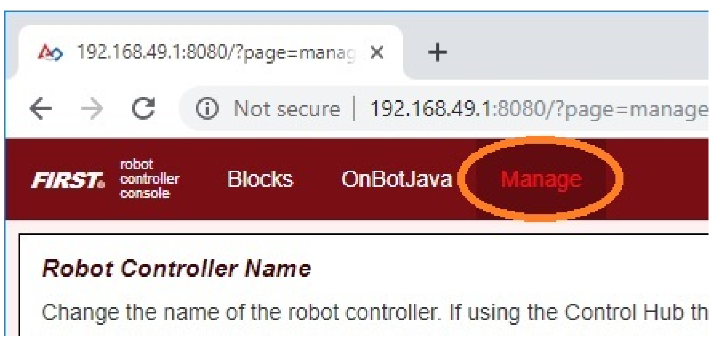
1. Verify that your laptop or Chromebook is connected to the Program & Manage wireless network of the smartphone Robot Controller. If you are connected to the network, you should be able to see the *Robot Controller Connection Info* page when you navigate to address "192.168.49.1:8080":



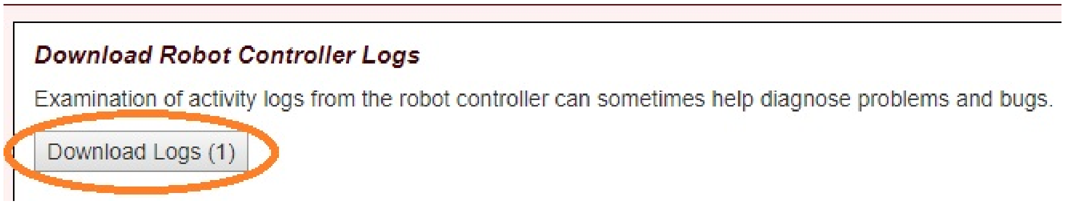
If your laptop or Chromebook is not connected and you are unable to access the *Robot Controller Connection Info* page, then read the instructions in the following tutorial to learn how to connect to the Program & Manage network.

Connecting a laptop to the Program & Manage Network

2. Click on the *Manage* link towards the top of the *Robot Controller Connection Info* page to navigate to the Manage page.

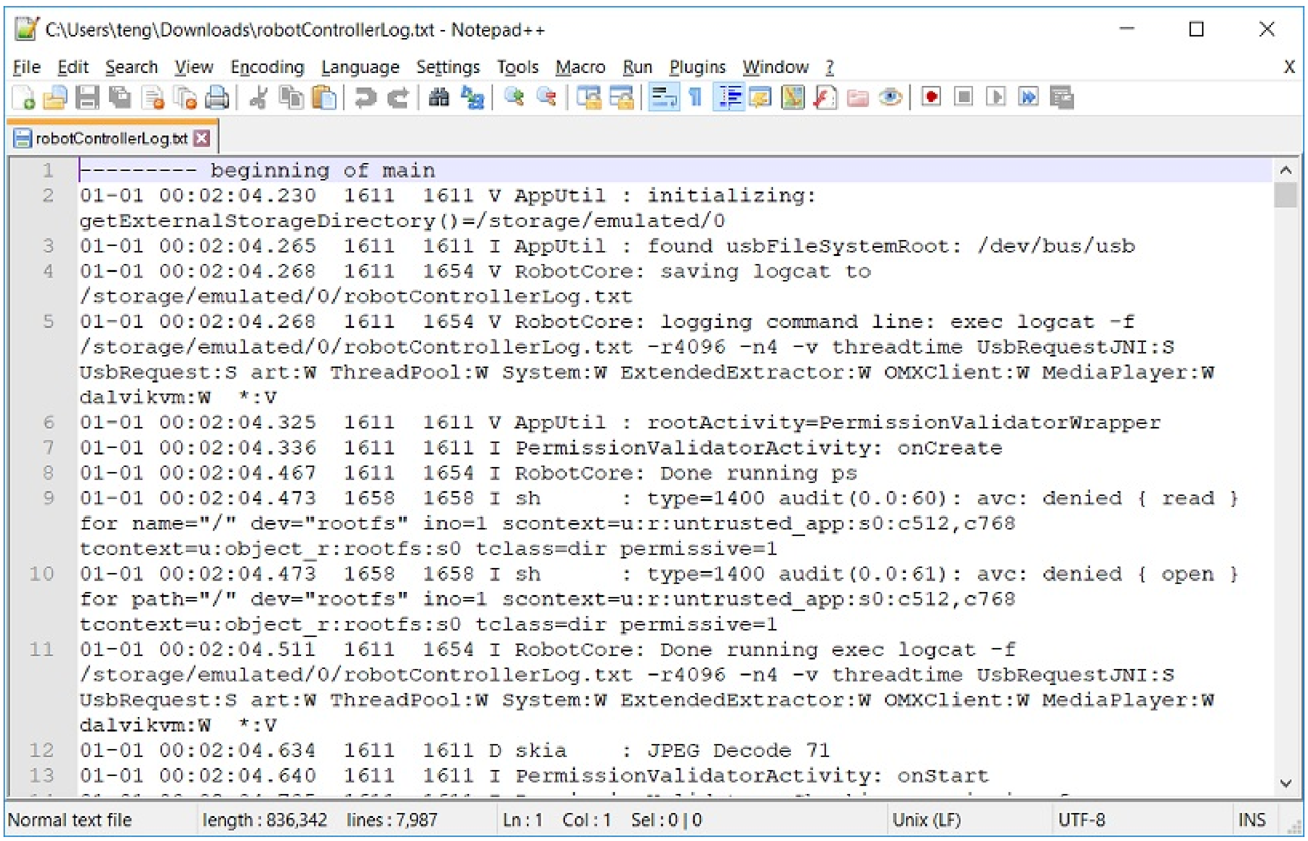


3. Click the *Download Logs* button to download the Robot Controller log file.



4. Verify that the Robot Controller log file was downloaded to the Downloads directory of your computer.

5. Use a text editor such as Notepad++ or Microsoft's WordPad to open and view the contents of the log file. Note that the Windows app, Notepad, will not properly display the contents of the log file.



Updating the Expansion Hub Firmware

A Robot Controller phone connects to a standalone REV Robotics Expansion Hub using a USB connection. The purpose of the Expansion Hub is to facilitate communication between the Robot Controller and the motors, servos, and sensors of the robot. Periodically, REV Robotics may release new versions of the firmware which contains fixes and improvements for the Expansion Hub. The firmware releases are in the form of a binary (".bin") file.

The REV Hardware Client software can update the firmware of an Expansion Hub plugged directly into the computer via USB cable.

As an alternate, you can use the *Manage* interface from a laptop or Driver Station (DS) connected to a Robot Controller phone with Expansion Hub plugged in via USB. The Manage page allows you to upload an Expansion Hub's firmware, or to update it using the included or uploaded version. New firmware images can be obtained from the REV Robotics website.

Also, included or uploaded Expansion Hub firmware can be updated in Robot Controller Advanced Settings, from a paired Driver Station (DS) app as shown below.

These three update methods do not apply to an Expansion Hub connected via RS485 data wire. Standalone Expansion Hubs must be updated by direct USB plug-in.

Updating the Expansion Hub Firmware Instructions

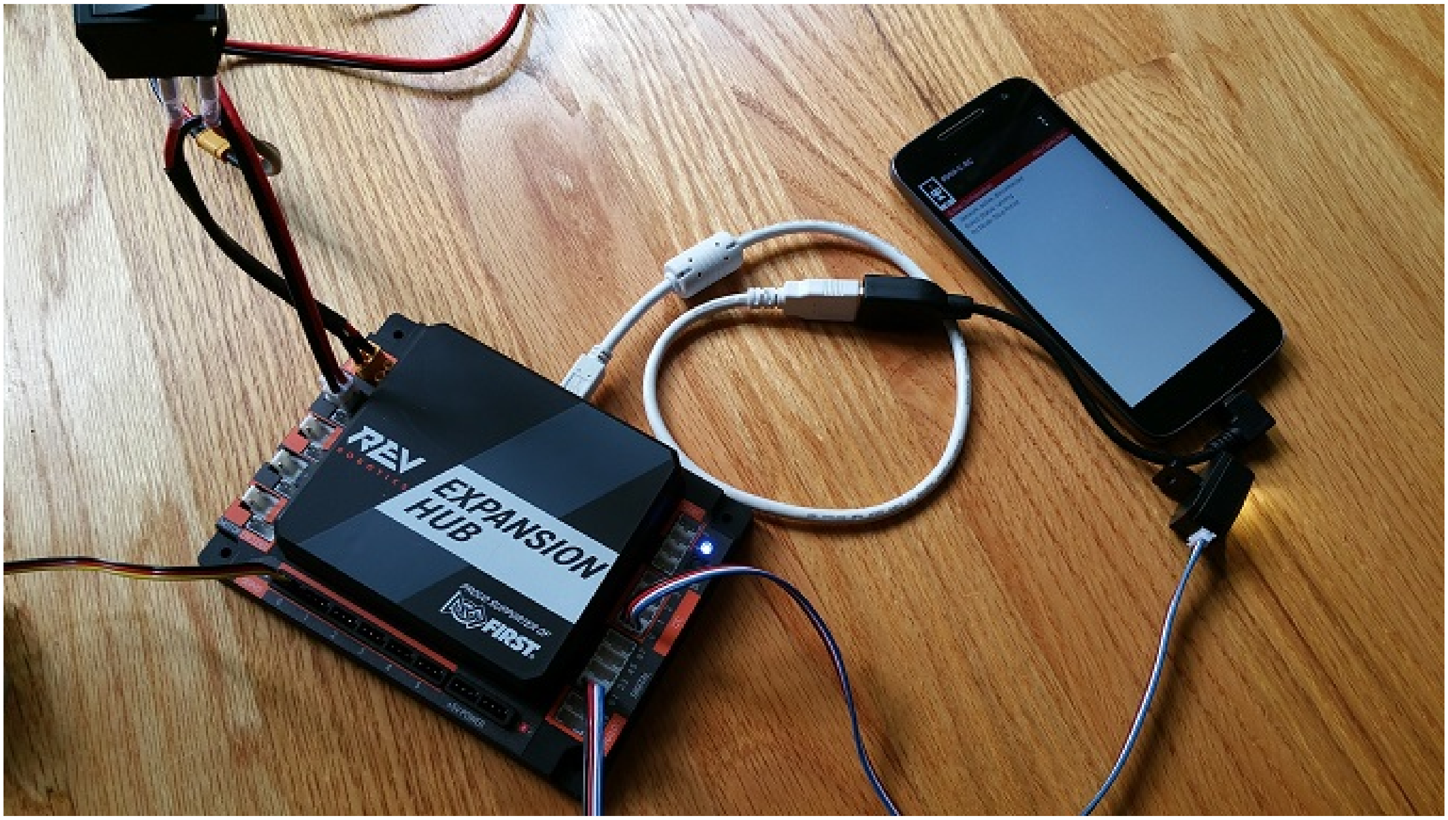
1. On the *Manage* page of the Robot Controller user interface, press the *Select Firmware* button to to select the firmware file that you would like to upload.

An _Upload_ button should appear after you successfully selected a file.

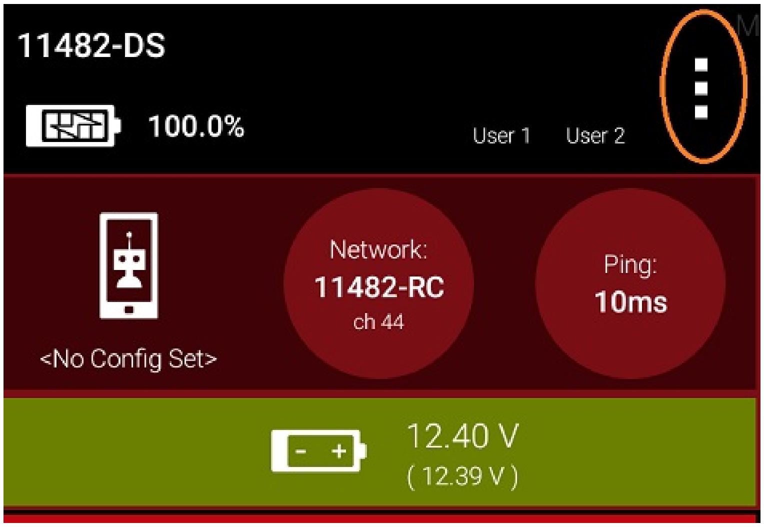
2. Press the *Upload* button to upload the firmware file from your computer to the Robot Controller.

The words "Firmware upload complete" should appear once the file has been uploaded successfully.

3. Make sure that your Expansion Hub is turned on and powered by a freshly charged 12V battery and that the Robot Controller phone is connected to the Expansion Hub through a USB connection. Note that the Robot Controller does **not** need to have the Expansion Hub included in an active configuration file in order for the update to work.

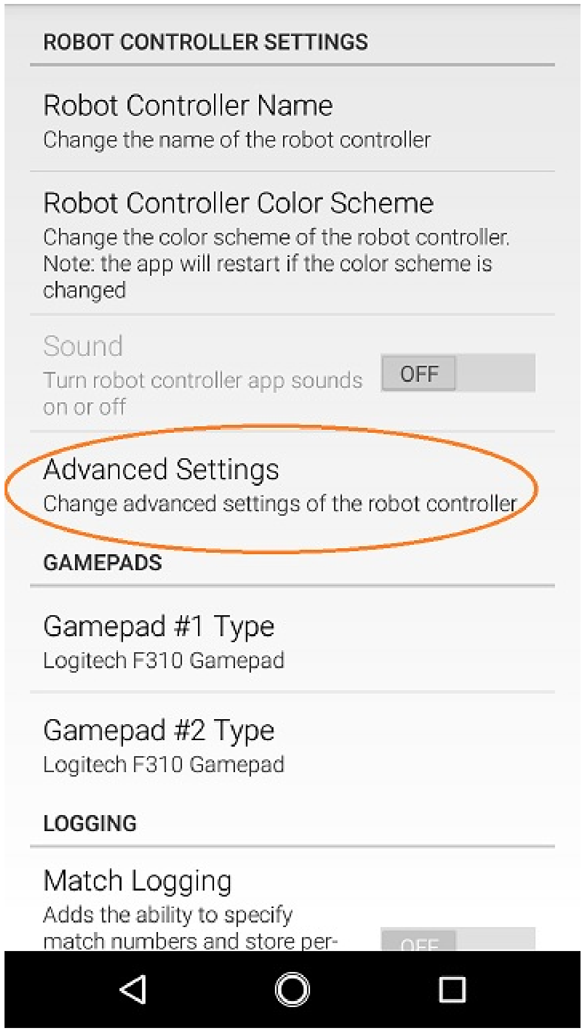


4. On the Driver Station, touch the three dots in the upper right hand corner to display a pop-up menu.

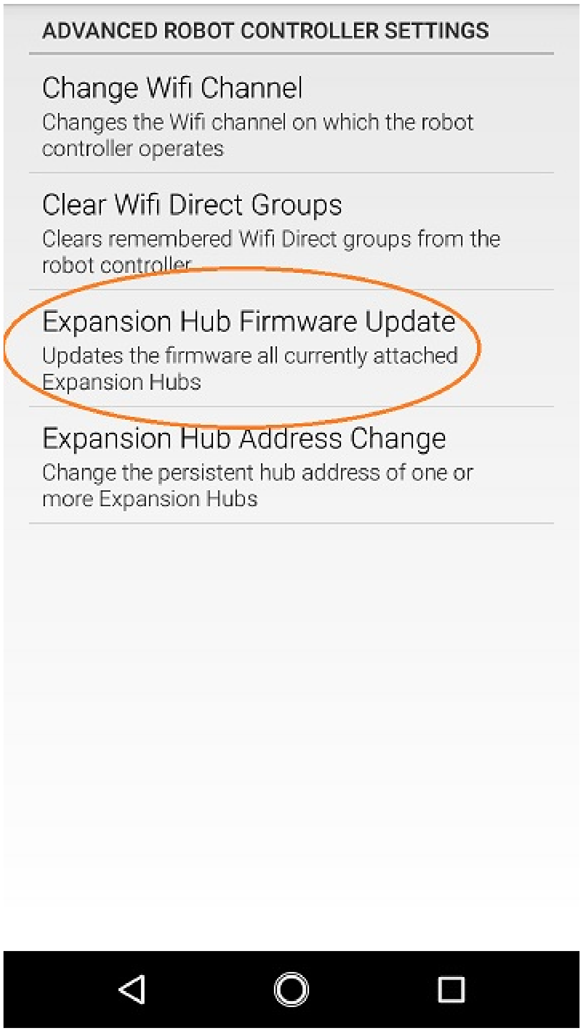


5. Select *Settings* from the pop-up menu to display the Settings activity.

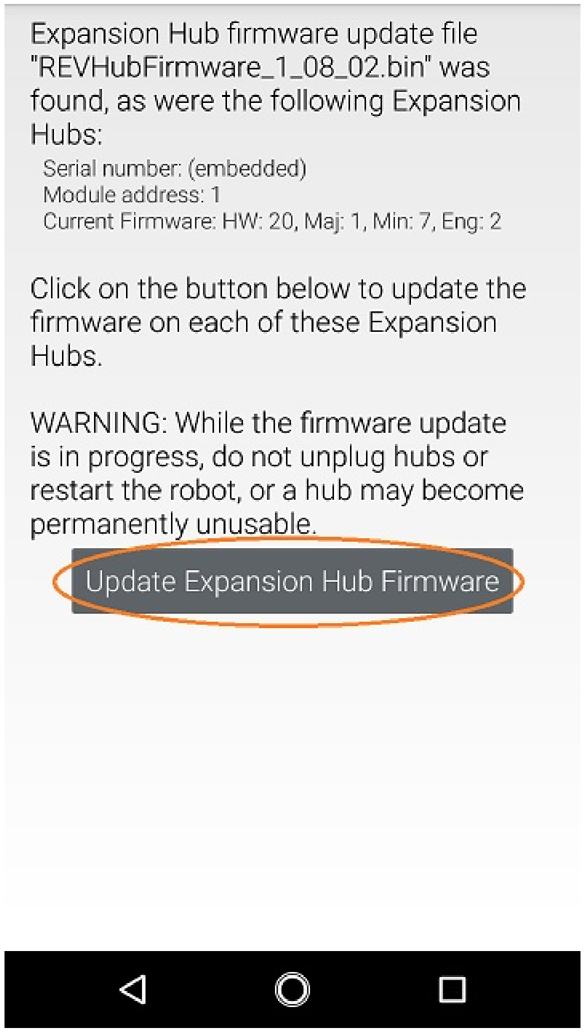
6. On the Driver Station, scroll down and select the *Advanced Settings* item (under the *ROBOT CONTROLLER SETTINGS* category).



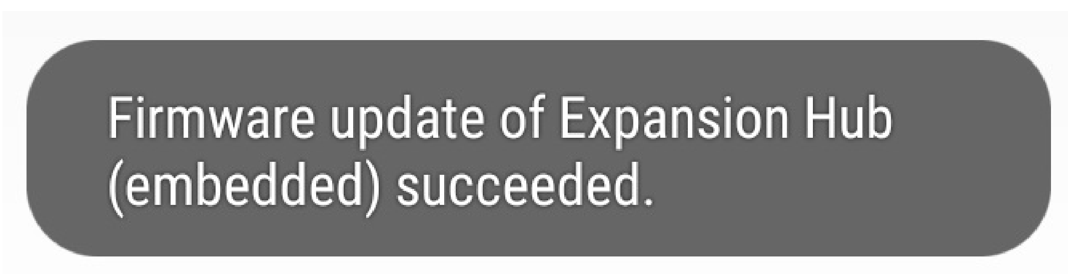
7. Select the *Expansion Hub Firmware Update* item on the *ADVANCED ROBOT CONTROLLER SETTINGS* activity.



8. If a firmware file that is different from the version currently installed on the Expansion Hub was successfully uploaded, the Driver Station should display some information about the current firmware version and the new firmware version. Press the *Update Expansion Hub Firmware* button to start the update process.



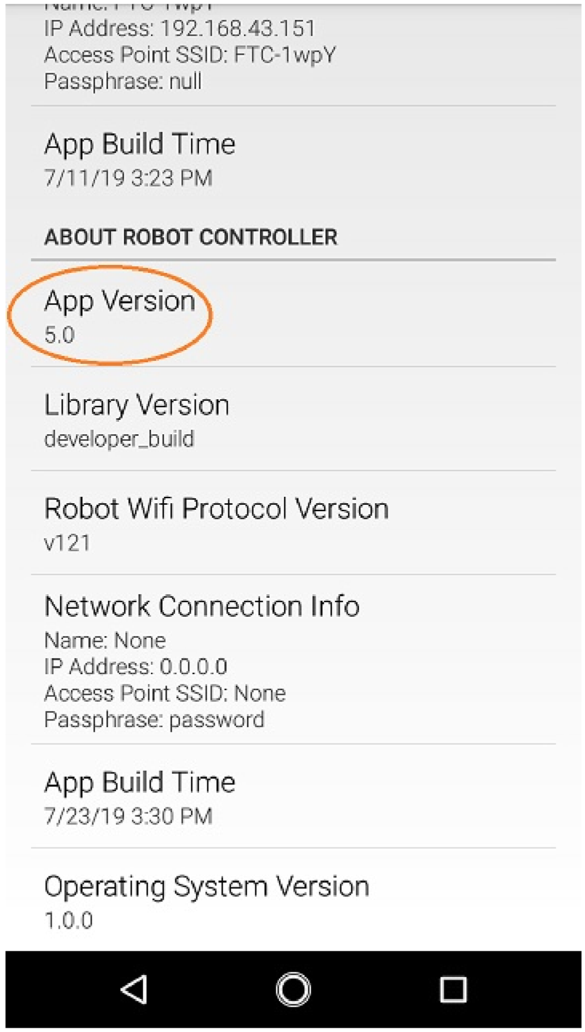
9. A progress bar will display while the firmware is being updated. Do not power off the Robot Controller/Expansion Hub during this process. The Driver Station will display a message when the update process is complete.



Updating the Robot Controller App

It is important to know how to update the Robot Controller app that is installed on your smartphone. FIRST periodically releases new versions of this app, which contain improvements and fixes, as well as season-specific data and features.

Note that you can see the Robot Controller app version number through the Robot Controller or Driver Station user interface. Select the *About* menu option on the Robot Controller or Driver Station and note the App Version number under the *ABOUT ROBOT CONTROLLER* section.



As of 2021, the apps (v 6.1 and higher) are no longer available on Google Play.

The REV Hardware Client software will allow you to download the apps to approved devices: REV Control Hub, REV Expansion Hub, REV Driver Hub, and approved Android devices. Here are some of the benefits:

- Connect a REV Control Hub via WiFi.
- One Click update of all software on connected devices.
- Pre-download software updates without a connected device.
- Back up and restore user data from Control Hub.
- Install and switch between DS and RC applications on Android Devices.

- Access the Robot Control Console on the Control Hub.

Teams using Blocks or OnBot Java for programming can use the REV Hardware Client to update the Robot Controller (RC) app on an RC phone.

Note it will take an estimated 7.5 minutes per device to complete this task.

As an alternate, the app releases are available on the FTCRobotController Github. Download the Robot Controller APK file to a computer, transfer it to the RC phone's Downloads folder, then open that file to install the RC app. This process is called "side-loading".

Tip: If you update the Robot Controller (RC) app, you should also update the Driver Station (DS) app to the same version number.

Important: Teams using Android Studio should not update the RC app with the REV Hardware Client or by side-loading. Instead, by updating to the newest version of the Android Studio project folder, you will update the Robot Controller app when you build the project and install it on your RC device. You can download the newest version of the project folder here.

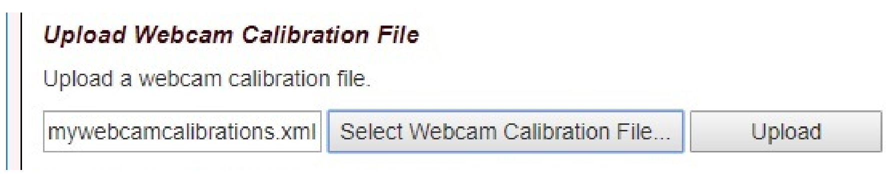
Uploading a Custom Webcam Calibration File

The Robot Controller app has built-in calibration information for a variety of commonly available webcams. Users can also create their own custom calibration files and then upload these files to a Control Hub.

A commented example of what the contents of a calibration file should look like can be found in a file called *teamwebcam-calibrations.xml*, which is included with the Android Studio project folder. This example calibration file can be found here.

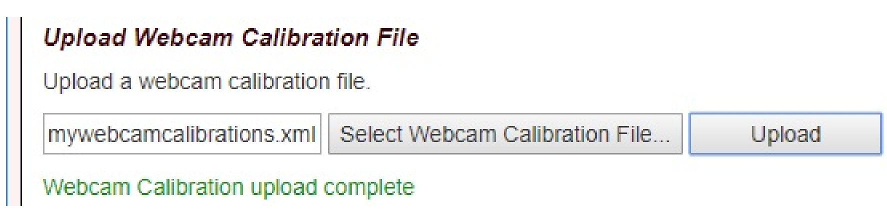
Uploading a Custom Webcam Calibration File Instructions

1. On the *Manage* page, click on the *Select Webcam Calibration File* button to select the calibration file.



   An *Upload* button should appear if a file was successfully selected.

2. Click on the *Upload* button to upload the selected file. If the upload was successful, then the *Manage* page will display a message indicating that the upload has completed.



21.3 AprilTag Programming

Topics for programming with AprilTags

21.3.1 AprilTag Introduction

Introduction

A popular camera-based technology is **AprilTag**, a scanned image similar to a QR Code. Its effectiveness and quick set-up on custom Signal Sleeves led to **wide adoption** in POWERPLAY (2022-2023) by *FIRST* Tech Challenge teams, especially those programming in Java.

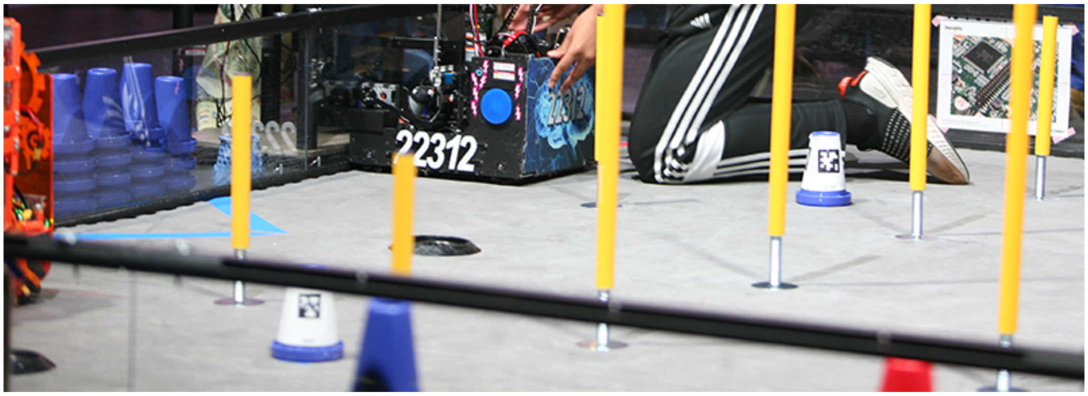


Fig. 27: Photo Credit: Mike Silversides

Those POWERPLAY teams, including those using FTC Blocks, learned how to use several resources:

- AprilTag: an open-source technology for evaluating formatted images
- EasyOpenCV: a *FIRST* Tech Challenge-optimized interface with OpenCV, an image processing library
- myBlocks: custom Blocks created in OnBot Java (OBJ)

Now these three areas are provided, or bundled, in the new *FIRST* **Tech Challenge Software Development Kit (SDK), version 8.2**.

Namely, key capabilities of **AprilTag** and **EasyOpenCV** are available to the Robot Controller (RC) and Driver Station (DS) apps, without special downloads. And AprilTag features are included in **FTC Blocks**, without needing custom myBlocks.

The AprilTag features work on Android RC phone cameras, and on webcams. A single OpMode can use AprilTag and TensorFlow Object Detection (TFOD).

In *FIRST* Tech Challenge, AprilTag is ready for the spotlight!

What is AprilTag?

Developed at the University of Michigan, AprilTag is like a 2D barcode or a simplified QR Code. It contains a numeric **ID code** and can be used for **location and orientation**.

AprilTag is a type of **visual fiducial**, or fiducial marker, containing information and designed for easy recognition.

The above samples represent different formats, or **families**. A project typically uses a single AprilTag family.

This year, *FIRST* Tech Challenge uses a common family called **36h11**. A PDF showing the numbers 0 through 20 from the 36h11 family can be downloaded here:

Fig. 28: AprilTags on Robots. Photo Credit: University of Michigan

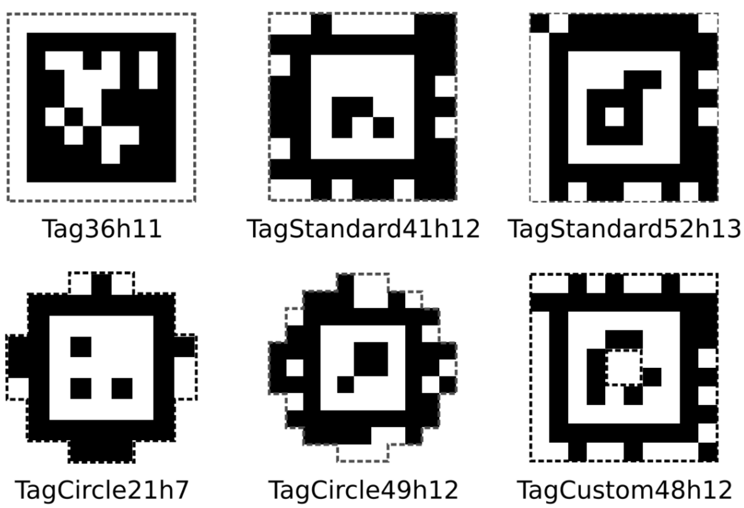


Fig. 29: A sample of different AprilTag families

- `AprilTag PDF 0-20`

Each number is the ID code of that tag.

Here's an AprilTag representing **ID code 2**. The SDK software recognizes and overlays the ID code onto the image (small blue rectangle **ID 02**).

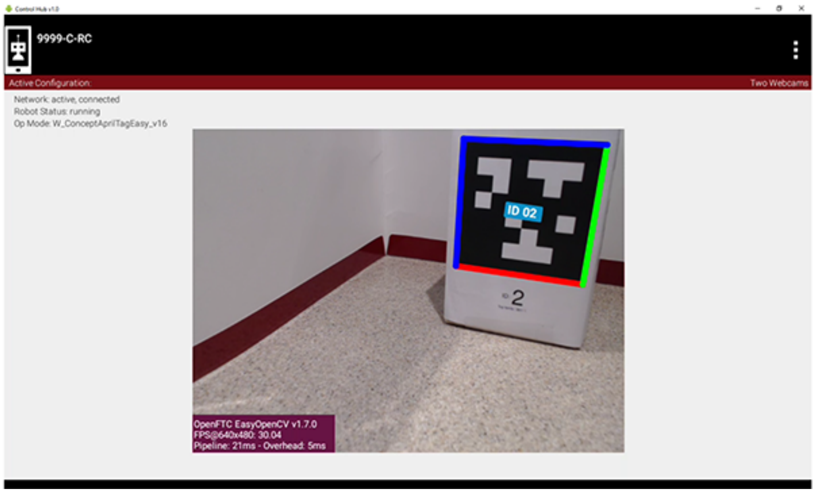


Fig. 30: Stream output showing the detected tag ID 02

The above image shows a camera preview image, called LiveView, from a Robot Controller device (Control Hub or RC phone).

The AprilTag family 36h11 has a capacity of 587 ID codes. To see them all, follow this link:

- https://github.com/rgov/apriltag-pdfs/tree/main/tag36h11/us_letter/100mm

The square AprilTag pattern contains smaller black and white squares, each called a **pixel**. A 36h11 tag contains 10 x 10 pixels, including an outer border of **all white pixels** and an inner border of **all black pixels**.

Tag size is measured across the outside edge of the **inner border** which comprises the black pixels for 36h11.

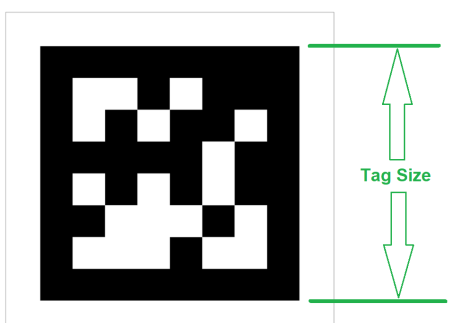


Fig. 31: Figure demonstrating the tag size measurement

The above image shows a complete AprilTag with outer white border. From the 36h11 family, its ID code is 42.

AprilTag Pose

Beyond ID code, the new SDK also provides **pose** data, namely position and orientation (rotation) from the **camera's point of view**. This requires a **flat AprilTag**, which was not possible with curved POWERPLAY Signal Sleeves.

Let's look again at the camera preview image, called LiveView, from a Robot Controller device (Control Hub or RC phone).

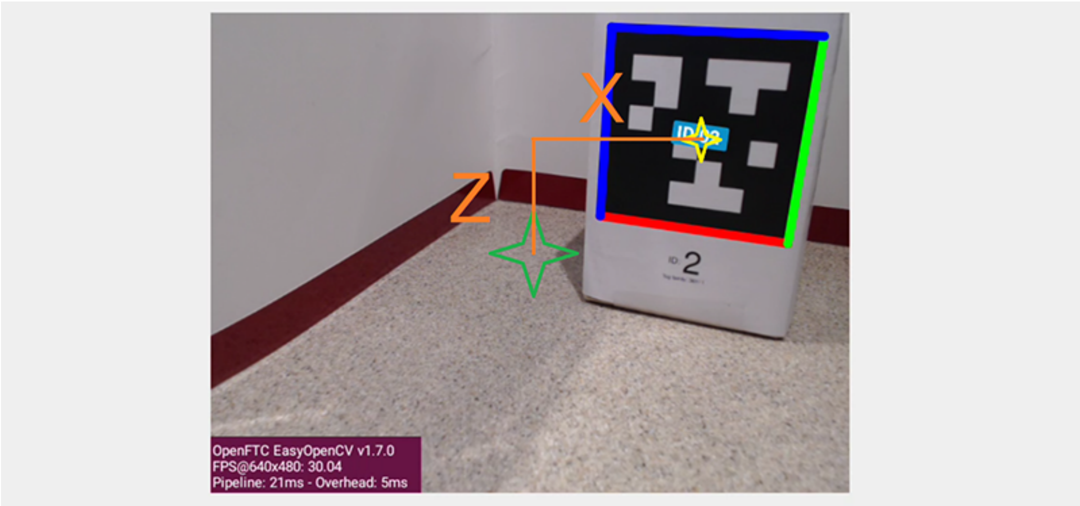


Fig. 32: LiveView Image with additional markings for explanation purposes

Imagine a laser beam pointing straight outward from the center of the camera lens. Its 3-dimensional path appears (to the camera) as a single point, indicated by the **green star**. You can see that the center of the AprilTag (**yellow star**) is offset from that "laser beam".

That **translation offset** can break down into three traditional components (X, Y and Z distances), along axes at 90 degrees to each other:

- X distance (horizontal orange line) is from the center, to the right
- Y distance (not shown) is from the lens center, outwards
- Z distance (vertical orange line) is from the center, upwards

The SDK provides these distances **in the real world**, not just reporting how many pixels on the screen. Very useful!

You can also see that the AprilTag's flat face is not parallel to the plane of the camera. That **rotation offset** can break down into three angles about the X, Y and Z axes. This is discussed further in the section below, called **AprilTag Axes**.

In summary, the SDK evaluates the AprilTag image and performs **"pose estimation"**, providing an estimated X, Y and Z **distance** between the tag and the camera, along with an estimated **angle** of rotation around those axes. A closer or larger AprilTag can yield a more accurate estimate of pose.

To provide good pose estimates, each RC phone camera or webcam requires **calibration data**, for a specific resolution. The SDK contains such data for a limited number of webcams and resolutions. Teams can generate their own data, called **lens intrinsics**, using a provided procedure.

Navigation

OpModes use AprilTag pose to achieve **navigation**: evaluating inputs and driving to a destination.

An OpMode can use pose data to drive towards the tag, or drive to a target position and orientation **relative to the tag**. (The new SDK provides Java **Sample OpModes** `RobotAutoDriveToAprilTagOmni.java` and `RobotAutoDriveToAprilTagTank.java`.) Another navigation possibility is mentioned below under **Advanced Use**.

Navigation is best done with **continuous** pose estimates, if the AprilTag remains within the camera's field of view. Namely, an OpMode "**while() loop**" should regularly read the updated pose data, to guide the robot's driving actions.

The new SDK supports **multiple cameras**, switchable or simultaneous. This can help if the robot changes direction, or you wish to navigate using another AprilTag (or TensorFlow object).

Other sensors can also be used for navigation, such as drive motor encoders, REV Hub IMU, deadwheel encoders, color/distance sensors, ultrasonic sensors, and more.

It's also possible to evaluate **non-AprilTag images** from the same camera and/or a second camera. For example, the SDK can estimate the horizontal angle (or Bearing) of an object detected with **TensorFlow**, another vision technology employed in *FIRST* Tech Challenge. Advanced teams might consider active camera pointing control, to keep an AprilTag or other object in view.

Annotations

In the preview (RC phone screen or DS Camera Stream), an official recognized AprilTag will display a **colored border** and its numeric **ID code**. These **annotations** allow easy visual confirmation of recognition:

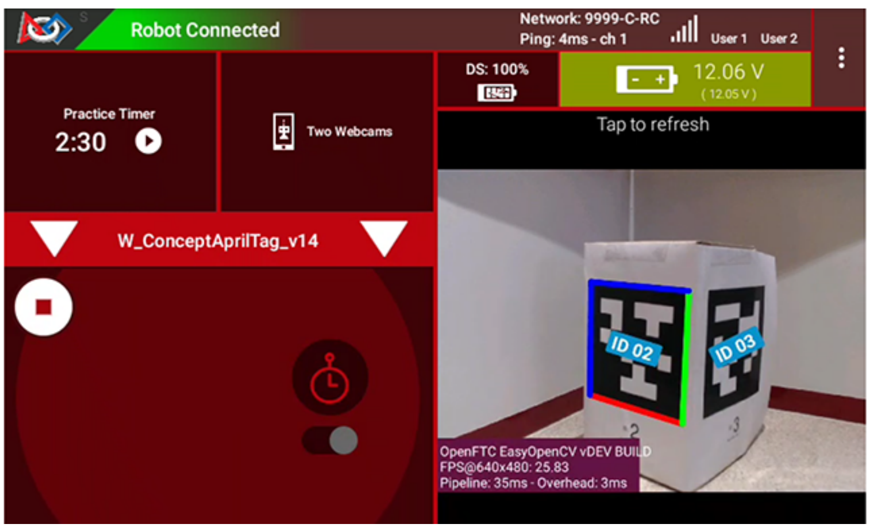


Fig. 33: Two AprilTags with different metadata being detected and annotations displayed

In the above *DS Camera Stream* preview, the left-side AprilTag was recognized from a tag **Library** (default or customized). A Library tag has pre-loaded information (called **Metadata**) including its tag size, which allows **pose estimation**. These are annotated by default with a **colored border**.

The right-side AprilTag was not in a tag Library. It has no Metadata, so the SDK can provide only its numeric **ID code**, shown here as **ID 03**. Such tags are **not** annotated by default with a colored border.

Note: **Camera Stream** displays a snapshot of the camera's view, on the Driver Station device. It's available only during the INIT phase of an OpMode, and also shows any AprilTag (or TFOD) annotations. Instructions are posted here:

- *Camera Stream Image Preview Documentation*

Optional annotations include **colored axes** at the tag center, and a **colored box** projecting from the tag image:

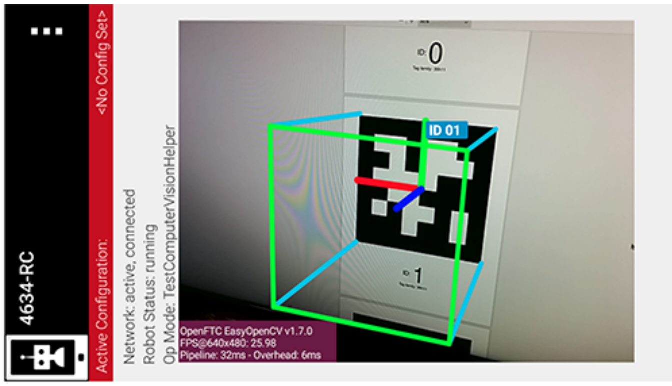


Fig. 34: LiveView with additional annotations enabled

The above image shows a preview (called LiveView) on an Android Robot Controller (RC) phone. The REV Control Hub does generate an RC preview, which can be seen with an HDMI external monitor, or with `scrcpy` which can be found here:

- https://github.com/Genymobile/scrcpy

AprilTag Axes

The SDK now provides the underlying pose data as follows:

- Position is based on X, Y and Z distance **from the camera lens to the AprilTag**.
- Orientation is based on rotation about those axes, using the right-hand rule.

Note: the optional red-green-blue annotated axes represent the **tag's frame of reference**, unrelated to SDK pose data. That annotation indicates only a successful AprilTag recognition.

Here are the axis designations in the new SDK:

- Y axis points **straight outward** from the camera lens center
- X axis points **to the right**, perpendicular to the Y axis
- Z axis points **upward**, perpendicular to Y and X

If the camera is upright and pointing forward on the robot, these axes are consistent with the Robot Coordinate System used for *IMU navigation*.

Note: these axes are different than the official AprilTag definitions, even from the camera's frame of reference.

The SDK provides AprilTag **rotation** data as follows:

- **Pitch** is the measure of rotation about the X axis
- **Roll** is the measure of rotation about the Y axis
- Heading, or **Yaw**, is the measure of rotation about the Z axis

Rotation follows the traditional right-hand rule: with the thumb pointing along the positive axis, the fingers curl in the direction of positive rotation.

Further discussion is provided here:

Understanding AprilTag Detection Values

Note: This article does not discuss the *FIRST* Tech Challenge *Field Coordinate System*.

Your OpModes might relate robot orientation to the overall field or `'global coordinates'` for navigation, but that's beyond this AprilTag introduction.

Advanced Use

Option 1

If a tag's position and orientation **on the game field** are specified in advance, the tag's pose data could be used by an advanced OpMode to calculate the robot's position on the field. This conversion math, an exercise for the reader, can allow a robot to use the tag's pose data in real-time to navigate to the desired location on the field.

Option 2

Vision processing can consume significant **CPU resources** and USB communications **bandwidth**. *FIRST* Tech Challenge teams may balance the benefits of higher resolution and speed (frames-per-second) against the risk of overloading CPU and bandwidth resources. The 8.2 SDK provides numerous tools to manage this balance:

- select the camera resolution
- disable and enable the RC preview (called LiveView)
- disable and enable the AprilTag (or TFOD) processor
- close the camera stream
- select a compressed video streaming format
- measure frames-per-second
- set decimation (down-sampling)
- select a pose solver algorithm

Option 3

Clearer camera images can improve AprilTag (and TFOD) vision processing. The SDK offers powerful **webcam controls** (Exposure, Gain, Focus, and more), now available in FTC Blocks! These controls can be applied under various lighting conditions.

Exposure and Gain are adjusted together. The new SDK offers Java Sample OpMode `ConceptAprilTagOptimizeExposure.java`.

Option 4

The frame of reference described above in **AprilTag Axes** is calculated and provided by default in the new 8.2 SDK. Advanced teams may prefer to perform their own pose calculations, based on **raw values** from the AprilTag/EasyOpenCV pipeline.

Those raw values are available to Java and Blocks programmers. The Java version is shown here:

```
for (AprilTagDetection detection : aprilTag.getDetections())  {

     Orientation rot = Orientation.getOrientation(detection.rawPose.R, AxesReference.INTRINSIC,␣
↪AxesOrder.XYZ, AngleUnit.DEGREES);

     // Original source data
     double poseX = detection.rawPose.x;
     double poseY = detection.rawPose.y;
     double poseZ = detection.rawPose.z;

     double poseAX = rot.firstAngle;
     double poseAY = rot.secondAngle;
     double poseAZ = rot.thirdAngle;
     }
```

These raw values are converted by the SDK to the default interface, as follows:

```
if (detection.rawPose != null)   {
     detection.ftcPose = new AprilTagPoseFtc();
```

(continues on next page)

(continued from previous page)

```
    detection.ftcPose.x =  detection.rawPose.x;
    detection.ftcPose.y =  detection.rawPose.z;
    detection.ftcPose.z = -detection.rawPose.y;

    Orientation rot = Orientation.getOrientation(detection.rawPose.R, AxesReference.INTRINSIC,␣
↪AxesOrder.YXZ, outputUnitsAngle);
    detection.ftcPose.yaw = -rot.firstAngle;
    detection.ftcPose.roll = rot.thirdAngle;
    detection.ftcPose.pitch = rot.secondAngle;

    detection.ftcPose.range = Math.hypot(detection.ftcPose.x, detection.ftcPose.y);
    detection.ftcPose.bearing = outputUnitsAngle.fromUnit(AngleUnit.RADIANS, Math.atan2(-detection.
↪ftcPose.x, detection.ftcPose.y));
    detection.ftcPose.elevation = outputUnitsAngle.fromUnit(AngleUnit.RADIANS, Math.
↪atan2(detection.ftcPose.z, detection.ftcPose.y));
    }
```

Again, further discussion is provided here:

Understanding AprilTag Detection Values

Summary

AprilTag is a popular camera-based technology, using a scanned image similar to a QR Code.

The new SDK version 8.2 now includes key capabilities of AprilTag and EasyOpenCV, a *FIRST* Tech Challenge-optimized interface with OpenCV for image processing. These methods are packaged for convenient use by **Java and Blocks programmers**.

By default, the SDK can detect the ID code for any AprilTag in the 36h11 family.

For AprilTags in a default or custom tag Library, the interface provides calculated **pose** estimates (position and rotation) from the **camera's frame of reference**. The source data is also available for advanced teams.

The AprilTag features work on Android RC phone cameras, and on webcams. Each camera requires **calibration data**, for a specific resolution, to provide good pose estimates.

Multiple cameras are supported, and a single OpMode can use AprilTag and TensorFlow Object Detection (TFOD). AprilTag detection is improved with webcam Camera Controls, now available also in FTC Blocks.

In *FIRST* Tech Challenge, AprilTag is ready to take CENTERSTAGE!

Much credit to:

- EasyOpenCV developer @Windwoes
- FTC Blocks developer @lizlooney
- FTC navigation expert @gearsincorg
- and the smart people at UMich/AprilTag.

Questions, comments and corrections to westsiderobotics@verizon.net

21.3.2 VisionPortal Overview

FIRST Tech Challenge introduces **VisionPortal**, a comprehensive new interface for vision processing.

- For **FTC Blocks and Java** teams, VisionPortal offers key capabilities of **AprilTag** and **EasyOpenCV**, along with **TensorFlow Object Detection (TFOD)** – at the same time!

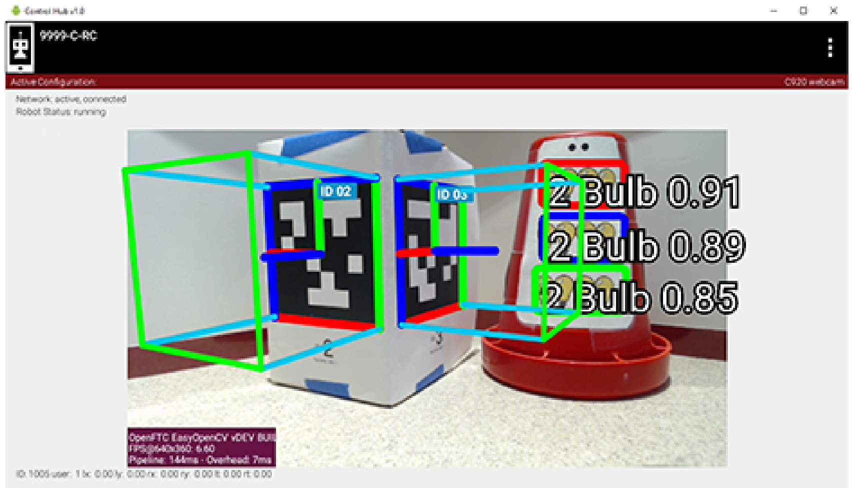


Fig. 35: Dual Preview with both AprilTags and TensorFlow

- **AprilTag** detections include ID code and **pose**: tag location and orientation, relative to the camera.
- **Camera Controls**, which can improve AprilTag and TFOD performance for webcam, are now fully available to **FTC Blocks** users.
- **Multiple cameras** can operate at the same time – phone camera and/or webcam.

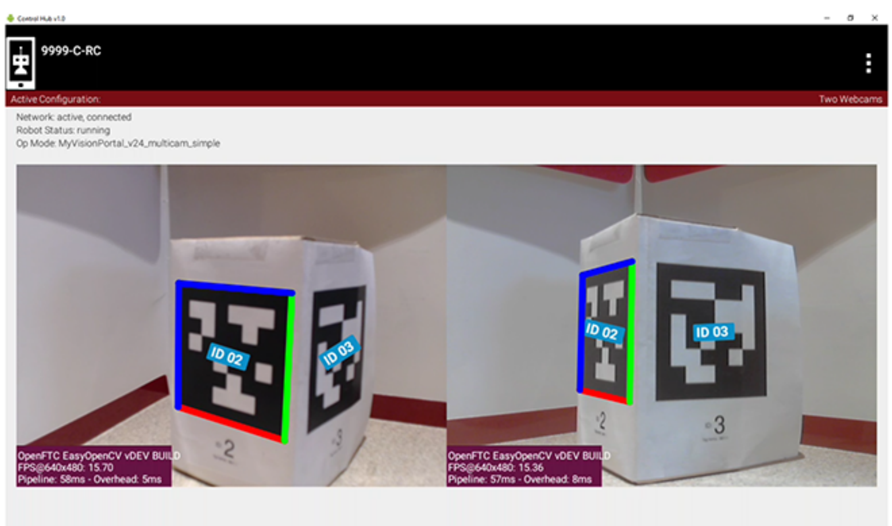


Fig. 36: Multiple Camera View

- **Sample OpModes** and new tools are available to operate and customize these features, including the **Builder pattern**.
- For heavy video processing, many options are available to manage **CPU resources** and **USB bandwidth**.
- DS and RC previews can be **BIG**!

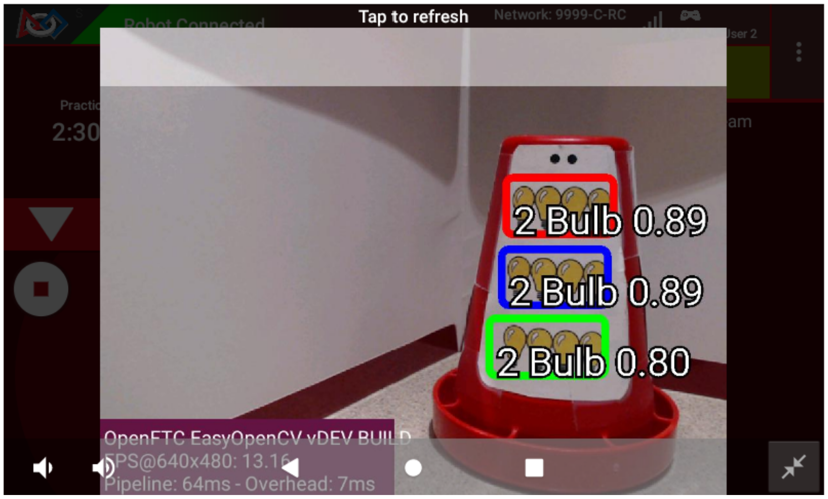


Fig. 37: Full Screen Preview

Many other new and improved features await your discovery in VisionPortal and beyond.

In preparation for the 2023-2024 CENTERSTAGE season, the new Software Development Kit (SDK) **VisionPortal** includes **built-in support for AprilTag technology**. Previously, Teams needed to download and incorporate external libraries, complicating the programming effort.

AprilTag is a popular vision technology for detecting a simple black-and-white tag, used to estimate **position and orientation**. In the 2022-2023 POWERPLAY game, many Teams enjoyed AprilTag's reliable Autonomous performance for Signal Sleeve recognition.

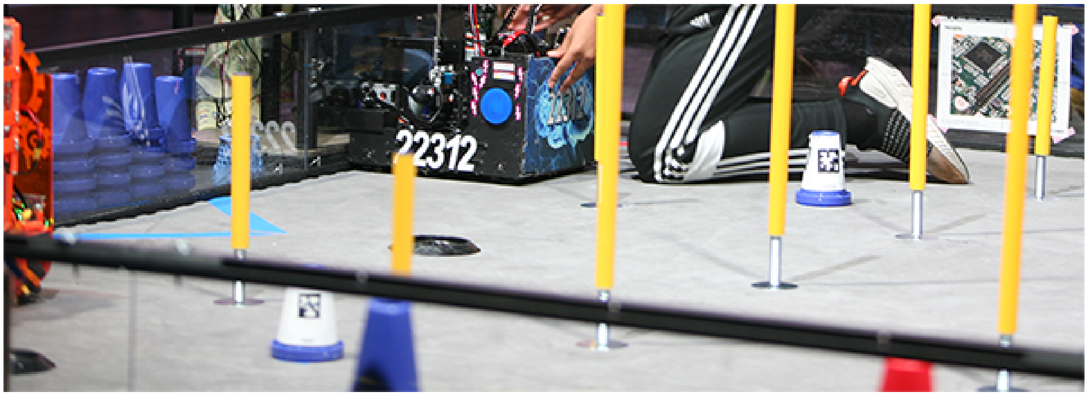


Fig. 38: Photo Credit: Mike Silversides

All sections of this Guide assume prior reading of the *AprilTag Introduction* **.**

The SDK describes AprilTag pose **relative to the camera**, by default. This computing process is called **pose estimation**, a term that emphasizes this is an estimate only, based on many factors including **camera calibration**. You must determine AprilTag's best use for reaching your goals.

Vision Processor Initialization

Processor Initialization - Overview

Your OpMode must **first prepare** for using AprilTag and/or TensorFlow Object Detection (TFOD) commands, or methods.

In the INIT portion of your Java or Blocks code, before `waitForStart()`, use these steps:

- **Step 1.** *Optional:*
  - Supplement the default **AprilTag Library** with additional tags. This task is not shown in the Sample OpModes, and is covered at the **Library** page (not here).
- **Step 2.** *Required:*
  - Create the **AprilTag Processor** (or the **TFOD Processor**), to analyze frames streaming in from the camera. "Under the hood", the Apriltag Processor is attached to an EOCV **pipeline**, which performs various steps, in order, to each stream frame. The stream is the input to the pipeline. A similar process happens for TFOD.
- **Step 3.** *Required:*
  - Create the FTC **VisionPortal**, to manage the overall pipeline. Here you specify that the Portal includes the AprilTag and/or TFOD Processor(s) from Step 2. The two Processors evaluate camera frames independently.

This page describes Step 2 in more detail, for both Processors. The *VisionPortal Init* page covers Step 3, **VisionPortal Initialization**.

AprilTag Initialization - Easy

Step 2 is creating the **AprilTag Processor**, software that evaluates frames streaming in from the camera.

The SDK provides an "easy" way to create the processor, using only **defaults** and not mentioning a "Builder":

Blocks

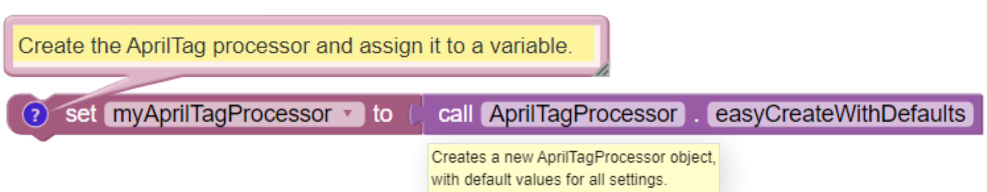


Fig. 39: Easy AprilTag Processor Initialization without a Builder

Java

Example of Easy AprilTag Processor Initialization without a Builder

```
AprilTagProcessor myAprilTagProcessor;
// Create the AprilTag processor and assign it to a variable.
myAprilTagProcessor = AprilTagProcessor.easyCreateWithDefaults();
```

AprilTag Initialization - Builder

The SDK also provides the "Builder" way to create the processor, allowing **custom settings**.

Builder is a Java pattern or structure for adding features or parameters, finalized with the `.build()` command. Such features are **not** modified later during an OpMode.

Inside the SDK, even the "easy" process uses the Builder pattern to set the default parameters.

Blocks

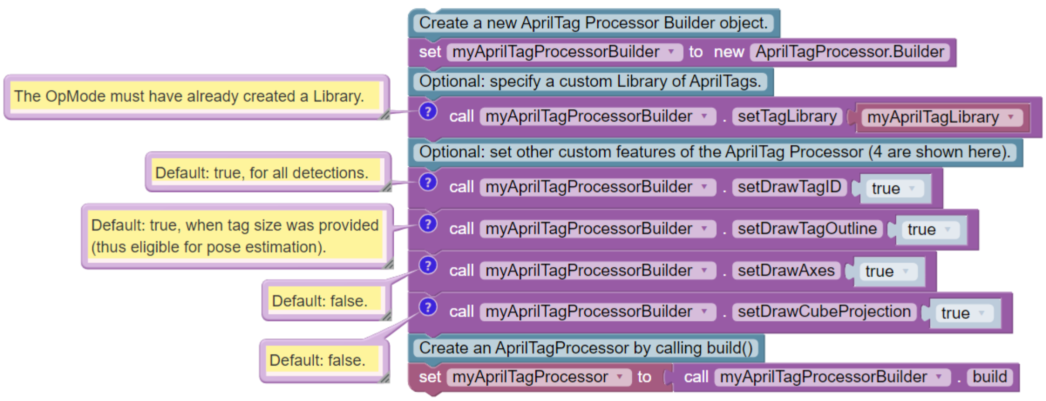


Fig. 40: AprilTag Processor Initialization with a Builder

Java

```
AprilTagProcessor.Builder myAprilTagProcessorBuilder;
AprilTagProcessor myAprilTagProcessor;

// Create a new AprilTag Processor Builder object.
myAprilTagProcessorBuilder = new AprilTagProcessor.Builder();

// Optional: specify a custom Library of AprilTags.
myAprilTagProcessorBuilder.setTagLibrary(myAprilTagLibrary);   // The OpMode must have already␣
↪created a Library.

// Optional: set other custom features of the AprilTag Processor (4 are shown here).
myAprilTagProcessorBuilder.setDrawTagID(true);       // Default: true, for all detections.
myAprilTagProcessorBuilder.setDrawTagOutline(true);  // Default: true, when tag size was provided␣
↪(thus eligible for pose estimation).
myAprilTagProcessorBuilder.setDrawAxes(true);        // Default: false.
myAprilTagProcessorBuilder.setDrawCubeProjection(true);        // Default: false.
```

(continues on next page)

(continued from previous page)

```
// Create an AprilTagProcessor by calling build()
myAprilTagProcessor = myAprilTagProcessorBuilder.build();
```

This example shows only 4 AprilTag Processor Builder features; others are available.

As seen above, Step 2 must specify any custom (non-default) Library from the optional Step 1 - otherwise the Processor will include only the default Library.

AprilTag Java Builder Chain

The Builder pattern can be implemented in a streamlined manner, using Java. The following code is equivalent to the above individual method calls.

Comments are omitted here, to clearly illustrate the chaining.

```
AprilTagProcessor myAprilTagProcessor;

myAprilTagProcessor = new AprilTagProcessor.Builder()
    .setTagLibrary(myAprilTagLibrary)
    .setDrawTagID(true)
    .setDrawTagOutline(true)
    .setDrawAxes(true)
    .setDrawCubeProjection(true)
    .build();
```

Here the object `myAprilTagProcessorBuilder` was not created; the build was performed directly on `myAprilTagProcessor`. The Builder pattern allows the "dot" methods to be chained in a single Java statement ending with `.build()`.

TensorFlow Initialization - Easy

Step 2 is similar for creating the **TensorFlow TFOD Processor**, software that evaluates frames streaming in from the camera.

The SDK provides an "easy" way to create the processor, using only **defaults** and not mentioning a "Builder":

Blocks

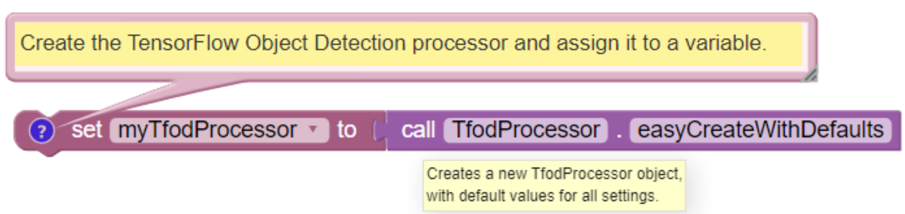


Fig. 41: Easy TensorFlow TFOD Processor Initialization without a Builder

Java

Example of TensorFlow TFOD Processor Initialization without a Builder

```
TfodProcessor myTfodProcessor;
// Create the TensorFlow Object Detection processor and assign it to a variable.
myTfodProcessor = TfodProcessor.easyCreateWithDefaults();
```

TensorFlow Initialization - Builder

The SDK also provides the "Builder" way to create the processor, allowing **custom settings**.

Builder is a Java pattern or structure for adding features or parameters, finalized with the `.build()` command. Such features are **not** modified later during an OpMode.

Inside the SDK, even the "easy" process uses the Builder pattern to set the default parameters.

Blocks

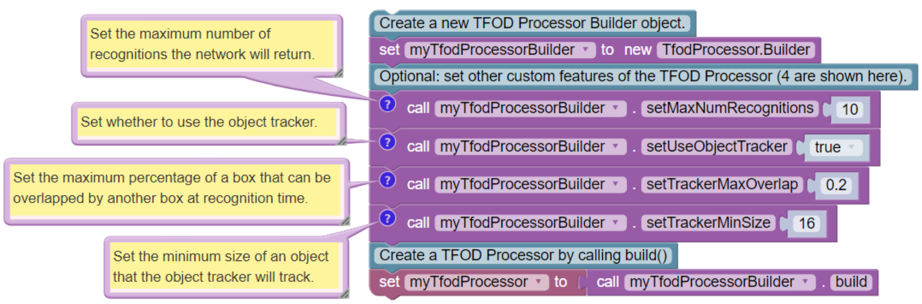


Fig. 42: TensorFlow TFOD Processor Initialization with a Builder

Java

```
TfodProcessor.Builder myTfodProcessorBuilder;
TfodProcessor myTfodProcessor;

// Create a new TFOD Processor Builder object.
myTfodProcessorBuilder = new TfodProcessor.Builder();

// Optional: set other custom features of the TFOD Processor (4 are shown here).
myTfodProcessorBuilder.setMaxNumRecognitions(10);  // Max. number of recognitions the network will
↪return
myTfodProcessorBuilder.setUseObjectTracker(true);  // Whether to use the object tracker
myTfodProcessorBuilder.setTrackerMaxOverlap((float) 0.2);  // Max. % of box overlapped by another
↪box at recognition time
myTfodProcessorBuilder.setTrackerMinSize(16);  // Min. size of object that the object tracker will
↪track

// Create a TFOD Processor by calling build()
myTfodProcessor = myTfodProcessorBuilder.build();
```

This example shows only 4 TFOD Processor Builder features; others are available. Most others relate to custom TFOD Models, beyond this scope of this VisionPortal Guide.

TensorFlow Java Builder Chain

The Builder pattern can be implemented in a streamlined manner, using Java. The following code is equivalent to the above individual method calls.

Comments are omitted here, to clearly illustrate the chaining.

```
TfodProcessor myTfodProcessor;

myTfodProcessor = new TfodProcessor.Builder()
    .setMaxNumRecognitions(10)
    .setUseObjectTracker(true)
    .setTrackerMaxOverlap((float) 0.2)
    .setTrackerMinSize(16)
    .build();
```

Here the object `myTfodProcessorBuilder` was not created; the build was performed directly on `myTfodProcessor`. The Builder pattern allows the "dot" methods to be chained in a single Java statement ending with `.build()`.

Enabling and Disabling Processors

For a Processor created here at Step 2, an OpMode does **not need** to enable that Processor at the following Step 3, **Vision-Portal Initialization**.

The `setProcessorEnabled()` command is **not** part of the Builder pattern.

Use `setProcessorEnabled( , true)` only to **re-enable** the processor, after **disabling** (by setting to `false`). This topic is covered further at the **Managing CPU and Bandwidth** page.

At the following page's Step 3, the `addProcessor()` command **automatically enables** the specified processor. Thus Op-Modes **do not initialize** with this, after Step 2:

Blocks

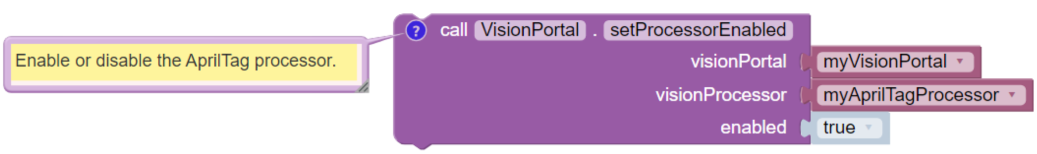


Fig. 43: Enable or Disable AprilTag Processor

Again, use this only to **re-enable** the processor, after **disabling** (by setting to *false*).

Java

```
// Enable or disable the AprilTag processor.
myVisionPortal.setProcessorEnabled(myAprilTagProcessor, true);
```

Again, use this only to **re-enable** the processor, after **disabling** (by setting to *false*).

Questions, comments and corrections to westsiderobotics@verizon.net

VisionPortal Initialization

Overview

Here we describe Step 3, **creating a VisionPortal**, to allow an OpMode to use AprilTag and/or TensorFlow Object Detection (TFOD). This continues from the previous page *Vision Processor Initialization*, which described Step 2: creating an AprilTag Processor and/or a TensorFlow Object Detection (TFOD) Processor. The two Processors evaluate camera frames independently.

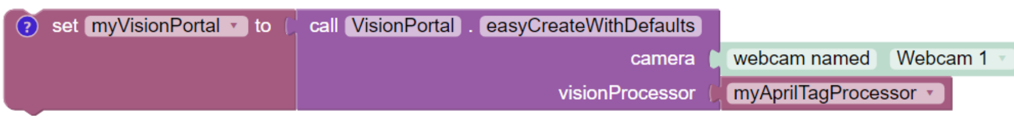
Steps 1, 2 and 3 are typically performed in the OpMode's INIT section, before the `waitForStart()` method or Block.

After this Step 3, actual use of AprilTag and TFOD can begin – before or after the DS Start button is touched.

VisionPortal Initialization - Easy

The SDK provides an "easy" way to make VisionPortal, using only **defaults** and not mentioning a "Builder":

Blocks



The FTC Blocks VisionPortal toolbox, or palette, offers "Easy Create" Blocks for:

- AprilTag or TFOD (or both)
- webcam, built-in RC phone camera, or "Switchable Camera Name"

That's 3 x 3 = 9 total choices, all "Easy".

Java

```java
VisionPortal myVisionPortal;

// Create a VisionPortal, with the specified camera and AprilTag processor, and assign it to a variable.
myVisionPortal = VisionPortal.easyCreateWithDefaults(hardwareMap.get(WebcamName.class, "Webcam 1"), myAprilTagProcessor);
```

To also use TFOD in the same OpMode, simply add it like this example:

```java
myVisionPortal = VisionPortal.easyCreateWithDefaults(hardwareMap.get(WebcamName.class, "Webcam 1"), myAprilTagProcessor, myTfodProcessor);
```

VisionPortal Initialization - Builder

The SDK also provides the "Builder" way to make VisionPortal, allowing **custom settings**:

Blocks

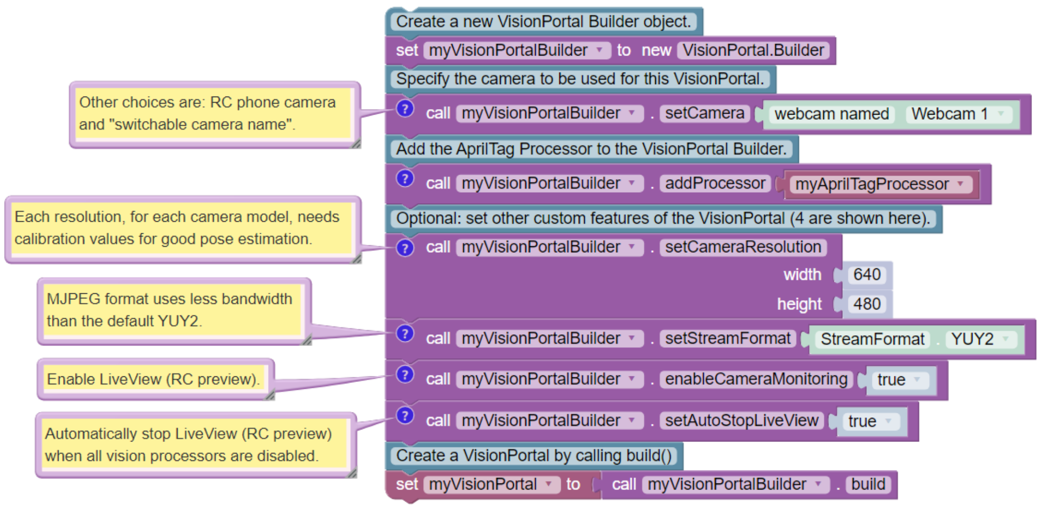


Fig. 44: VisionPortal Initialization with a Builder

Java

```java
VisionPortal.Builder myVisionPortalBuilder;
VisionPortal myVisionPortal;

// Create a new VisionPortal Builder object.
myVisionPortalBuilder = new VisionPortal.Builder()

// Specify the camera to be used for this VisionPortal.
myVisionPortalBuilder.setCamera(hardwareMap.get(WebcamName.class, "Webcam 1"));      // Other choices are: RC phone camera and "switchable camera name".

// Add the AprilTag Processor to the VisionPortal Builder.
myVisionPortalBuilder.addProcessor(myAprilTagProcessor);       // An added Processor is enabled by default.

// Optional: set other custom features of the VisionPortal (4 are shown here).
myVisionPortalBuilder.setCameraResolution(new Size(640, 480));  // Each resolution, for each camera model, needs calibration values for good pose estimation.
myVisionPortalBuilder.setStreamFormat(VisionPortal.StreamFormat.YUY2);  // MJPEG format uses less bandwidth than the default YUY2.
myVisionPortalBuilder.enableCameraMonitoring(true);      // Enable LiveView (RC preview).
myVisionPortalBuilder.setAutoStopLiveView(true);     // Automatically stop LiveView (RC preview) when all vision processors are disabled.

// Create a VisionPortal by calling build()
myVisionPortal = myVisionPortalBuilder.build();
```

This example shows only 4 VisionPortal Builder features; others are available.

To also use TFOD in the same OpMode, simply insert its `addProcessor(myTfodProcessor)` Block or Java method.

The SDK allows multiple, fully capable Portals. This is covered separately at the **MultiPortal** page.

Java Builder Chain

The Builder pattern can be implemented in a streamlined manner, using Java. The following code is equivalent to the above individual method calls.

Comments are omitted here, to clearly illustrate the chaining.

```
VisionPortal myVisionPortal;

myVisionPortal = new VisionPortal.Builder()
    .setCamera(hardwareMap.get(WebcamName.class, "Webcam 1"))
    .addProcessor(myAprilTagProcessor)
    .setCameraResolution(new Size(640, 480))
    .setStreamFormat(VisionPortal.StreamFormat.YUY2)
    .enableCameraMonitoring(true)
    .setAutoStopLiveView(true)
    .build();
```

Here the object `myVisionPortalBuilder` was not created; the build was performed directly on `myVisionPortal`. The Builder pattern allows the "dot" methods to be chained in a single Java statement ending with `.build()`.

Enabling and Disabling Processors

This note is repeated from the previous page 2, *Vision Processor Initialization*

For a Processor created at Step 2, an OpMode does **not need** to enable that Processor at this Step 3, **VisionPortal Initialization**.

The `setProcessorEnabled()` command is **not** part of the Builder pattern.

Use `setProcessorEnabled( , true)` only to **re-enable** the processor, after **disabling** (by setting to `false`). This topic is covered further at the **Managing CPU and Bandwidth** page.

At this page's Step 3, the `addProcessor()` command **automatically enables** the specified processor. Thus OpModes **do not initialize** with this, after Step 2 or 3:

Blocks

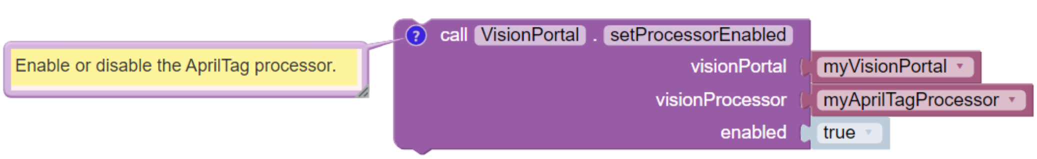


Fig. 45: VisionPortal Enabling/Disabling

Java

```java
// Enable or disable the AprilTag processor.
myVisionPortal.setProcessorEnabled(myAprilTagProcessor, true);
```

Again, use this only to **re-enable** the processor, after **disabling** (by setting to `false`).

Questions, comments and corrections to westsiderobotics@verizon.net

VisionPortal Previews

Introduction

Managing AprilTag and TFOD performance is greatly enhanced with visual feedback of the camera's view.

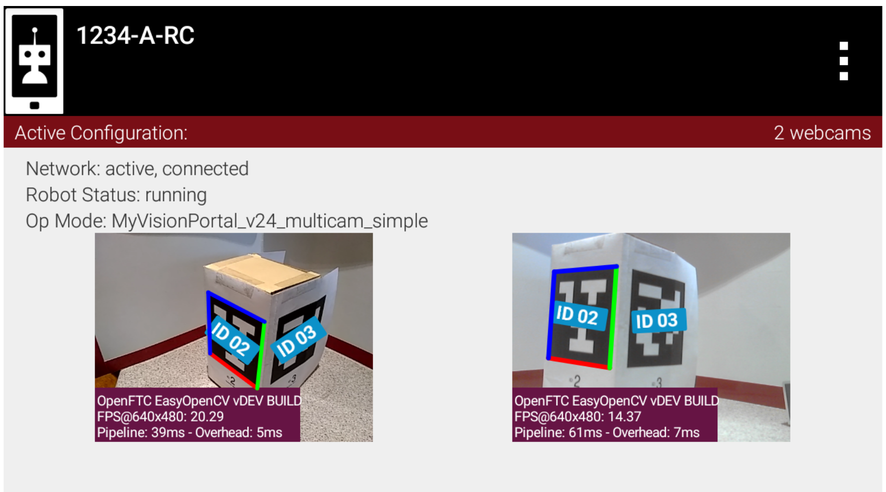


Fig. 46: LiveView demonstrating multiple camera support

The Driver Station and Robot Controller apps offer a camera preview on both devices:

- **LiveView** on Robot Controller (RC) device – RC phone or Control Hub (see below)
- **Camera Stream** on Driver Station (DS) device – DS phone or Driver Hub

LiveView refers only to the **Robot Controller** preview (example shown above). It's completely separate from **DS Camera Stream**, which still operates normally even if LiveView is stopped (manually or automatically).

Instructions for viewing DS Camera Stream are shown at *ftc-docs*.

Camera Stream uses its own frame collection process, which naturally still requires the camera/pipeline status to be `STREAMING`. Disabling the stream will prevent the DS preview. Camera status is covered at the **Managing CPU and Bandwidth** page, and the **VisionPort Camera Controls** page.

Side Note: For SDK 8.2, "LiveView" became the new universal name for the RC preview. There remain two instances of old names:

- `myVisionPortalBuilder.enableCameraMonitoring(true);`
- `VIEWPORT` appears in the preview status window, when stopped

LiveView on Control Hub

The Control Hub does generate an RC preview, despite not having a built-in screen. LiveView can be seen in two ways:

- Plug an HDMI monitor into the Control Hub's (full-size) HDMI port
- Use `scrcpy` (pronounced "screen copy"), available here:
  - https://github.com/Genymobile/scrcpy

Camera Controls

Images in LiveView and Camera Stream are both affected by Camera Controls, for webcam. Changing values of Exposure and Gain, for example, do affect the displayed image and the actual recognitions.

During Camera Stream, manual adjustments to Camera Controls cannot be made in real time (with visible feedback) since gamepads are disabled.

Thus teams wanting to optimize AprilTag or TFOD recognitions with Camera Controls should use `scrcpy` or an HDMI monitor. Doing this via Camera Stream ("back and forth") will be less effective and less efficient.

More information is available at the **VisionPortal Camera Controls** page, and at the *Webcam Control tutorial*.

Aspect Ratios in Previews

Here's a Control Hub's LiveView (via `scrcpy`) of TFOD recognitions:

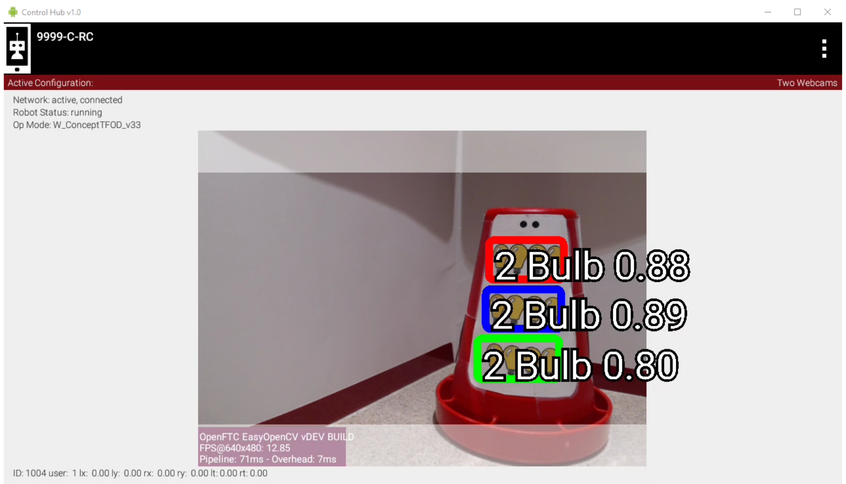


Fig. 47: LiveView demonstrating Grey Bands from Aspect Ratio mismatch

The **greyed bands** at top and bottom are from the **mismatch of aspect ratios**:

- 4:3 for camera (640x480)
- 16:9 for TFOD (per model training)

Both of these ratios are set as defaults, hidden from the user in some Sample OpModes. Only the non-greyed region is eligible for TFOD recognitions.

Note that the TFOD annotations (text) extend beyond the image.

BIG Previews

A new feature of SDK 8.2, the Driver Station's **Camera Stream** preview can appear regular-size or **BIG**.

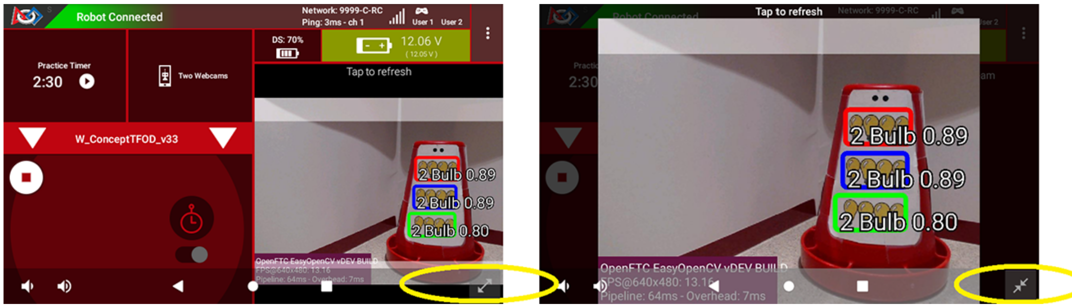


Fig. 48: Camera Stream preview enlargement buttons

Circled in yellow are the user buttons to **go BIG** or return to the **default** screen.

Note the annotations have shifted to fit in the image.

Orientation Notes

With SDK 8.2, the default image orientation is **SENSOR_NATIVE**.

This Java **enum** SENSOR_NATIVE means that the processing pipeline is getting the image in the native orientation of the camera sensor. Namely, no rotation is performed. Note that (former) enum UPRIGHT for a webcam is the same as SENSOR_NATIVE, while for a phone camera, (former) enum SIDEWAYS_LEFT is the same as SENSOR_NATIVE.

SENSOR_NATIVE is ideal because the overhead of rotating the image stream is rather high.

Note that viewing the video stream from the same orientation as the statistics text box will show you the orientation of the stream passed to the AprilTag and/or TFOD processors.

Also note that for RC phone cameras, the LiveView preview is rotated (independent of rotation enum) such that the preview is the way you "expect" as if you were to open the camera app on the phone. That rotation happens during the GPU-accelerated rendering of the bitmap and is significantly easier on resources.

Questions, comments and corrections to westsiderobotics@verizon.net

AprilTag ID Codes

After the AprilTag Processor and VisionPortal have been **initialized**, your OpMode can begin tag detection.

Let's start with the simple task of retrieving the **ID code** of a detected AprilTag. For tag family 36h11, the numeric ID code ranges from 0 to 586. The FTC SDK can provide over 30 fields per detected AprilTag, if that tag's size was provided (thus eligible for pose estimation). Otherwise only tag ID code is available.

Blocks

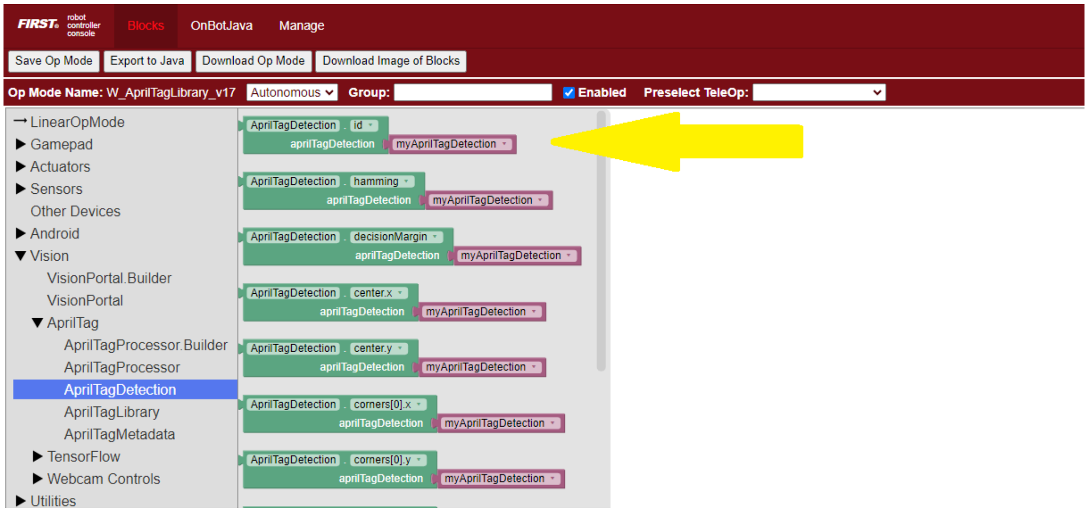


Fig. 49: Highlighting blocks for getting AprilTag ID Code

Java

Example of retrieving AprilTag ID

```
AprilTagDetection myAprilTagDetection;
int myAprilTagIdCode = myAprilTagDetection.id;
```

Since the camera might see multiple AprilTags at once, retrieving any field(s) is usually done with a **`for() loop`**. The loop can process each detection, one at a time:

Blocks

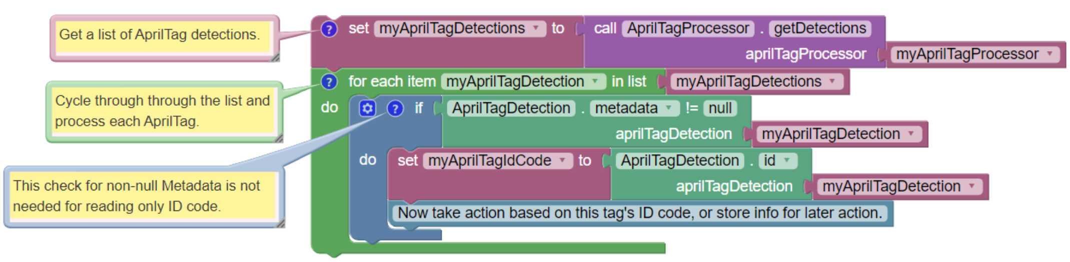


Fig. 50: All AprilTag Detections being read in a FOR Loop

This code snippet assumes `myAprilTagProcessor` and VisionPortal have been initialized, as described at previous pages **Processor Initialization** and **VisionPortal Initialization**.

Java

Example reading all detected AprilTags in a FOR Loop

```
AprilTagProcessor myAprilTagProcessor;
List<AprilTagDetection> myAprilTagDetections;  // list of all detections
AprilTagDetection myAprilTagDetection;         // current detection in for() loop
int myAprilTagIdCode;                           // ID code of current detection, in for() loop

// Get a list of AprilTag detections.
myAprilTagDetections = myAprilTagProcessor.getDetections();

// Cycle through through the list and process each AprilTag.
for (myAprilTagDetection : myAprilTagDetections) {

     if (myAprilTagDetection.metadata != null) {  // This check for non-null Metadata is not needed for reading only ID code.
          myAprilTagIdCode = myAprilTagDetection.id;

          // Now take action based on this tag's ID code, or store info for later action.

     }
}
```

This code snippet assumes `myAprilTagProcessor` and VisionPortal have been initialized, as described at previous pages **Processor Initialization** and **VisionPortal Initialization**.

The OpMode should take the desired action for each AprilTag **inside** the `for() loop`, or store information for later action. In the above example, the variable `myAprilTagIdCode` receives the different values of each detection, ending with only the **last tag's value**.

By default, the FTC SDK recognizes the ID code of **any** 36h11 AprilTag, even if the OpMode did not place that tag in the AprilTag Library. Some tags are placed in the Library automatically by the SDK: for example, ID codes 583-586 used by Sample OpModes.

An OpMode can also place other tags in a Library, to supplement or overwrite default tags. This is covered further at the **Library** page.

Questions, comments and corrections to westsiderobotics@verizon.net

AprilTag Metadata

Introduction

A Library tag stores **Metadata**, a collection of at least 4 fields (of these Blocks/Java types):

- ID code (number/`int`)
- tag name (text/`String`)
- tag size (number/`double`)
- unit for tag size and estimated position (`DistanceUnit INCH, MM, CM, METER`)

Two optional Metadata fields are described at the **Advanced Use** page.

The full use of Metadata Blocks and Java methods is covered at the **Library** page. For now it's enough to know the 4 basic elements of Metadata.

Tag Contents

The SDK 8.2 Sample OpModes use AprilTags with these Metadata values:

- `583, Nemo, 4, DistanceUnit.INCH`
- `584, Jonah, 4, DistanceUnit.INCH`
- `585, Cousteau, 6, DistanceUnit.INCH`
- `586, Ariel, 6, DistanceUnit.INCH`

These four are available with the `getSampleTagLibrary()` Block or Java method.

After Kickoff in September 2023, the CENTERSTAGE game tags will be available with `getCenterStageTagLibrary()`. That Library currently contains 3 placeholder tags: MEOW (ID 0), WOOF (ID 1) and OINK (ID 2).

Before and after Kickoff, a call to `getCurrentGameTagLibrary()` will provide **both sets** of tags.

These three Libraries are discussed further at the **Library** page.

Tag Names

A tag name, whether default or custom, can be retrieved as follows:

Blocks

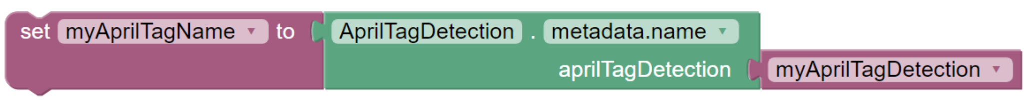


Fig. 51: Example of Reading AprilTag Names

Java

Example of retrieving AprilTag Name

```
AprilTagDetection myAprilTagDetection;
String myAprilTagName;
myAprilTagName = myAprilTagDetection.metadata.name;
```

As with tag ID code, the tag name is usually retrieved inside a `for()` loop, for immediate processing or stored for later use. See the **Initialization** page for sample `for()` loop code.

Unlike tag ID code, a detected AprilTag might have **no tag name** – if it was not placed into the Library by default or with the custom Builder pattern.

To avoid logic errors, an OpMode can check the Metadata for a **null** condition before attempting to process a tag name. This is illustrated in these Sample OpModes:

- Blocks: `ConceptAprilTag`
- Java: `ConceptAprilTag.java`

Questions, comments and corrections to westsiderobotics@verizon.net

AprilTag Reference Frame

Introduction

Before discussing AprilTag **pose** (at the next page), the **FTC axes** or reference frame must be described. Pose data is based on the **camera's point of view**, and requires a **flat AprilTag**.

Here are the axis designations in the new SDK:

- Y axis points **straight outward** from the camera lens center
- X axis points **to the right** (looking outward), perpendicular to the Y axis
- Z axis points **upward**, perpendicular to Y and X

Note: Note 1: If the camera is upright and pointing "forward" on the robot, these axes are consistent with the Robot Coordinate System used for *IMU navigation*

Note: Note 2: these axes are different than the official AprilTag definitions, even from the camera's frame of reference.

Translation Offset

If the AprilTag is detected within the camera's field of view, the tag's center is described in that reference frame as (X, Y, Z) position, also called displacement or translation.

This is illustrated with a camera preview image, called LiveView, from a Robot Controller device (Control Hub or RC phone).

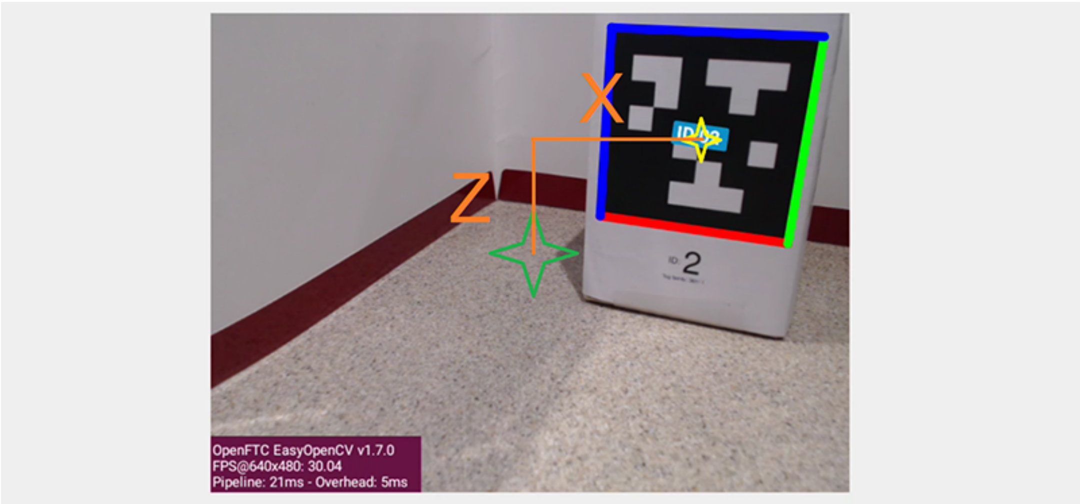


Fig. 52: Image depicting LiveView offsets

Imagine a laser beam pointing straight outward from the center of the camera lens. Its 3-dimensional path appears (to the camera) as a single point, located at the **green star**. You can see that the center of the AprilTag (**yellow star**) is offset from that "laser beam".

That **translation offset** can break down into three traditional components (X, Y and Z distances), along axes at 90 degrees to each other:

- X distance (horizontal orange line) is from the center, to the right
- Y distance (not shown) is from the lens center, outwards

- Z distance (vertical orange line) is from the center, upwards

The SDK provides these distances **in the real world**, not just reporting how many pixels on the screen. The **distance units** are specified in each tag's Metadata (default is **inches**).

Think of the Y distance as the length of the "laser beam", when the tip of the horizontal orange line touches the yellow star **on the tag**.

If the tag is exactly in front of the camera, X and Z are zero, while Y represents the positive distance to the tag.

Rotation Offset

You can also see that the AprilTag's flat face is not parallel to the plane of the camera. That **rotation offset** can break down into three angles about the X, Y and Z axes.

Any off-axis pointing or tilting of the AprilTag is reported by the SDK as rotation about axes X, Y or Z. Here are some examples:

- If that tag is parallel to the camera but tilted, say, clockwise, this is expressed as positive angular rotation (Roll) about the Y axis.
- If a tag appears to the left side of the camera's view, this has an X-axis displacement or translation. It's a negative translation, since X points to the right.
- If that left-displaced tag is also angled, say, to face the camera, this is expressed as angular rotation about the vertical Z axis. It's a positive Yaw angle, according to the **right-hand rule**: with the thumb pointing along the positive axis, the fingers curl in the direction of positive rotation.
- If a detected tag is angled or pointing, say, slightly upward to the ceiling, this is expressed as rotation about X. Use the right-hand rule to confirm this would be a negative Pitch angle, since X points to the right. This example assumes the camera is pointing parallel to the ground/mat.

Related Info

More discussion of the AprilTag reference frame is available here:

Understanding AprilTag Detection Values

This section described the SDK's default AprilTag reference frame. Teams are welcome to make other calculations, such as the pose of the camera (or robot) relative to the AprilTag, or **relative to the game field**. Such advanced efforts can be useful and a good learning exercise, beyond the scope of the SDK and this guide.

Questions, comments and corrections to westsiderobotics@verizon.net

AprilTag Camera Calibration

To provide good pose estimates, **each RC phone camera or webcam model** requires calibration data, for **each specific resolution**.

> *"Without a camera calibration, the best you could achieve is being able to turn towards the target. Range information would be incorrect." – FIRST Tech Challenge navigation expert @gearsincorg*

The *FIRST* Tech Challenge SDK contains such data for a limited number of webcams and resolutions. Teams can generate their own data, called **lens intrinsics**.

Here's one possible procedure, of several free choices available publicly.

Utility OpMode

First, create an OpMode from the Java Sample `UtilityCameraFrameCapture.java`. Android Studio teams can find this utility program in the External Samples folder.

FTC Blocks teams can duplicate this OpMode, requiring a custom myBlock only for the method `saveNextFrameRaw()`. At some future time, this Java method may become available as a regular Block, avoiding the need for a myBlock. Learn more about myBlocks here:

- *MyBlocks Tutorial*.

This Utility OpMode helps calibrate a webcam or RC phone camera, needed for AprilTag pose estimation. It captures a camera frame (image) and stores it on the Robot Controller (Control Hub or RC phone), with each press of the gamepad button X (or Square).

To illustrate, the OpMode stores the first two captured images as:

- `VisionPortal-CameraFrameCapture-000000.png`
- `VisionPortal-CameraFrameCapture-000001.png`

This is done for each run of the OpMode. Teams should move each set of frames to its own folder (on a computer), to avoid overwriting the previous run's results.

Mac OSX users may need special software for Android file transfer.

Next, read and follow the calibration instructions posted at ftc-docs. Other calibration programs are widely available online.

Existing Warnings

Running `ConceptDoubleVision` (or any AprilTag Sample OpModes) using a built-in RC phone camera, gives the following error message on both devices:

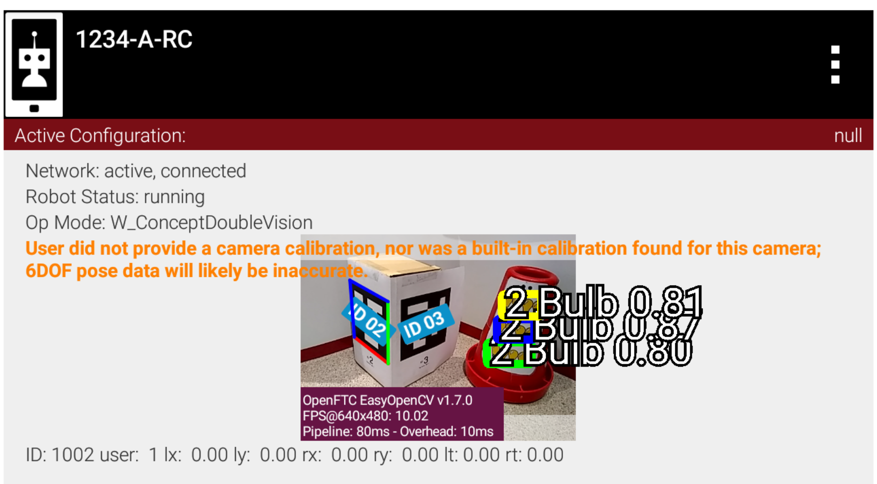


Fig. 53: Warning of no camera calibration provided

The SDK gives a different warning that covers a **special case**, where the OpMode uses:

- a camera model for which the SDK **does have** lens intrinsics, and
- a user-specified resolution for which

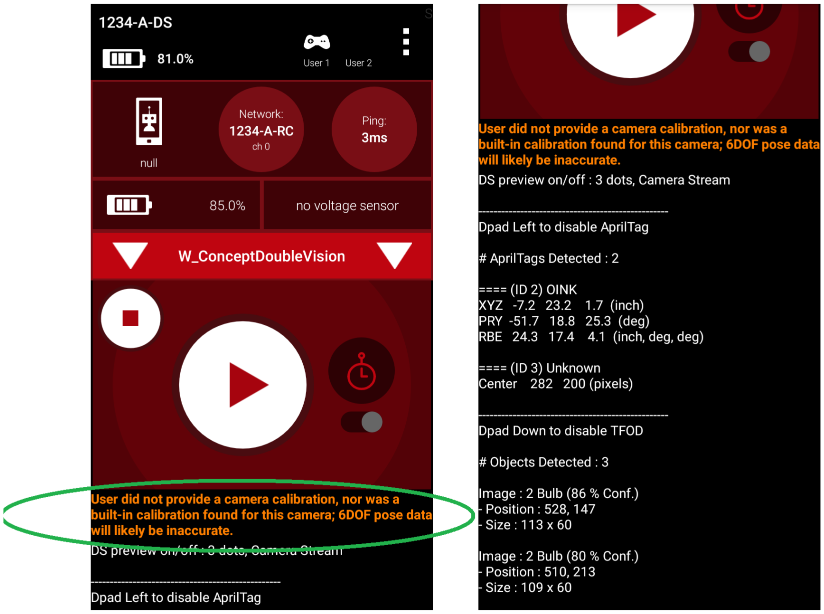


Fig. 54: Right-hand image shows that the warning still allows detections.

- (a) the SDK **does not have** lens intrinsics, and
- (b) the **aspect ratio** matches that of lens intrinsics that the SDK **does have** (for that camera model).

In such a case, the SDK **scales** the results in an attempt to estimate AprilTag pose.

For example, changing the Logitech C270 resolution from 640x480 to 800x600 (also 4:3 aspect ratio), gives this warning on the RC preview and the DS screen:

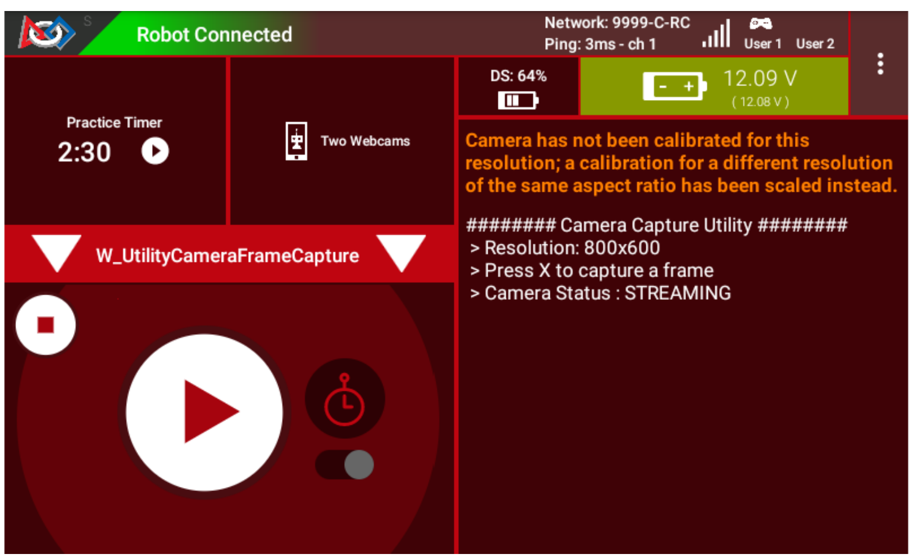


Fig. 55: Warning about no calibration at this resolution

The above warning advises the user of this situation, with the opportunity to accept/adjust the scaled estimate or provide actual calibration values.

This warning does not affect the function of capturing and storing camera frames.

SDK Calibration Data

The Logitech C270 webcam offers 18 resolutions, each wanting calibration. The Logitech C920 offers 19 resolutions.

For the "standard" Logitech C270 (from the *FIRST* Storefront), the SDK 8.2 currently has a set of lens intrinsics for **one resolution**, 640x480.

Currently the SDK has calibration data for 10 resolutions spread among 4 webcams:

- Logitech HD Webcam C270, 640x480
- Logitech HD Pro Webcam C920, 640x480, 800x600, 640x360, 1920x1080, 800x448, 864x480
- Logitech HD Webcam C310, 640x480, 640x360
- Microsoft Lifecam HD 3000 v1/v2, 640x480

These are found in the SDK file `builtinwebcamcalibrations.xml`. In Android Studio, navigate to the subfolders `RobotCore`, `res`, `xml`.

Android RC phone cameras also need calibration data for good pose estimates. The SDK provides no lens intrinsics for these cameras.

Questions, comments and corrections to westsiderobotics@verizon.net

AprilTag Pose

The SDK can evaluate a **flat AprilTag** (not curved) to estimate **pose**, the combination of:

- relative position **from the camera lens center to the AprilTag center**, and
- orientation of the AprilTag **in the camera's reference frame**

As described at the previous page **FTC Reference Frame**, position is expressed as (X, Y, Z). Orientation is expressed as rotation about (X, Y, Z), called Pitch, Roll and Yaw respectively.

The tag must be in the Library, which ensures that tag size (with units) is defined. Estimating pose requires knowing the tag size.

As demonstrated in the Sample OpModes, here are ways to retrieve the estimated pose values.

Blocks

Use each of these green Blocks to pass a Pose value to a Telemetry Block, or to a Variable:

Java

Use these *ftcPose* fields for Telemetry, or assign to a Variable:

```
AprilTagDetection myAprilTagDetection;
double myTagPoseX = myAprilTagDetection.ftcPose.x;
double myTagPoseY = myAprilTagDetection.ftcPose.y;
double myTagPoseZ = myAprilTagDetection.ftcPose.z;
double myTagPosePitch = myAprilTagDetection.ftcPose.pitch;
double myTagPoseRoll = myAprilTagDetection.ftcPose.roll;
double myTagPoseYaw = myAprilTagDetection.ftcPose.yaw;
```

The SDK terms for Pitch, Roll and Yaw are **not the same** *as the native AprilTag terms, due to the FTC reference frame.*

Teams may find it helpful to use a **calculated extension** of the basic pose, with these terms:

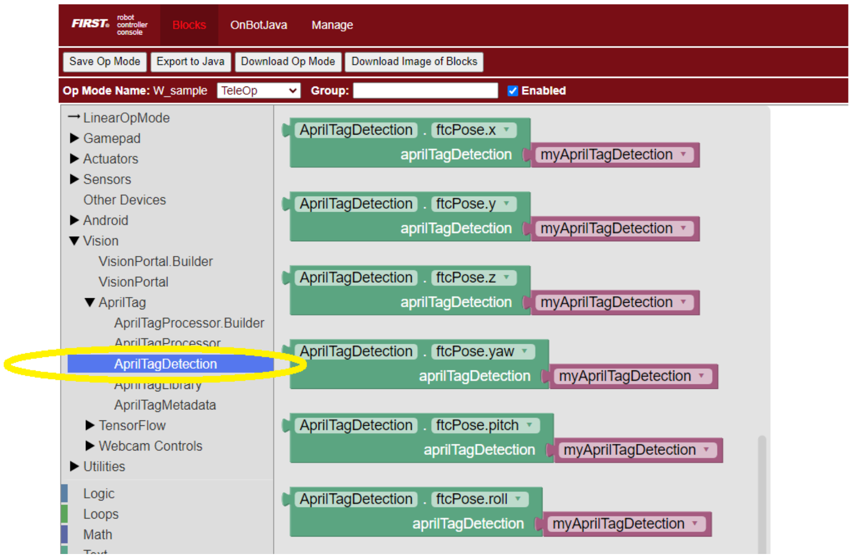


Fig. 56: AprilTag Pose Blocks

- **Range**, direct (point-to-point) distance to the tag center
- **Bearing**, the angle the camera must turn (left/right) to point directly at the tag center
- **Elevation**, the angle the camera must tilt (up/down) to point directly at the tag center

Blocks

Here each green Block assigns its value to a Variable:

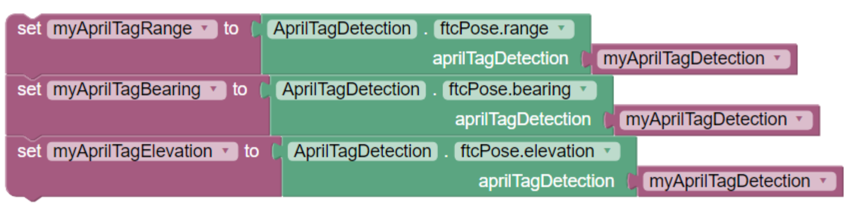


Fig. 57: AprilTag Range, Bearing, Elevation Blocks

Java

Use these *ftcPose* fields for Telemetry, or assign to a Variable:

```
AprilTagDetection myAprilTagDetection;
double myTagPoseRange = myAprilTagDetection.ftcPose.range;
double myTagPoseBearing = myAprilTagDetection.ftcPose.bearing;
double myTagPoseElevation = myAprilTagDetection.ftcPose.elevation;
```

Here, the terms do agree with the SDK method names, because they are calculated within the SDK from the native AprilTag pose values shown above (XYZ distances and PRY rotations).

As with tag ID code, pose data is usually retrieved inside a `for() loop`, for immediate processing or stored for later use. See the **Initialization** page for sample `for() loop` code.

Unlike tag ID code, a detected AprilTag might provide **no pose data** – if it was not placed into the Library by default or with the custom Builder pattern. Namely, the tag might lack Metadata including **tag size**, required for pose estimation.

To avoid logic errors, an OpMode can check the Metadata for a **null** condition before attempting to process pose data. This is illustrated in these Sample OpModes:

- Blocks: *ConceptAprilTag*
- Java: *ConceptAprilTag.java*

More discussion of AprilTag pose data is available here:

Understanding AprilTag Detection Values

Questions, comments and corrections to westsiderobotics@verizon.net

AprilTag Library

For a *FIRST* Tech Challenge match, your OpMode has a known set of AprilTags to detect. They are preloaded by default or specified by you, with or without custom tags.

These tags form an **AprilTag Library**. Each Library tag has a set of 4 to 6 properties, described at the **Metadata** page.

This page shows many ways to create an AprilTag Library. The **Initialization** page explained this is the optional **Step 1** of preparing to use AprilTags in an OpMode.

Try these examples in order, to master the use of AprilTag Libraries.

The Easy Way

Let's start with… no Library! If your OpMode will use only the current season defaults, no Library action is needed.

Simply create the AprilTagProcessor as follows:

Blocks

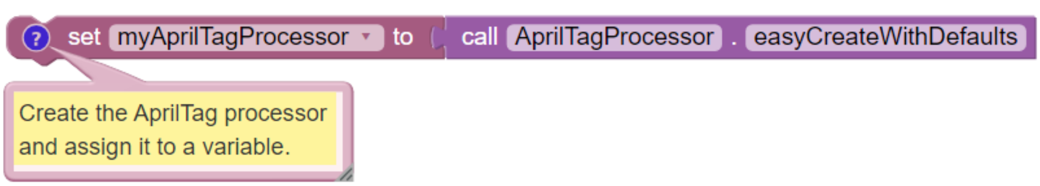


Fig. 58: Simple AprilTag Processor

Java

```
AprilTagProcessor myAprilTagProcessor;

// Create the AprilTag processor and assign it to a variable.
myAprilTagProcessor = AprilTagProcessor.easyCreateWithDefaults();
```

Specifying a Library is needed for creating an AprilTag Processor. Even this "Easy Way" does specify the default Library, behind the scenes.

Default Libraries

The SDK uses two core Libraries of predefined AprilTags:

- tags used only in Sample OpModes
- tags used only in the Robot Game (competition)

The first Library, called `SampleTagLibrary`, is available now with SDK 8.2. Its basic Metadata values are:

- `583, Nemo, 4, DistanceUnit.INCH`
- `584, Jonah, 4, DistanceUnit.INCH`
- `585, Cousteau, 6, DistanceUnit.INCH`
- `586, Ariel, 6, DistanceUnit.INCH`

The second Library, called `CenterStageTagLibrary`, is planned for future competition only. It's available now in SDK 8.2, but currently holding three "placeholder" tags:

- `0, MEOW, 0.166, DistanceUnit.METER`
- `1, WOOF, 0.166, DistanceUnit.METER`
- `2, OINK, 0.166, DistanceUnit.METER`

After Kickoff in September 2023, these will be replaced (in SDK 9.0) by the **real tags** for CENTERSTAGE.

For convenience, a third Library contains **both** of these core Libraries, and nothing else. This is the default, called `CurrentGameTagLibrary`.

AprilTag Processor

Specifying **any aspect** of a Processor is done with a **Processor Builder**, requiring at least 2 commands:

- create the Builder, using the Java keyword new
- after specifying/adding features, finalize with a call to the `.build()` method

In between these actions, your OpMode will add one of the three Libraries directly to the Processor Builder. It's easiest to use the default `CurrentGameTagLibrary`, containing all of the predefined tags.

Blocks

First create this expression, drawing the first component from the `AprilTagProcessor.Builder` toolbox (or palette), and drawing the second component from the `AprilTagLibrary` toolbox.

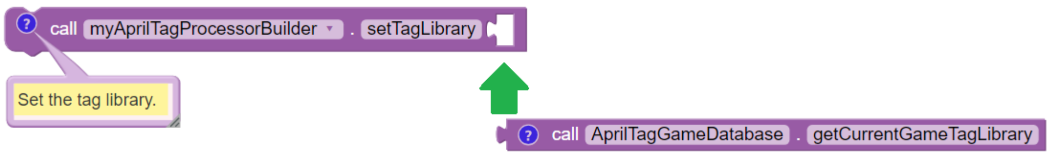


Fig. 59: Setting Current Game Tag Library

Around this (before and after), place one Block to **create** the Processor Builder, and another Block to **finalize** the process with `.build()`.

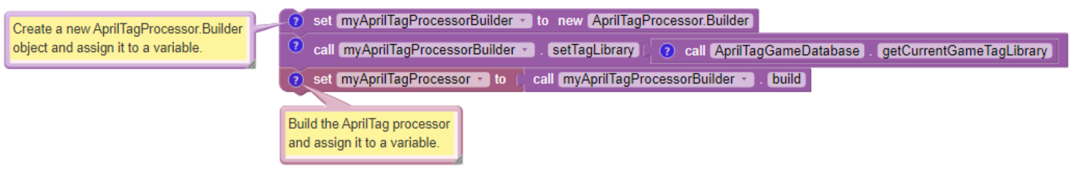


Fig. 60: Completing Builder

These are the first and last Blocks in the `AprilTagProcessor.Builder` toolbox. The remaining Blocks are used to set optional features of the Processor. Here we are setting only the Library.

Java

```
AprilTagProcessor.Builder myAprilTagProcessorBuilder;
AprilTagProcessor myAprilTagProcessor;

// Create a new AprilTagProcessor.Builder object and assign it to a variable.
myAprilTagProcessorBuilder = new AprilTagProcessor.Builder();

// Set the tag library.
// Get the AprilTagLibrary for the current season.
myAprilTagProcessorBuilder.setTagLibrary(AprilTagGameDatabase.getCurrentGameTagLibrary());

// Build the AprilTag processor and assign it to a variable.
myAprilTagProcessor = myAprilTagProcessorBuilder.build();
```

Library Variable

As an alternate, you could first store the Library in your own Variable name. Then specify that name for the AprilTag Processor. Here we use `myAprilTagLibrary` (any other name is fine).

Blocks

First create this expression, drawing the first component from the `AprilTagLibrary` toolbox, and drawing the second component from the `AprilTagProcessor.Builder` toolbox.

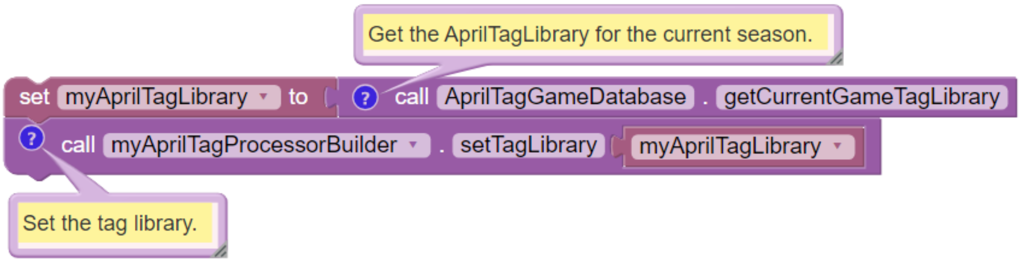


Fig. 61: Set the Tag Library

As before, **around this** (before and after), place one Block to **create** the Processor Builder, and another Block to **finalize** the process with `.build()`.

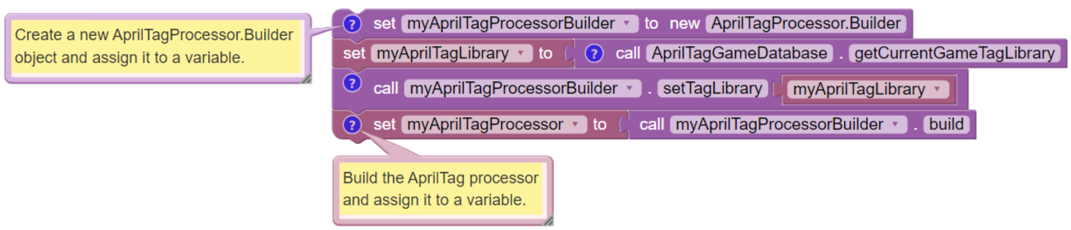


Fig. 62: Build the AprilTag Processor

Java

```
AprilTagProcessor.Builder myAprilTagProcessorBuilder;
AprilTagProcessor myAprilTagProcessor;
AprilTagLibrary myAprilTagLibrary;

// Create a new AprilTagProcessor.Builder object and assign it to a variable.
myAprilTagProcessorBuilder = new AprilTagProcessor.Builder();

// Get the AprilTagLibrary for the current season.
myAprilTagLibrary = AprilTagGameDatabase.getCurrentGameTagLibrary();

// Set the tag library.
myAprilTagProcessorBuilder.setTagLibrary(myAprilTagLibrary);

// Build the AprilTag processor and assign it to a variable.
myAprilTagProcessor = myAprilTagProcessorBuilder.build();
```

Library Builder, with Defaults

Next we try the Builder pattern, to create a Library containing only predefined AprilTags. It's not needed (nothing new to Build!), but it's an easy introduction.

Blocks

- Create a Library Builder, not the same as a Processor Builder.
- Then use the `addTags` Block – note the plural "tags", not "tag".
- Finalize the process with the `.build` command.

The built Library is assigned or saved to your Variable, here called `myAprilTagLibrary`.

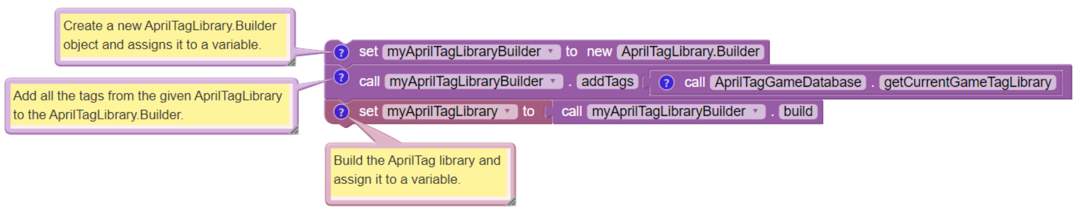


Fig. 63: Build the Tag Library

Next comes the familiar steps:

- Create a Processor Builder.
- Then set, or add, the Library built and saved in the previous sequence.
- Finalize the process with the `.build` command.

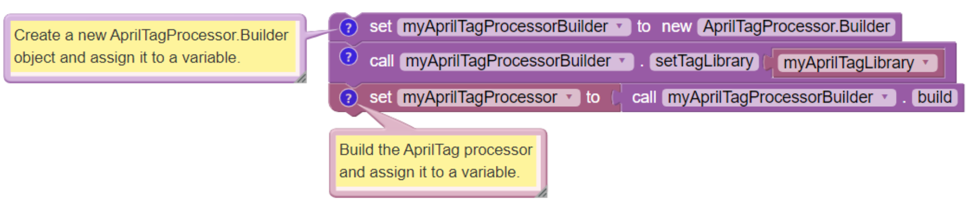


Fig. 64: Build the Tag Processor

The final result is the same as the previous examples, but now you see how to use a Library Builder.

Java

```
AprilTagLibrary.Builder myAprilTagLibraryBuilder;
AprilTagProcessor.Builder myAprilTagProcessorBuilder;
AprilTagLibrary myAprilTagLibrary;
AprilTagProcessor myAprilTagProcessor;

// Create a new AprilTagLibrary.Builder object and assigns it to a variable.
myAprilTagLibraryBuilder = new AprilTagLibrary.Builder();

// Add all the tags from the given AprilTagLibrary to the AprilTagLibrary.Builder.
// Get the AprilTagLibrary for the current season.
myAprilTagLibraryBuilder.addTags(AprilTagGameDatabase.getCurrentGameTagLibrary());
```

(continues on next page)

(continued from previous page)

```
// Build the AprilTag library and assign it to a variable.
myAprilTagLibrary = myAprilTagLibraryBuilder.build();

// Create a new AprilTagProcessor.Builder object and assign it to a variable.
myAprilTagProcessorBuilder = new AprilTagProcessor.Builder();

// Set the tag library.
myAprilTagProcessorBuilder.setTagLibrary(myAprilTagLibrary);

// Build the AprilTag processor and assign it to a variable.
myAprilTagProcessor = myAprilTagProcessorBuilder.build();
```

Custom Tag - Direct

Finally, we are ready to add custom tags to a Library.

Each tag needs Metadata. You can enter Metadata values directly to a new tag, as follows.

Blocks

The third Block adds the custom tag to the Library. All other Blocks are the same as the previous example.

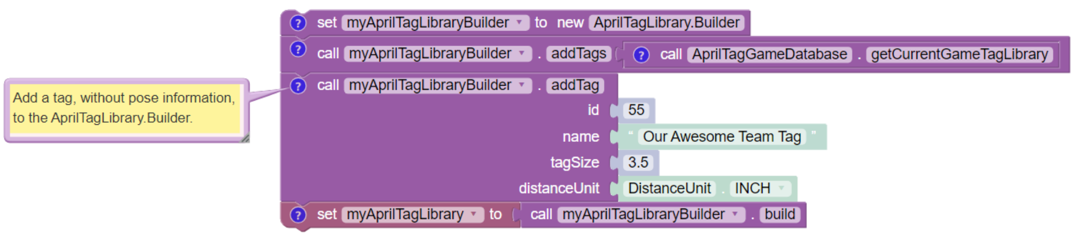


Fig. 65: Add custom tags to Tag Library

Java

The custom tag is added with **one new line** of code, otherwise the same as the previous example.

```
AprilTagLibrary.Builder myAprilTagLibraryBuilder;
AprilTagProcessor.Builder myAprilTagProcessorBuilder;
AprilTagLibrary myAprilTagLibrary;
AprilTagProcessor myAprilTagProcessor;

// Create a new AprilTagLibrary.Builder object and assigns it to a variable.
myAprilTagLibraryBuilder = new AprilTagLibrary.Builder();

// Add all the tags from the given AprilTagLibrary to the AprilTagLibrary.Builder.
// Get the AprilTagLibrary for the current season.
myAprilTagLibraryBuilder.addTags(AprilTagGameDatabase.getCurrentGameTagLibrary());

// Add a tag, without pose information, to the AprilTagLibrary.Builder.
myAprilTagLibraryBuilder.addTag(55, "Our Awesome Team Tag", 3.5, DistanceUnit.INCH);
```

(continues on next page)

(continued from previous page)

```
// Build the AprilTag library and assign it to a variable.
myAprilTagLibrary = myAprilTagLibraryBuilder.build();

// Create a new AprilTagProcessor.Builder object and assign it to a variable.
myAprilTagProcessorBuilder = new AprilTagProcessor.Builder();

// Set the tag library.
myAprilTagProcessorBuilder.setTagLibrary(myAprilTagLibrary);

// Build the AprilTag processor and assign it to a variable.
myAprilTagProcessor = myAprilTagProcessorBuilder.build();
```

Custom Tag - Variable

As an alternate, you can assign Metadata to a Variable, then use that Variable to create a new AprilTag.

Blocks

These two Blocks could replace the single new Block in the previous example.

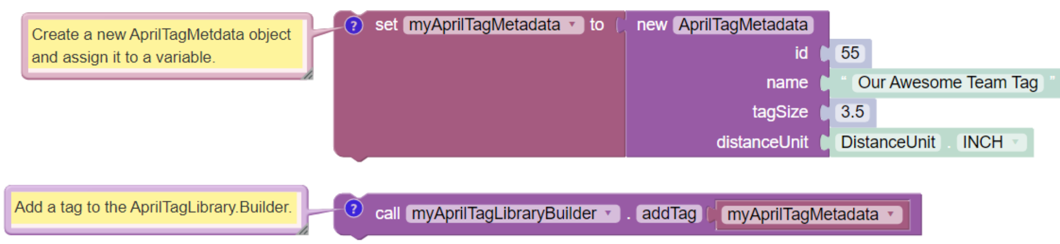


Fig. 66: Setting Metadata with Variable

These Blocks are separated, to illustrate that the Metadata Variable can be initialized/assigned anywhere before being added with the Library Builder. It doesn't have to appear immediately before use.

Java

The custom tag is added with **two lines** of code, replacing the **one new line** in the previous example.

```
AprilTagMetadata myAprilTagMetadata;
AprilTagLibrary.Builder myAprilTagLibraryBuilder;
AprilTagProcessor.Builder myAprilTagProcessorBuilder;
AprilTagLibrary myAprilTagLibrary;
AprilTagProcessor myAprilTagProcessor;

// Create a new AprilTagLibrary.Builder object and assigns it to a variable.
myAprilTagLibraryBuilder = new AprilTagLibrary.Builder();

// Add all the tags from the given AprilTagLibrary to the AprilTagLibrary.Builder.
// Get the AprilTagLibrary for the current season.
myAprilTagLibraryBuilder.addTags(AprilTagGameDatabase.getCurrentGameTagLibrary());

// Create a new AprilTagMetdata object and assign it to a variable.
myAprilTagMetadata = new AprilTagMetdata(55, "Our Awesome Team Tag", 3.5, DistanceUnit.INCH);
```

(continues on next page)

(continued from previous page)

```
// Add a tag to the AprilTagLibrary.Builder.
myAprilTagLibraryBuilder.addTag(myAprilTagMetadata);

// Build the AprilTag library and assign it to a variable.
myAprilTagLibrary = myAprilTagLibraryBuilder.build();

// Create a new AprilTagProcessor.Builder object and assign it to a variable.
myAprilTagProcessorBuilder = new AprilTagProcessor.Builder();

// Set the tag library.
myAprilTagProcessorBuilder.setTagLibrary(myAprilTagLibrary);

// Build the AprilTag processor and assign it to a variable.
myAprilTagProcessor = myAprilTagProcessorBuilder.build();
```

For Blocks or Java, multiple tags could be added with multiple (shorter!) Variable names, such as `myTag1`, `myTag2`, etc.

Overwriting

You might create a custom AprilTag with the **same ID code** as a tag already in the Library. This is **overwriting**, which you can allow or not allow.

If `setAllowOverwrite()` is set to `false` (the default) and overwrite is attempted, the OpMode will crash with a suitable error message.

If set to `true`, the custom tag will replace the existing tag.

Why might you do this? Suppose a tag size is not quite correct. You could enter a new tag with the same Metadata, but with a corrected tag size.

Or you might prefer to use your own tag names, or distance units.

Advanced users might want to specify the **location** of a predefined tag **on the game field**. This can be done with the optional Vector and Quaternion fields.

Questions, comments and corrections to westsiderobotics@verizon.net

VisionPortal CPU and Bandwidth

Introduction

Vision processing can consume significant **CPU resources** and USB communications **bandwidth**. Reaching such limits may affect previews, and cause an OpMode or Robot Controller to slow down, or freeze, or crash.

Teams can balance the benefits of higher resolution and speed (frames-per-second) against the risk of overloading CPU and bandwidth resources.

The 8.2 SDK provides numerous tools to manage this balance:

- disable and enable the RC preview (called LiveView) - “Level 1”
- disable and enable the AprilTag (or TFOD) processor - “Level 2”
- stop and resume the camera stream - “Level 3”
- close VisionPortal - “Level 4”

- monitor frames-per-second (FPS)
- select a compressed video streaming format
- select the camera resolution
- set decimation (down-sampling)
- select a pose solver algorithm
- get all or only fresh detections from the AprilTag Processor
- get all or only fresh recognitions from the TFOD Processor

The first four actions are informally rated for benefit and response:

- **LiveView** "Level 1": some reduction of resources used, resumes very quickly after stopping
- **Processor(s)** "Level 2": more reduction of resources used, resumes quickly after stopping
- **Camera Stream** "Level 3": high reduction of resources used, resumes less quickly after stopping
- **VisionPortal** "Level 4": maximum reduction of resources used, do not resume after stopping

Camera Status

Before discussing the tools to manage vision processing resources, we should review again the available **camera states**. This may help you monitor, evaluate and troubleshoot your optimization efforts.

Repeated from the **Camera Controls** page, these camera states are now available:

- OPENING_CAMERA_DEVICE - No vision processing is happening.
- CAMERA_DEVICE_READY - Camera is open. No processing is happening, including background processing from EOCV (i.e. pulling frames and performing color conversion). Ready to call `resumeStreaming()`.
- STARTING_STREAM - No processing is happening.
- STREAMING - Frames are available for processing (AprilTag and/or TFOD recognitions) and preview (LiveView RC preview and DS Camera Stream).
- STOPPING_STREAM - Processing may or may not be happening. This status is followed by `CAMERA_DEVICE_READY`.
- CLOSING_CAMERA_DEVICE - No processing is happening.
- CAMERA_DEVICE_CLOSED - Nothing is running, USB comms are closed. Once closed, don't open camera again during this OpMode.
- ERROR

These **enums** are listed in sequence, as if opening a camera (fresh build), then starting or resuming streaming, then stopping streaming, then closing the VisionPortal.

All of the above is completely separate from the AprilTag and/or TFOD processor status. Those can be enabled or disabled at any time, but naturally require `STREAMING` status to actually process camera images.

About Previews

As noted at the **Previews** page, LiveView refers only to the **Robot Controller** preview. It's completely separate from the Driver Station (DS) **Camera Stream**, which still operates normally even if LiveView is stopped (manually or automatically).

DS Camera Stream uses its own frame collection process, which naturally still requires the camera/pipeline status to be `STREAMING`.

Instructions for viewing DS Camera Stream are shown at *ftc-docs*.

DS Camera Stream can display only one camera's image, even under the new MultiPortal feature. Teams can create specialty OpModes to see one camera's image or the other camera's image, if needed for match set-up.

Side Note: For SDK 8.2, "LiveView" became the new universal name for the RC preview. There remain two instances of old names:

- `myVisionPortalBuilder.enableCameraMonitoring(true);`, discussed below
- `VIEWPORT` appears in the preview status window, when stopped

Pause LiveView - Direct

One way to conserve CPU resources ("Level 1") is **directly pausing** LiveView, while running an OpMode. The CPU continues processing camera images for AprilTag and/or TFOD recognitions, but does not actually generate an RC preview image (video).

Blocks

These are found in the `VisionPortal` toolbox, or palette, under the `Vision` category.

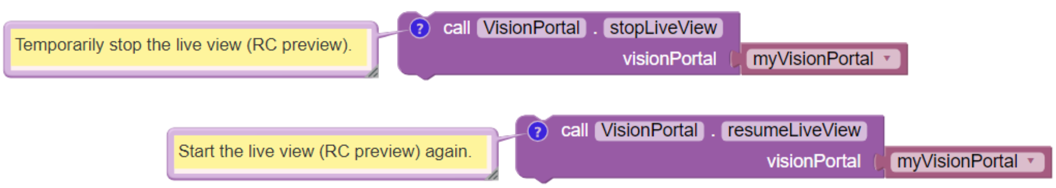


Fig. 67: Examples of Toggling LiveView

Java

```
// Temporarily stop the live view (RC preview).
myVisionPortal.stopLiveView();

// Start the live view (RC preview) again.
myVisionPortal.resumeLiveView();
```

Your OpMode will **not** need to work with camera status **enums** here, since these "stop" and "resume" actions happen quickly.

The above commands toggle only LiveView; the DS Camera Stream preview (touch to refresh) remains available.

Pause LiveView - Indirect

The SDK also offers an **indirect** control of LiveView, available in Blocks and Java:

```
builder.setAutoStopLiveView(true)
```

This setting causes LiveView to stop **automatically** if both processors (AprilTag and TFOD) are disabled. Being part of the Builder pattern, this feature cannot be directly toggled `true` and `false` during the OpMode.

This setting is triggered when **both** processors are disabled. When set to `false`, by default, the monitor continues showing the camera's view without annotations. If set to `true`, the monitor is Auto Paused, showing a solid orange screen if no processors are enabled. Thus the preview **can** effectively be toggled off and on, using this AutoPause feature.

When one or both processors are re-enabled, LiveView resumes. This setting affects only LiveView; the Driver Station Camera Stream preview remains available.

Disable LiveView

The SDK also contains a different Builder setting that allows (or disallows) LiveView **in general**, available in Blocks and Java:

```
builder.enableLiveView(true);
```

Sample OpModes set this Builder field to `true` by default.

This could be set to `false`, if the OpMode will not need the LiveView preview at all. Being part of the Builder pattern, this feature cannot be directly toggled `true` and `false` during the OpMode.

Toggle Processors

Another way to conserve CPU resources ("Level 2") is **disabling an AprilTag or TFOD Processor**, while running an OpMode.

Blocks

These are found in the `VisionPortal` toolbox, or palette, under the `Vision` category.

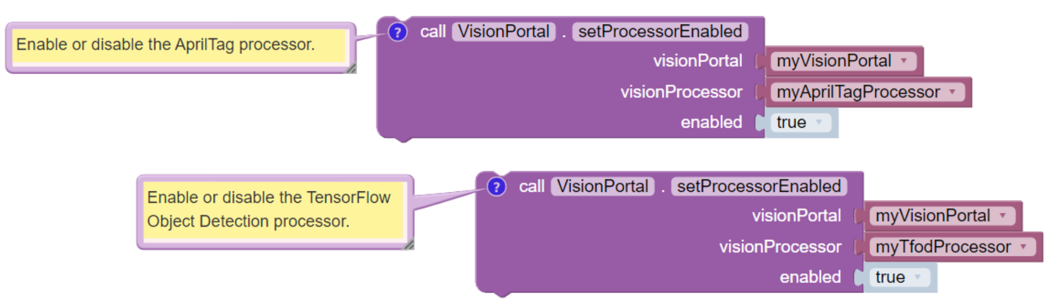


Fig. 68: Examples of Toggling Processors

Java

```
// Enable or disable the AprilTag processor.
myVisionPortal.setProcessorEnabled(myAprilTagProcessor, true);

// Enable or disable the TensorFlow Object Detection processor.
myVisionPortal.setProcessorEnabled(myTfodProcessor, true);
```

Disabling a Processor does not close LiveView, with its own controls described above. Any annotations will stop appearing in the preview.

Disabling and re-enabling processors is very fast, and saves CPU resources. But EOCV frame pulling and color conversion continue running in the background.

Toggle Camera Stream

A more active way to conserve CPU resources ("Level 3") is **stopping the camera stream**, while running an OpMode. Naturally this also achieves Levels 1 and 2: stopping LiveView and preventing operation of the AprilTag and TFOD Processors. DS Camera Stream provides no new snapshots.

Blocks

These are found in the `VisionPortal` toolbox, or palette, under the `Vision` category.

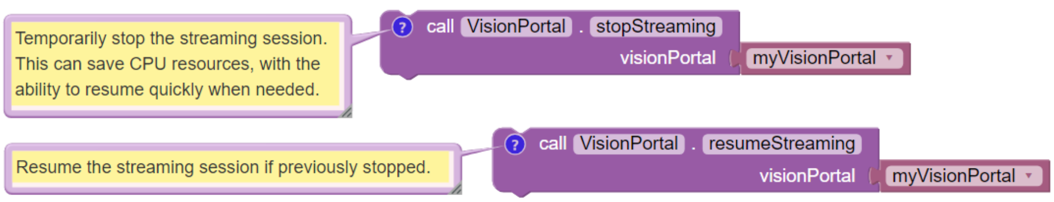


Fig. 69: Examples of Toggling Camera Stream

Java

```
// Temporarily stop the streaming session. This can save CPU
// resources, with the ability to resume quickly when needed.
myVisionPortal.stopStreaming();

// Resume the streaming session if previously stopped.
myVisionPortal.resumeStreaming();
```

Stopping (and later resuming) the stream is slightly risky, can take about 1 second, and stops all background processing. This is what happens when switching cameras, in the Sample OpModes called `SwitchableCameras`. One stream stops, and the other stream starts.

Close VisionPortal

Closing the portal with `close()` stops all background processing permanently ("Level 4"), and closes USB communication with the camera.

Blocks

These are found in the `VisionPortal` toolbox, or palette, under the `Vision` category.

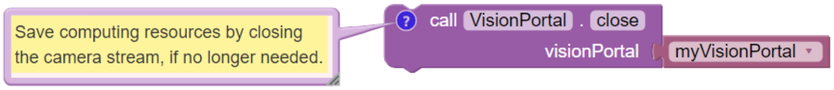


Fig. 70: Close VisionPortal Example

Java

```
// Save computing resources by closing VisionPortal at any time, if no
// longer needed.
myVisionPortal.close();
```

The `close()` process is a "teardown" of all camera processing. It is not recommended to "re-open" the camera within the same OpMode, by building another VisionPortal. This is risky and might take several seconds.

Accordingly, the SDK offers no corresponding `reopen()` or `resume()` method.

The `close()` process happens automatically at the end of any OpMode.

Calling `stopStreaming()` before calling `close()` is allowed (for clarity), but not required, since `close()` internally calls `stopStreaming()` if applicable.

Rapid Toggling

Your OpMode (or manual testing) should avoid or handle rapid stacking of the "on" and "off" actions described above.

It's legal to call `resumeStreaming()` while the status is `STOPPING_STREAM`. But the program will be **blocked** until the stopping operation is done.

Blocking means the latest function doesn't return immediately. So the code is temporarily "stuck" there, as if executing a `sleep()` command.

The same applies if calling `stopStreaming()` while the status is `STARTING_STREAM`. It's allowed, but your code may have to wait.

To avoid blocking, it's best to check the relevant **status enum** to make sure the previous operation is complete. This can be done with an empty `while()` loop, in a linear OpMode.

CPU Management Choices

So far, there are **10 possible configurations** to evaluate CPU performance, using only the vision process controls discussed above:

- VisionPortal closed
- VisionPortal open, Streaming off

Then 4 with Streaming on, Preview off:

- only AprilTag processor enabled
- only TFOD processor enabled
- both enabled
- both disabled

Then 4 with Streaming on, Preview on:

- only AprilTag processor enabled
- only TFOD processor enabled
- both enabled
- both disabled

This gives Teams ample opportunity to evaluate and manage CPU performance and USB Bandwidth. Many other tools remain:

- monitor frames-per-second (FPS)
- select a compressed video streaming format
- select the camera resolution
- set decimation (down-sampling)
- select a pose solver algorithm
- get all or only fresh detections from the AprilTag Processor
- get all or only fresh recognitions from the TFOD Processor

Frame Rate

The VisionPortal **automatically optimizes** for maximum frame rate, the number of processed frames per second (FPS). Presuming this optimization is based on **CPU resources**, measuring effects on **frame rate** could indirectly reflect CPU resource status/consumption/capacity.

Frame rate is reported visually in the LiveView status window. It's also available for your OpMode to track, record and evaluate, in Blocks and Java:

```
float myFPS = myVisionPortal.getFps();
```

Teams can collect FPS data to illustrate the general effects of, for example, (a) resolution and (b) processors running, on CPU performance. Results will depend on many team-specific factors such as webcams, codebase (other processing), vision targets (number, type, distance), etc.

Learn more about such studies at this Datalogging tutorial.

Dual Webcams

Before discussing Streaming Formats, we should mention that **USB Bandwidth** can be a concern for **dual webcams**.

Note: Internal phone cameras have an independent high-speed interconnect (not USB), unaffected by an added USB webcam.

The two webcams do *not* need to use the same format or resolution.

For dual webcams **plugged directly into the Control Hub**, the USB 2.0 and USB 3.0 ports are on different buses. This reduces the concern about bandwidth capacity, although higher resolution can cause the auto-optimized frame rate to reduce.

Using the Control Hub's two USB ports, the choice of stream format has little impact. But the USB 2.0 bus also carries the Control Hub's **WiFi radio**; adding a webcam may affect its reliability.

On the other hand, both webcams on an **external USB Hub** (plugged into the CH 3.0 port) can reach **bandwidth limits**, causing preview failures and OpMode crashes. This can be managed by factors discussed already, and by the choice of **streaming format**.

Streaming Formats

Under the legacy **YUY2 format**, one webcam or the other (on a shared hub) may stop streaming above roughly 640x360 resolution. This is **below the default** resolution of 640x480.

Bandwidth problems are often indicated by **no detections**, and a blue screen in LiveView. A team using default resolutions may quickly conclude (incorrectly) that dual webcams **does not work**.

The SDK now offers a compressed **MJPEG format**. This can significantly reduce USB bandwidth issues, but must be evaluated also for speed and quality of recognitions.

Under the MJPEG format, resolutions under roughly 432x240 may degrade the image to prevent AprilTag detection on at least 1 webcam, while higher resolutions may occasionally stop the RC app or crash the Control Hub.

For both formats, higher resolution can reduce frame rate.

These factors offer much opportunity for experimentation and Datalogging, to help optimize your VisionPortal performance.

Camera Resolution

Some teams believe "higher resolution is better", when purchasing webcams and specifying resolution for AprilTag and TFOD use.

As indicated in the previous sections here, it's more useful to consider a "suitable resolution" that satisfies multiple goals and challenges:

- quick and reliable AprilTag detections
- quick and reliable TFOD recognitions, including object tracking
- accurate AprilTag pose estimates
- smooth, accurate navigation while driving (higher FPS)
- avoid CPU overload
- avoid USB bandwidth limits
- resolution (or aspect ratio) for which calibration values exist
- accommodates lighting conditions and any Camera Controls applied

You might end up preferring the **lowest resolution** that meets your needs.

It's easy to find out which resolutions are supported by your camera. Just try to run any VisionPortal OpMode with an **incorrect (fake) resolution**; the error message will tell you the supported resolutions. Write these down for future reference.

Other Tools

This topic continues at the **AprilTag Advanced Use** page, to discuss advanced tools for managing CPU usage. It includes a Test OpMode in Blocks and Java.

For now, these are left for interested users to research and investigate:

- set decimation (down-sampling)
- select a pose solver algorithm
- get all or only fresh detections from the AprilTag Processor
- get all or only fresh recognitions from the TFOD Processor

All of the above features are easily found in the **FTC Blocks** toolboxes, or palettes, under Vision category.

Java users should review the VisionPortal interface at the SDK Javadocs site. Click **FRAMES** for easy navigation.

Questions, comments and corrections to westsiderobotics@verizon.net

VisionPortal Camera Controls

Clearer camera images can improve AprilTag (and TFOD) vision processing. The SDK offers powerful **webcam controls** (Exposure, Gain, Focus, and more), now available in Blocks! These controls can be applied under various lighting conditions.

The SDK documentation already provides a *Camera Controls tutorial*. You are encouraged to learn more there.

Note that Exposure and Gain are adjusted together. The new SDK offers Java Sample OpMode `ConceptAprilTagOptimizeExposure.java`, which can be constructed also in FTC Blocks.

Webcam States

Camera Controls cannot be used until the webcam has reached the state `CAMERA_DEVICE_READY`.

Under the new FTC VisionPortal these camera states are now available:

- OPENING_CAMERA_DEVICE
- CAMERA_DEVICE_READY
- STARTING_STREAM
- STREAMING
- STOPPING_STREAM
- CLOSING_CAMERA_DEVICE
- CAMERA_DEVICE_CLOSED
- ERROR

These **enums** are listed in sequence, as if opening a camera (fresh build), then starting or resuming streaming, then stopping streaming, then closing the camera.

Notes and Guidelines for Enums

- `OPENING_CAMERA_DEVICE` - no vision processing is happening
- `CAMERA_DEVICE_READY` - Camera is open. No processing is happening, including background processing from EOCV (i.e. pulling frames and performing color conversion). Ready to call `resumeStreaming()`
- `STARTING_STREAM` - no processing is happening
- `STREAMING` - Frames are available for processing (AprilTag and/or TFOD recognitions) and preview (RC preview and DS Camera Stream)
- `STOPPING_STREAM` - processing may or may not be happening. This status is followed by `CAMERA_DEVICE_READY`.
- `CLOSING_CAMERA_DEVICE` - no processing is happening
- `CAMERA_DEVICE_CLOSED` - nothing is running, USB comms are closed. Once closed, don't open camera again during this OpMode.

Observing Controls

Teams wanting to optimize AprilTag or TFOD recognitions with Camera Controls should consider using `scrcpy` here:

- https://github.com/Genymobile/scrcpy

or an HDMI monitor. Optimizing via DS Camera Stream will be less effective and less efficient.

DS Camera Stream shows the same images as `scrcpy`, namely with Exposure and Gain affecting recognitions. But the image is a snapshot only, and adjustments cannot be made in real time, with gamepads disabled during Camera Stream.

Control Ranges

Each webcam model has its own level of support for Camera Controls.

The Logitech C920 supports all the control features offered by the SDK; many webcams don't. More info is at *ftc-docs Webcam Controls*.

For example, here are control ranges reported by the Logitech C920:

- Exposure 0 to 204 ms
- Gain 0 to 255
- White Balance 2000 to 6500 (often defaults to 2000)
- Focus 0 to 250, usual default 0 (infinite)
- Pan and Tilt +/- 36,000, default (0,0)
- Zoom 100 to 500, but no effect after about 250

Note that Camera Control Zoom affects both AprilTag and TFOD, while TFOD Zoom affects only TFOD recognitions.

Camera Control zoom (PTZ) will affect the camera calibration, thus significantly affecting the **pose estimation**. TFOD zoom will not affect pose.

Also:

- AprilTag detections are not affected by camera or object orientation
- TFOD recognitions are affected by camera or object orientation

Setter Blocks

The **setter Blocks** under Webcam Controls can now change/toggle, when choosing “use return value” or “ignore return value” from each Block’s context (right-click) menu.

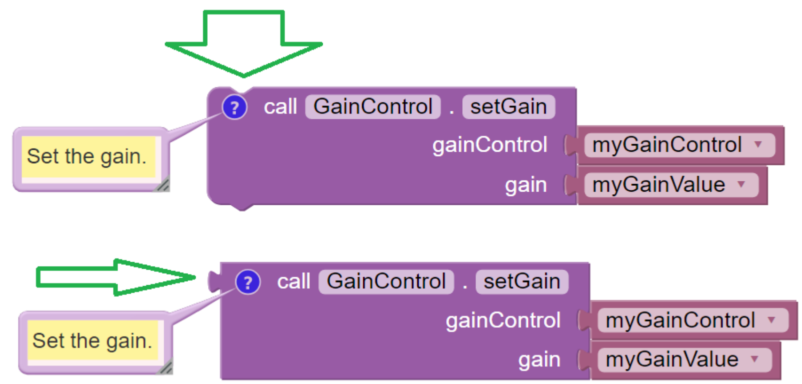


Fig. 71: Examples of Setter Blocks with togglable return values

In either form, the setting task **is performed**.

The “non-return” version comment is:

> *Set the gain. Right-click, “use return value” for a Boolean indicating success or completion.*

The “plug” version comment is:

> *Set the gain, and return a Boolean indicating success or completion. Or right-click, “ignore return value”.*

Gain and Exposure

Autoexposure mode manages both gain and exposure.

Gain can be adjusted only if ExposureControl Mode is set to MANUAL (not the default).

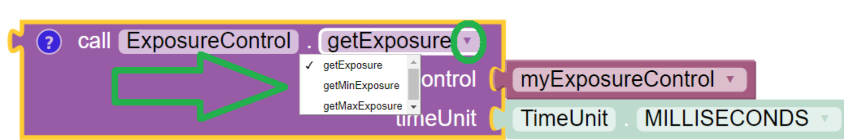
The old *Camera Controls tutorial <programming_resources/vision/webcam_controls/index:webcam controls>* says:

> *Gain can be managed in coordination with exposure.*

Actually, in the SDK, Gain **must** be managed with Exposure.

Shared Blocks

FTC Blocks offers an arrangement where 3 similar Blocks use a pull-down list to share a common structure (and common comment):



This is used six places in the Webcam Controls section.

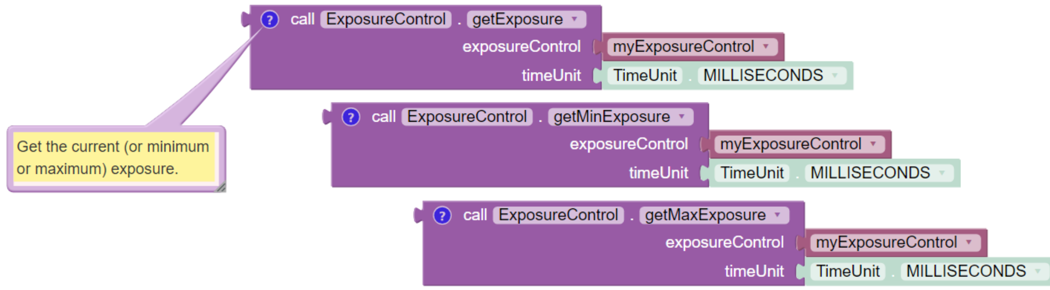


Fig. 72: Examples of Exposure Blocks with pull-down lists

Pan-Tilt Holder

See this Block with the NEW operator (green oval):

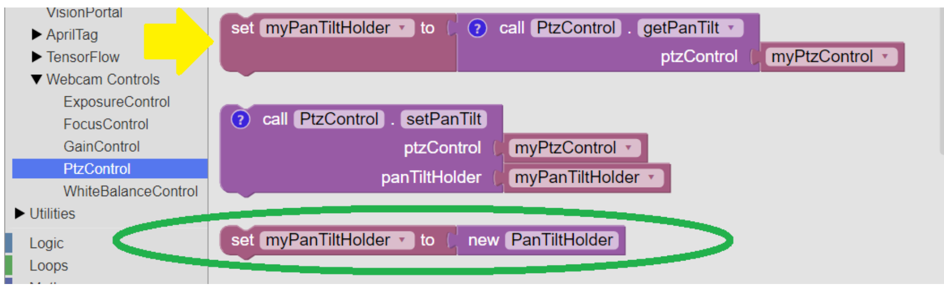


Fig. 73: Examples of Pan/Tilt Blocks

It's **not needed** if the OpMode will call `getPanTilt()` and assign it to the variable, as shown above (yellow arrow).

It **is needed** if instead the OpMode will next try to get (or set) that variable's pan and/or tilt values, or try to pass that variable to `setPanTiltHolder()`. The values will be zero.

Gain and Exposure Test OpMode

The SDK offers a built-in test OpMode to optimize Gain and Exposure. See the Sample Java Sample called `ConceptAprilTagOptimizeExposure.java`.

From its introduction notes:

> *This OpMode determines the best Exposure for minimizing image motion-blur on a webcam. Note that it is not possible to control the exposure for a Phone Camera, so if you are using a Phone for the Robot Controller this OpMode/Feature only applies to an externally connected Webcam.*
>
> *The goal is to determine the smallest (shortest) Exposure value that still provides reliable Tag Detection. Starting with the minimum Exposure and maximum Gain, the exposure is slowly increased until the Tag is detected reliably from the likely operational distance.*
>
> *The best way to run this optimization is to view the camera preview screen while changing the exposure and gain.*
>
> *To do this, you need to view the RobotController screen directly (not from Driver Station) This can be done directly from a RC phone screen (if you are using an external Webcam), but for a Control Hub you must either plug an HDMI monitor into the Control Hub HDMI port, or use an external viewer program like scrcpy (https://scrcpy.org/)*

Other Test OpModes

As an alternate, Camera Controls can be tested using these Blocks OpModes:

- Exposure & Gain
- Focus
- Pan, Tilt, Zoom (PTZ)
- White Balance

For Java versions, click `Export to Java` at the Blocks editing interface.

Another test OpMode is posted here and shown below. It uses 7 of the 11 Exposure Control Blocks, omitting 4 unlikely to be used.

The gamepad can raise and lower the webcam's **Exposure value**, while observing the **live effect** on previews and TFOD recognitions. This allows a team to quickly find their preferred Exposure value in that environment.

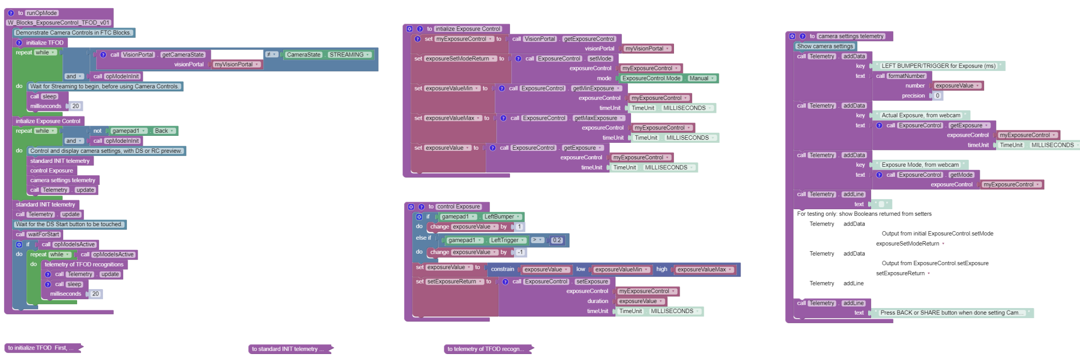


Fig. 74: Blocks Exposure OpMode Example

Questions, comments and corrections to westsiderobotics@verizon.net

Vision MultiPortal

The SDK can accommodate two portals, each with full features including AprilTag and TFOD processors, and even switchable cameras. USB Bandwidth must be considered, especially for webcams sharing an external USB hub.

Viewport ID

Each portal is assigned a `Viewport ID` by the Android operating system. At initialization, the OpMode must **capture** and use these ID numbers for operating the portals.

Android typically assigns a different Viewport ID number with each run of an OpMode. If desired, you could observe this by sending Telemetry to the Driver Station.

The `makeMultiPortalView()` Block or method returns a list of Viewport IDs. Each ID must be extracted from the list, then provided to each VisionPortal Builder using the `setCameraMonitorViewId()` Block or method.

"Dual cameras" was previously (and still is) available with EasyOpenCV. Now this is possible within the SDK.

Test OpMode

A sample FTC Blocks OpMode is posted here to demonstrate AprilTag detections from **two cameras at the same time**. For a Java version, click `Export to Java` in the Blocks editing window.

Here's the image of that test OpMode. Careful study will allow a better understanding of the Blocks and Java methods to create and operate multiple camera streams at the same time.

Right-click to open image in new browser tab, to magnify on a large PC screen.

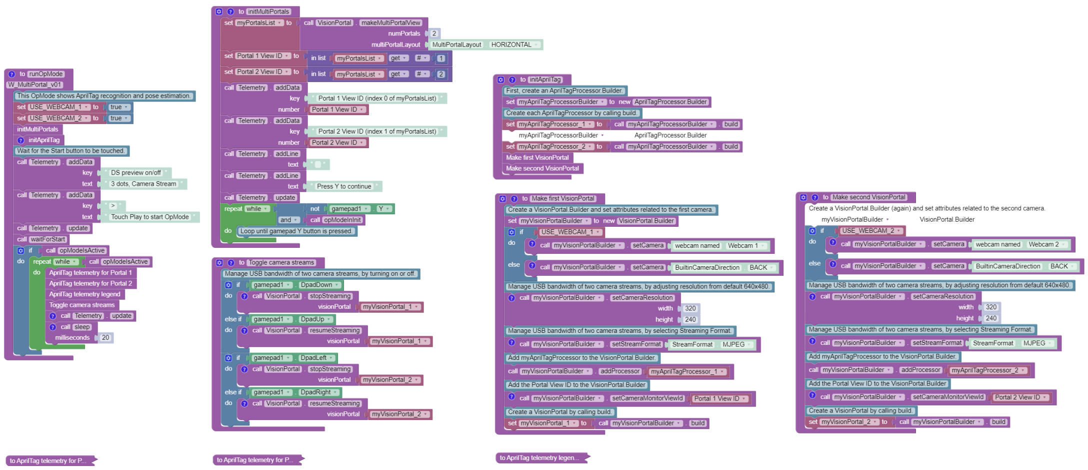


Fig. 75: Example Blocks Multiportal OpMode

On a Moto e4 RC phone, the OpMode can run the built-in phone camera along with a webcam.

On a Control Hub, it can run two webcams:

- both plugged in directly to the Hub, or
- both plugged into an unpowered USB Hub (with more restricted USB bandwidth)

Dual Previews

The dual RC previews can be displayed as `VERTICAL`, or side-by-side with the enum `HORIZONTAL`:

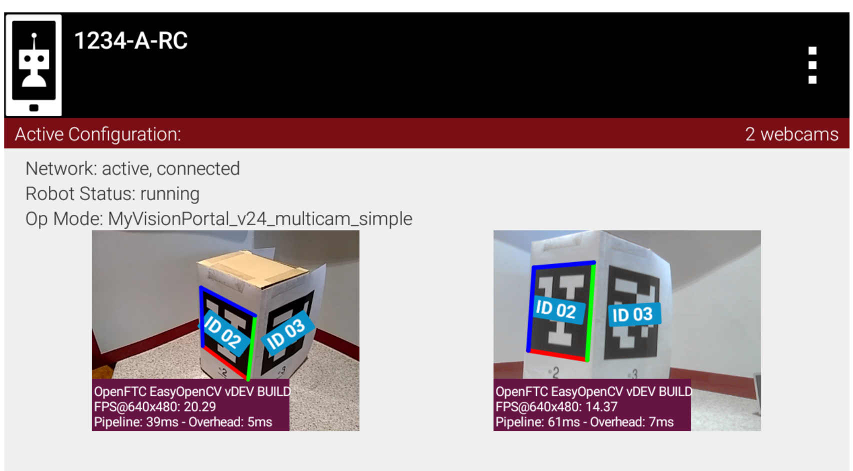


Fig. 76: Dual RC Previews

The DS Camera Stream preview can display only one camera's view (a known characteristic).

Gracious Professionalism® - "Doing your best work while treating others with respect and kindness - It's what makes FIRST, first."

USB Bandwidth

USB Bandwidth is a concern for dual **webcams**; internal phone cameras have an independent high-speed interconnect (not USB), unaffected by an added USB webcam.

See the USB bandwidth analysis at the **Managing CPU and Bandwidth** page.

The two webcams do *not* need to use the same format or resolution. For the testing mentioned above, the same format and resolution were applied to a Logitech C920 and a Logitech C270.

Control Hub

For dual webcams **plugged directly into the Control Hub**, the USB 2.0 and USB 3.0 ports are on different buses.

This reduces the concern about USB bandwidth capacity, although higher resolution causes the auto-optimized frame rate to reduce (see test data below).

Here the choice of stream format has little impact. But the USB 2.0 bus also carries the Control Hub's **WiFi radio**; adding a webcam may affect its reliability.

External USB Hub

On the other hand, both webcams on an **external USB Hub** (plugged into the CH 3.0 port) can exceed **USB bandwidth limits** (not quantified here).

Under the legacy **YUY2 format**, one webcam or the other may stop streaming above roughly 640x360 resolution. This is indicated by no detections, and a blue screen in RC preview via `scrcpy`.

Under **MJPEG format**, resolutions under roughly 432x240 may degrade the image to prevent AprilTag detection on at least 1 webcam, while higher resolutions may occasionally stop the RC app or crash the Control Hub.

For both formats, higher resolution reduces frame rate. The **Managing CPU and Bandwidth** page discusses testing, tradeoffs and optimization.

Teams can evaluate these tradeoffs, assisted by the new reporting feature `getFps()`, providing Frames Per Second (FPS). It's available for Blocks and Java.

Questions, comments and corrections to westsiderobotics@verizon.net

AprilTag Advanced Use

Overview

This page will offer tips for *FIRST* Tech Challenge teams seeking more info about specialized features of the new VisionPortal.

Optional Metadata

An AprilTag Library tag can store two optional **Metadata** fields (of these Blocks/Java types):

- `fieldPosition`: tag location on the game field (`VectorF`)
- `fieldOrientation`: tag orientation on the game field (`Quaternion`)

The reference frame is the *FIRST* Tech Challenge **Field Coordinate System**, provided here:

- *Field Coordinate System*

Here is a simple graphic showing the *FIRST* Tech Challenge global axes that apply to **every game, every season**:

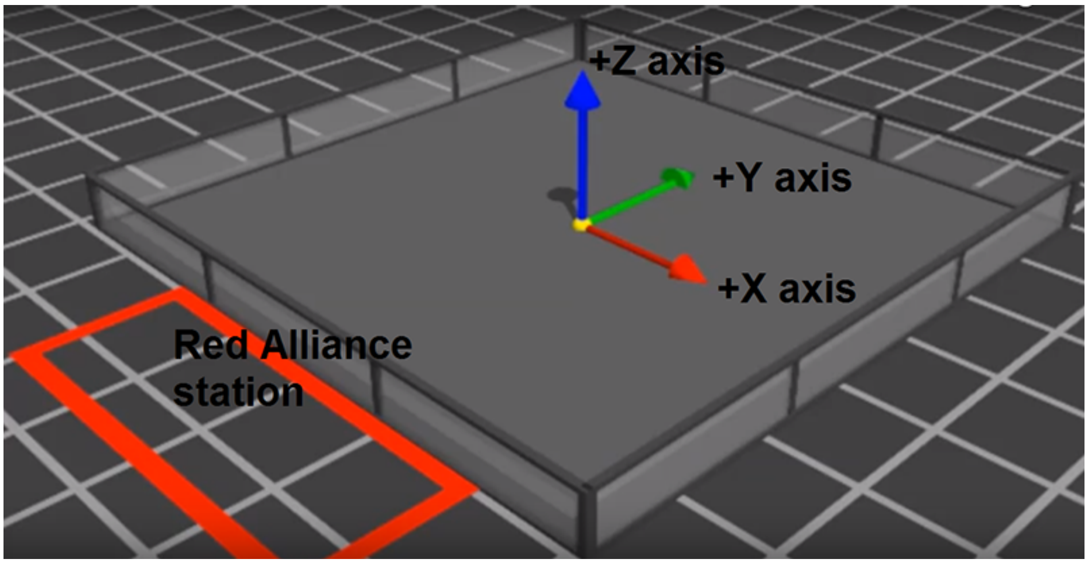


Fig. 77: Image Credit: Phil Malone

With a tag's **field position** and **orientation** specified in advance as Metadata, the tag's pose data could be used by an advanced OpMode to calculate the robot's position on the field. This conversion math, an exercise for the reader, can allow a robot to use the tag's pose data in real-time to navigate to the desired location on the field.

Raw Pose Values

The frame of reference described at the **AprilTag Reference Frame** page is provided **by default** in the new 8.2 SDK.

Advanced teams may prefer to perform their own pose calculations, based on **raw values** from the AprilTag/EasyOpenCV pipeline.

Those raw values are available to Java and Blocks programmers. The Java version is shown here:

```
for (AprilTagDetection detection : aprilTag.getDetections())  {

     Orientation rot = Orientation.getOrientation(detection.rawPose.R, AxesReference.INTRINSIC,␣
↪AxesOrder.XYZ, AngleUnit.DEGREES);

     // Original source data
     double poseX = detection.rawPose.x;
     double poseY = detection.rawPose.y;
     double poseZ = detection.rawPose.z;

     double poseAX = rot.firstAngle;
```

(continues on next page)

(continued from previous page)

```
    double poseAY = rot.secondAngle;
    double poseAZ = rot.thirdAngle;
    }
```

These raw values are converted by the SDK to the default interface, as follows:

```
if (detection.rawPose != null)   {
    detection.ftcPose = new AprilTagPoseFtc();

    detection.ftcPose.x =  detection.rawPose.x;
    detection.ftcPose.y =  detection.rawPose.z;
    detection.ftcPose.z = -detection.rawPose.y;

    Orientation rot = Orientation.getOrientation(detection.rawPose.R, AxesReference.INTRINSIC, AxesOrder.YXZ, outputUnitsAngle);
    detection.ftcPose.yaw = -rot.firstAngle;
    detection.ftcPose.roll = rot.thirdAngle;
    detection.ftcPose.pitch = rot.secondAngle;

    detection.ftcPose.range = Math.hypot(detection.ftcPose.x, detection.ftcPose.y);
    detection.ftcPose.bearing = outputUnitsAngle.fromUnit(AngleUnit.RADIANS, Math.atan2(-detection.ftcPose.x, detection.ftcPose.y));
    detection.ftcPose.elevation = outputUnitsAngle.fromUnit(AngleUnit.RADIANS, Math.atan2(detection.ftcPose.z, detection.ftcPose.y));
    }
```

Further discussion is provided here:

Understanding AprilTag Detection Values

Advanced CPU Management

This section continues from the **VisionPortal CPU and Bandwidth** page, which covered many basic tools for avoiding limits of CPU usage and USB bandwidth.

To evaluate multiple factors, changing at the same time, a customized Test OpMode can be very useful. This section provides an example that allows **live gamepad control** to:

- toggle AprilTag Processor on and off
- toggle TFOD Processor on and off
- toggle LiveView on and off
- toggle Streaming on and off

Other features of this Test OpMode include:

- All controls are independent, to explore the combinations and their effect on frame rate (FPS).
- The previews can be observed, and detections/recognitions can be monitored via annotations and Telemetry.
- Frame rate is provided in LiveView and DS Telemetry.
- The Telemetry functions include an alternate for getting **all** or **only fresh** detections/recognitions.

This Test OpMode can be downloaded for FTC Blocks or Java. The Blocks version is shown below; right-click to open in a new browser tab and zoom in.

The OpMode uses "Webcam 1", or change USE_WEBCAM for a built-in RC phone camera. For Control Hub, set up an HDMI monitor or scrcpy. Follow the DS gamepad button guide.

At that **VisionPortal CPU and Bandwidth** page, four tools mentioned were not discussed:

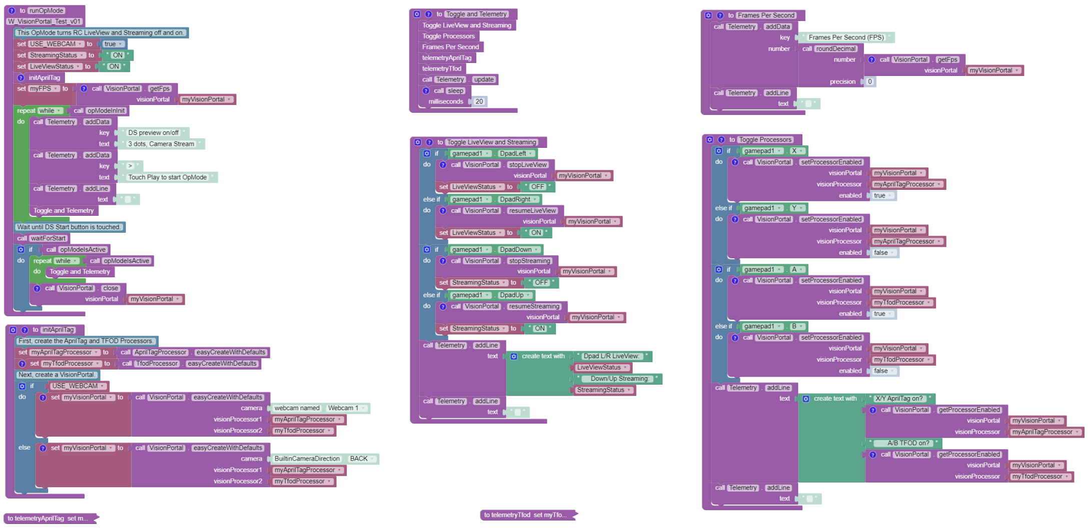


Fig. 78: VisionPortal Example OpMode

- set decimation (down-sampling)
- select a pose solver algorithm
- get all or only fresh detections from the AprilTag Processor
- get all or only fresh recognitions from the TFOD Processor

For now, these are left for interested Blocks and Java users to research and investigate. In time, more information may be posted at this page.

All of the above features are easily found in the relevant **FTC Blocks** toolbox, or palette, under the Vision category.

Java users should review the VisionPortal interface at the SDK Javadocs site. Click **FRAMES** for easy navigation.

Questions, comments and corrections to westsiderobotics@verizon.net

---

Much credit to

- EasyOpenCV developer @Windwoes
- FTC Blocks developer @lizlooney
- *FIRST* Tech Challenge navigation expert @gearsincorg
- and the smart people at UMich/AprilTag.

Questions, comments and corrections to westsiderobotics@verizon.net

21.3.3 Webcams for Vision Portal

This is a short list of common webcams that are known to work with the *FTC VisionPortal* and the *FTC Camera Controls*.

VisionPortal is a comprehensive new interface for FTC vision processing, including AprilTag and TensorFlow Object Detection (TFOD).

Many more webcams can work with the FTC VisionPortal; this is a short list of models with built-in calibrations suitable for AprilTag *pose estimation*.

Logitech C270

Fig. 79: Logitech C270 Camera

The Logitech C270 is available at the FIRST Storefront for new FTC teams, and at many online retailers.

FTC Hot Take:

- The Logitech C270 is Logitech's budget line of webcams. It is incredibly inexpensive, and fairly reliable. The C270 is the workhorse webcam for *FIRST* Tech Challenge.
- The Logitech C270 has a 60-degree field of view, and a maximum frame rate of 30fps at 720p which makes it a reasonable choice for vision processing.
- The audio quality of the C270 is lackluster, but audio is not generally a factor in *FIRST* Tech Challenge.

Supported Resolutions: 160x120, 176x144, 320x176, 320x240, 352x288, 432x240, 544x288, 640x360, 640x480, 752x416, 800x448, 800x600, 864x480, 960x544, 960x720, 1024x576, 1184x656, 1280x720

The FTC SDK provides **built-in calibration values** for the FTC VisionPortal default resolution of 640x480, and no others. Learn more at *AprilTag Camera Calibration*.

Logitech C310

Fig. 80: Logitech C310 Camera

The Logitech C310 is available at some online retailers.

FTC Hot Take:

- The Logitech C310 is also in Logitech's budget line of webcams. It is slightly more expensive than the C270, and is a marginal step up.
- Like the C270, the Logitech C310 has a 60-degree field of view, and a maximum frame rate of 30fps at 720p which makes it a reasonable choice for vision processing.
- The C310 has slightly better color correction and dynamic color range than the C270, but these likely won't be realized without using the webcam control interface provided by the FTC SDK.
- The audio quality of the C310 is slightly better than the C270, but again audio is not generally a factor in *FIRST* Tech Challenge.

Supported Resolutions: not published; probably similar to Logitech C270.

The FTC SDK provides **built-in calibration values** for the FTC VisionPortal default resolution of 640x480, and for 640x360. Learn more at *AprilTag Camera Calibration*.

Logitech C920

Fig. 81: Logitech C920 Camera

The Logitech C920 is available at many online retailers.

FTC Hot Take:

- The Logitech C920 is in Logitech's mid-range line of webcams. It is slightly more expensive than the C310, but is a dramatic step-up in quality. If you find a C310 for almost the same price as a C920, just buy the C920.
- The Logitech C920 has a 78-degree field of view, and a maximum frame rate of 30fps at 1080p which makes it a fabulous choice for vision processing. The C920 also includes an auto-focus option, whereas the C270 and C310 are fixed-focus, though the auto-focus tends to be slow.
- The C920 has additional options for mounting the camera, with a 1/4 inch threaded mount. The C920 also has a much better mounting clip.
- The audio quality of the the C920 is phenomenally better than the C270, or C310, but again audio is not generally a factor in *FIRST* Tech Challenge.

Supported Resolutions: 160x90, 160x120, 176x144, 320x180, 320x240, 352x288, 432x240, 640x360, 640x480, 800x448, 800x600, 864x480, 960x720, 1024x576, 1280x720, 1600x896, 1920x1080, 2304x1296, 2304x1536.

The FTC SDK provides **built-in calibration values** for the FTC VisionPortal default resolution of 640x480, and five others: 640x360, 800x448, 800x600, 864x480, and 1920x1080. Learn more at *AprilTag Camera Calibration*.

Microsoft LifeCam HD-3000 v1/v2

Fig. 82: Microsoft LifeCam HD-3000 v1/v2

The Microsoft LifeCam HD-3000 is available at some online retailers.

FTC Hot Take:

- The Microsoft LifeCam HD-3000 has been a mainstay in *FIRST* Robotics Competition for a number of years, so it's likely a local team might have one they will just give you. The HD-3000 has been around for over 10 years, with a "don't fix what isn't broken" mentality. It defines the "budget" part of Microsoft's "budget" line of webcams.
- The HD-3000 sports a 68.5 degree field of view, slightly wider than the Logitech C270 and C310 webcams, at 30fps at 720p (same as the others).
- The HD-3000 is as "no-frills" as it gets otherwise, but at its price point that shouldn't be much of a surprise.

Supported Resolutions: not published; up to 1280x720.

For v1 and v2 of this webcam, the FTC SDK provides **built-in calibration values** for the FTC VisionPortal default resolution of 640x480, and no others. Learn more at *AprilTag Camera Calibration*.

Other Webcams

Many other webcams are available online, with and without published UVC compatibility. The FTC SDK supports **only** UVC compatible webcams. Many modern webcams are UVC compatible without specifically advertising it; often indicated by "no drivers needed".

In general, other webcams (not listed above) will require user-provided *Camera Calibration Values*. for AprilTag *pose estimation*.

A digital camera opens its shutter to allow light ("the image") to reach the detector's array of small sensors (pixels). (Webcam shutters are typically electronic, not mechanical.) Most webcams use a **"rolling shutter"**, where the the image data is read **one pixel row at a time**.

Another type of webcam, called **"global shutter"**, reads all pixels at the same time. This can help when the webcam (robot) is moving fast. Teams are encouraged to research the characteristics of rolling shutter vs. global shutter.

One difference is that many global shutter cameras use a compressed video format called **MJPEG**. This saves bandwidth, to offset a higher frame rate (frames per second or FPS). The FTC VisionPortal uses a default (uncompressed) video format called **YUY2**, but does allow specifying MJPEG.

Below is one example of a global shutter webcam, tested to work with the FTC VisionPortal.

Arducam Global Shutter 120 FPS

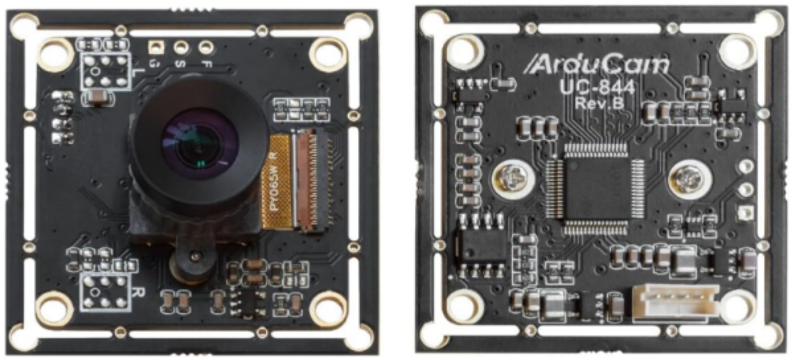


Fig. 83: Arducam GS 120 Camera

The Arducam Global Shutter 120 FPS is available at some online retailers, including Amazon.

FTC Hot Take:

- The Arducam OV9281 Global Shutter camera can pump out 100+fps in MJPG mode at full resolution, with phenomenal resistance to motion blur effects (due to the Global Shutter design).
- The Arducam OV9281 is a monochrome (black&white) camera, so applications needing color should look elsewhere.
- The Arducam OV9281 is fantastic in low-light scenarios, and has a very low-distortion lens making it perfect for object tracking and motion detection.
- The Arducam required a patch to the SDK and EasyOpenCV to work properly at high speeds, so it is not guaranteed to work properly with the FTC SDK prior to SDK 9.0.
- The FTC software have been observed to not function properly with more than one Arducam OV9281 at a time. If you encounter this issue please refer to the *Serial Number Tool <https://docs.arducam.com/UVC-Camera/Serial-Number-Tool-Guide/>* to reassign at least one of the Arducam serial numbers.

Supported Resolutions in YUY2 format: 1280x720, 1280x800. Note frame rate limitations.

Supported Resolutions in MJPEG format: 320x240, 640x480, 800x600, 1280x720, 1280x800.

The FTC SDK provides **no** built-in calibration values for this webcam. Learn more at *AprilTag Camera Calibration*.

Other Global Shutter Cameras

Two other tested global shutter webcams (offering different resolutions than the Arducam) are from

- Kayeton
- ELP

both of these are available from AliExpress and other online retailers.

Quick Summary

This below table summarizes the most common and known-supported cameras with the *FIRST* Tech Challenge SDK, including resolutions with built-in calibrations and those without calibrations.

Table 6: Cameras and Supported Resolutions

Camera	Features	Resolutions with Built-In Calibrations	Resolutions without Calibrations
Logitech C270	60 DegFOV, 30fps@720p	640x480	160x120, 176x144, 320x176, 320x240, 352x288, 432x240, 544x288, 640x360, 752x416, 800x448, 800x600, 864x480, 960x544, 960x720, 1024x576, 1184x656, 1280x720
Logitech C310	60 DegFOV, 30fps@720p	640x480, 640x360	All other resolutions
Logitech C920	78 DegFOV, 30fps@1080p	640x480, 640x360, 800x448, 800x600, 864x480, 1920x1080	160x90, 160x120, 176x144, 320x180, 320x240, 352x288, 432x240, 960x720, 1024x576, 1280x720, 1600x896, 2304x1296, 2304x1536
Microsoft LifeCam HD-3000 v1/v2	68.5 DegFOV, 30fps@720p	640x480	All other resolutions
Arducam Global Shutter 120 FPS	70 DegFOV, 120fps@1280x800 MJPG, Monochrome	No Built-In Calibrations	MJPEG: 320x240, 640x480, 800x600, 1280x720, 1280x800; YUY2: 1280x720, 1280x800
Kayeton Global Shutter (*Other Global Shutter Cameras*)	70 DegFOV, 120fps@720p MJPG, Monochrome	No Built-In Calibrations	All resolutions
ELP Global Shutter (*Other Global Shutter Cameras*)	70 DegFOV, 90fps@1920x1200 MJPG, Monochrome	No Built-In Calibrations	All resolutions

Questions, comments and corrections to westsiderobotics@verizon.net

21.3.4 Understanding AprilTag Detection Values

Last Updated: 7/05/2023

Introduction

When an AprilTag is detected by the new SDK vision processing system, the core code returns a collection of raw data that is often not easily interpreted. However, the data can be further transformed into a familiar frame of reference to make it more easily utilized.

In the *FIRST* Tech Challenge SDK, the AprilTag API will present the Team OpMode with a collection of translation and rotation values, called *ftcPose*, that represent the Tag's position in 3D space.

To understand how to interpret these values, it's easier to consider a simpler 2D scenario where the vertical component is ignored. This is what is described below.

Figure 1 below represents one possible 2D scenario.

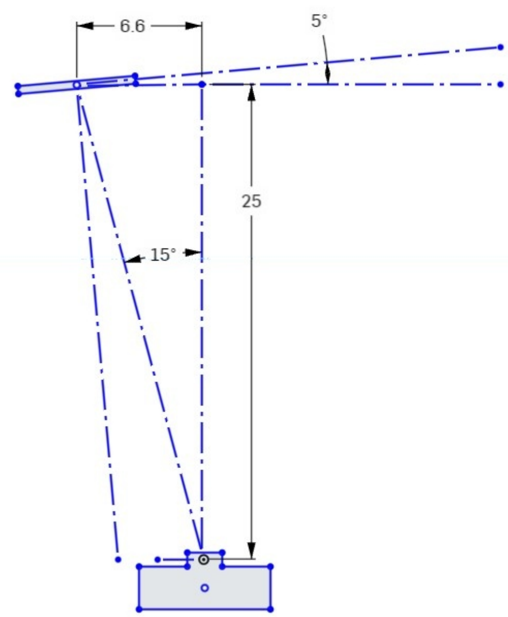


Fig. 84: Figure 1: Top view of AprilTag scenario

This diagram looks down on the Camera and AprilTag from above. The camera's "forward" direction is identified by a dashed line drawn straight out from the camera.

The AprilTag image is shown in the upper left of the figure. The tag is located 100 units forward of the camera and 36.4 units to the left (measured at right angles to the forward view).

The AprilTag is also rotated 5 degrees counterclockwise from a normal "face on" orientation.

Now that we have a clear understanding of one possible detection scenario, we can look at the meaning of the various values returned as *ftcPose* by the SDK.

Figure 2 shows the measured values associated with the camera/target scenario shown in Figure 1.

Since this is a simple 2D diagram, the vertical "Z" (up) axis is being ignored, so it is not shown here.

The green X axis value represents the sideways offset to the tag. Note that this value is negative (to the left of the camera center).

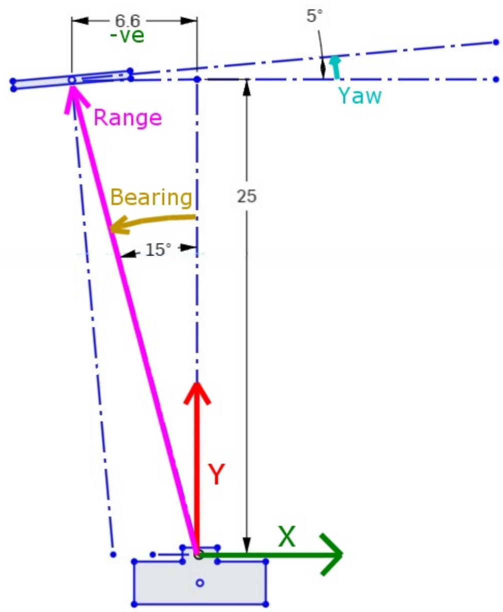


Fig. 85: Figure 2: Measured values associated with scenario

The red Y axis value represents the forward distance to the Tag.

The cyan Yaw value represents the rotation of the tag around the Z axis. A Counter-Clockwise rotation is considered positive. Note that a Yaw value of zero means that the tag image is parallel to the face of the camera.

Note: Fun Fact: If the camera is pointing forward, the X, Y & Z axes are consistent with the Robot Coordinate system.

Three additional parameters are derived from the X and Y axis values, these are *Range* (which is the direct distance to the center of the target), *Bearing* (which is how many degrees the camera must turn to point directly at the target) and *Elevation* (which is how many degrees the camera must tilt UP to center on the tag). Note that Target Bearing has the same positive counterclockwise orientation.

Investigating some real data

To illustrate this process, consider some real-world tags. The data that follows is from a pair of tags printed on a card. The left-most tags has an identical setup as described above. In Figure 3 the protractor origin is positioned directly in front of the camera, at a distance of 25 inches. Both tags are to the left of the camera centerline, and both are rotated +5 degrees. The left tag is 6.6" from the centerline, and the right tag is 1.5" from the centerline. The camera is located 6" inches above the center of the targets, looking out horizontally (parallel to the ground).

The AprilTag video preview image from the Camera Stream preview is shown below. The left tag has an ID of 0 and the right tag has an ID of 1. This video is being captured by a Logitech C920 Pro HD webcam, running at 648x480 resolution. In this mode the camera has Field-Of-View (FOV) of 60 degrees. The physical tags are 3.4" square.

Notice that both tags are in the bottom-left corner of the image. The center of the image corresponds to the location the camera is pointed at, which is centered on the protractor and directly above the top of the tags.

Based on this setup, let's review the data returned by the "ConceptAprilTag.java" sample OpMode.

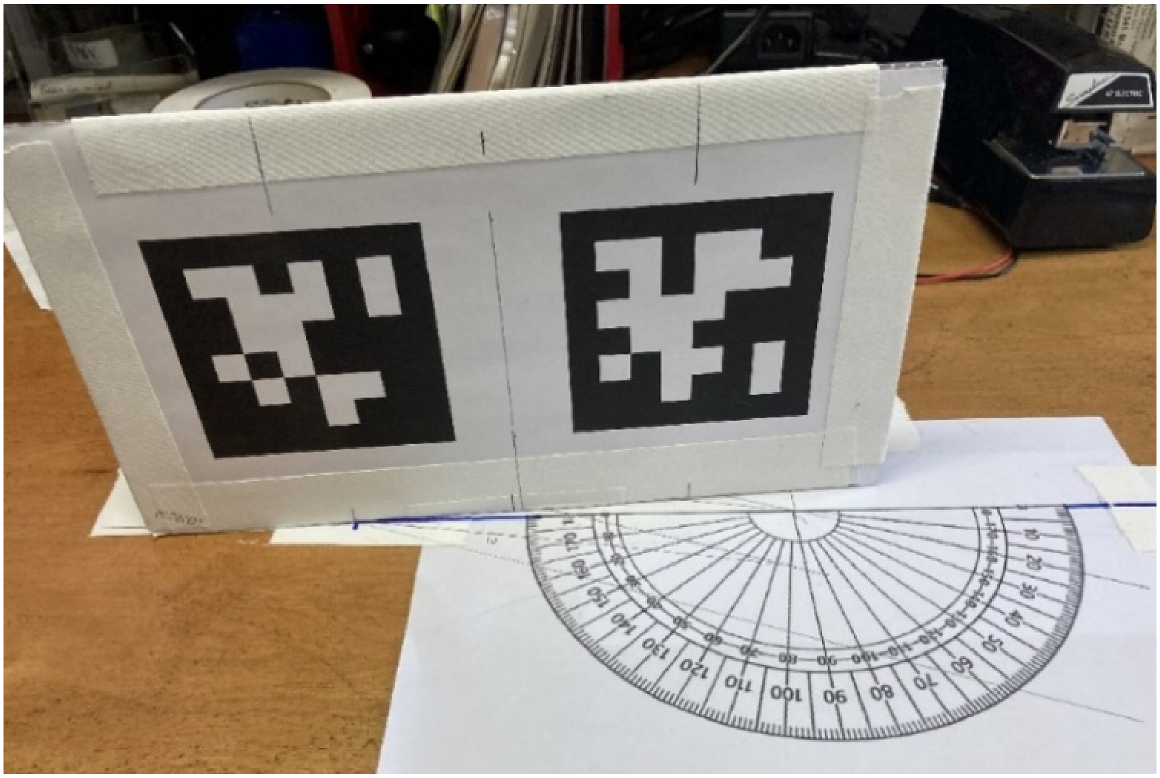


Fig. 86: Figure 3: Sample Tag setup for testing

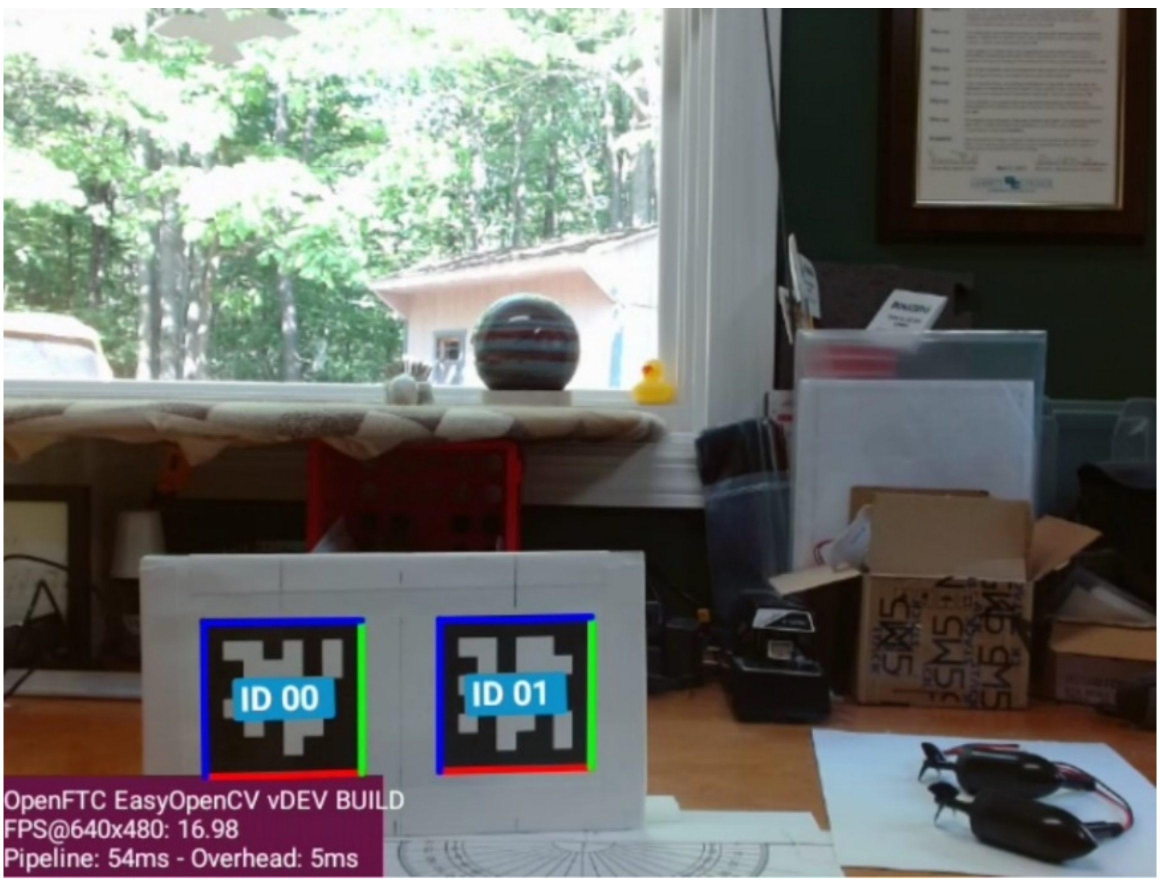


Fig. 87: Figure 4: Camera Preview showing two detected AprilTags

> **Warning:** Since the creation of this document, the tags used in the ConceptAprilTag.java sample have changed. Therefore, in order to reproduce this example the appropriate tags will need to be used instead of Tag0 and Tag1.

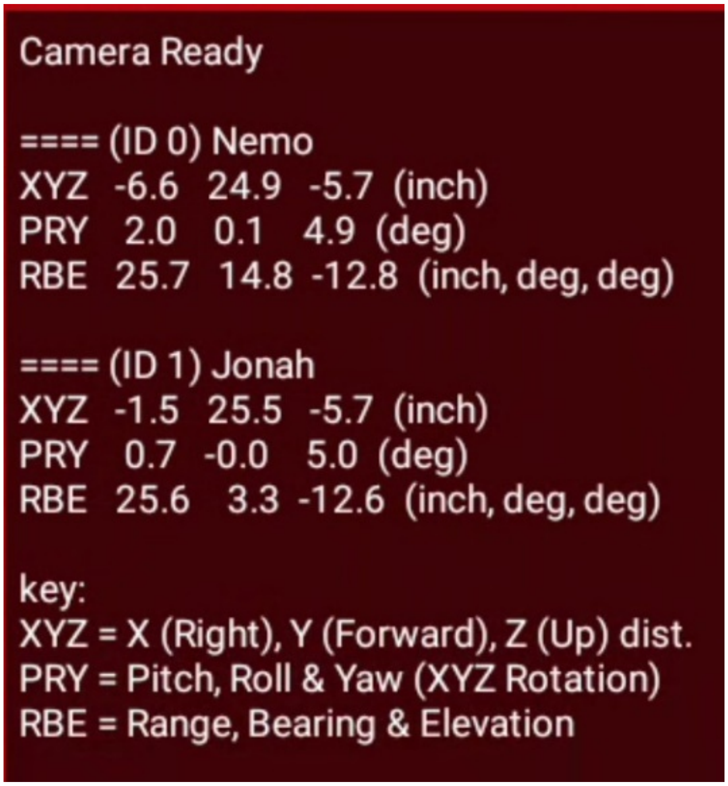


Fig. 88: Figure 5: Values displayed by AprilTag OpMode

The values for the two AprilTags are listed as "ID0 Nemo", and "ID1 Jonah". These are the names assigned when adding the tags to the Tag Library.

The OpMode displays values that correspond to those parameters shown in **Figure 2**. The XYZ line shows the three axes translation values (X, Y & Z) in inches. The PRY line shows the corresponding rotations (Pitch, Roll & Yaw) around those axes, in degrees. The RBE line shows the target Range (in inches), Bearing, and Elevation (in degrees). The angle of Elevation results from the height difference between the camera and the Tag.

Several items to observe:

- Both Y values are about 25", but the Y value for Tag 1 is slightly larger because it is behind the protractor base line.
- The X values for Tag 0 and 1 correspond to the offset distances described earlier (-6.6" and -1.5")
- Both tags show a Yaw of approximately 5 Deg, although this can vary 1-2 degrees depending on other orientation factors.
- The Range to both targets are almost equal but the Bearing of Tag 0 is much greater due to its displacement to the left.
- Both targets show the same negative Z value of -5.7, which is consistent with them being centered about 6" below the height of the camera.
- Each tag also has an "Elevation" of about -12.6 degrees, which is a downward viewing angle to the center of each tag.

Ways to use this data

There are several ways the AprilTag position data can be used, but here are two basic ways.

1. Pointing towards a target (Tank Drive).

   If an AprilTag is being used to mark the location of a target that you need to shoot towards, then the two main properties of interest are Tag Range and Tag Bearing. The Tag Bearing is an indication of how many degrees you would need to turn to point directly at the tag, and the Tag Range is an indication of how far you would need to shoot. Even with a simple differential (tank) drive, these two parameters would enable you to turn towards the target and drive to the correct range (or adjust your shooting power based on the range).

   A simple proportional controller could take the Tag Bearing, multiply it by a suitable gain and then use it in place of the turning joystick to turn the robot towards the target. Likewise, you could subtract the desired shooting range from the current Tag Range and use the result to control the robot's forward speed.

   Note that this approach does not guarantee that you are squared up to the front of the target, merely that you are pointing towards it. To get squarely aligned, you need to consider the Yaw angle as shown in the next approach.

   See SDK Sample: RobotAutoDriveToAprilTagTank.java for more info.

2. Approaching a target squarely (Omni Drive).

   If an AprilTag is being used to mark the location of something that must be approached squarely from the front, then it's important to consider the Tag Yaw value. This is a direct indication of how far off (in degrees) the camera is from the tag image's centerline. This is related to, but not the same as the Tag Bearing. So, all three parameters (Range, Bearing & Yaw) must be used to approach the target and end up directly in front of it.

   Reaching a certain distance directly in front of the target can be easily performed by a robot with a holonomic (Omni-directional) drive, because strafing can be used for direct sideways motion. A three-pronged approach can be used. 1) The Target bearing can be used to turn the robot towards the target (as described above). 2) The Target Yaw can be used to strafe sideways, thereby rotating around the target to get directly in front of it. 3) The target range can be used to drive forward or backward to obtain the correct standoff distance.

   Each of the three axis motions could be controlled by a simple proportional control loop, where turning towards the tag is given the highest gain (priority), followed by strafing sideways, followed by approaching the tag.

   See SDK Sample: RobotAutoDriveToAprilTagOmni.java for more info.

21.3.5 AprilTag Localization

Introduction

In *FIRST Tech Challenge* (FTC), **localization** uses sensor inputs to determine the robot's current place **on the game field**.

Since 2023, an FTC OpMode can read the **pose** (position and orientation) of an AprilTag, **relative to the camera**. An OpMode can also read that AprilTag's **global** pose (on the FTC game field), stored as metadata.

This means it's possible to calculate the camera's **global** pose – namely its position and orientation on the game field.

Furthermore, if the camera's pose is specified in the robot's reference frame, then an OpMode can determine the **global pose of the robot** (on the game field).

This **localization** is a calculation to determine the robot's global position and rotation, based on sensing one or more fixed landmarks – AprilTags in this case.

This capability is provided in 2024 with FTC SDK version 10.0, including a Sample OpMode, thanks to Dryw Wade. This tutorial describes how to use that OpMode.

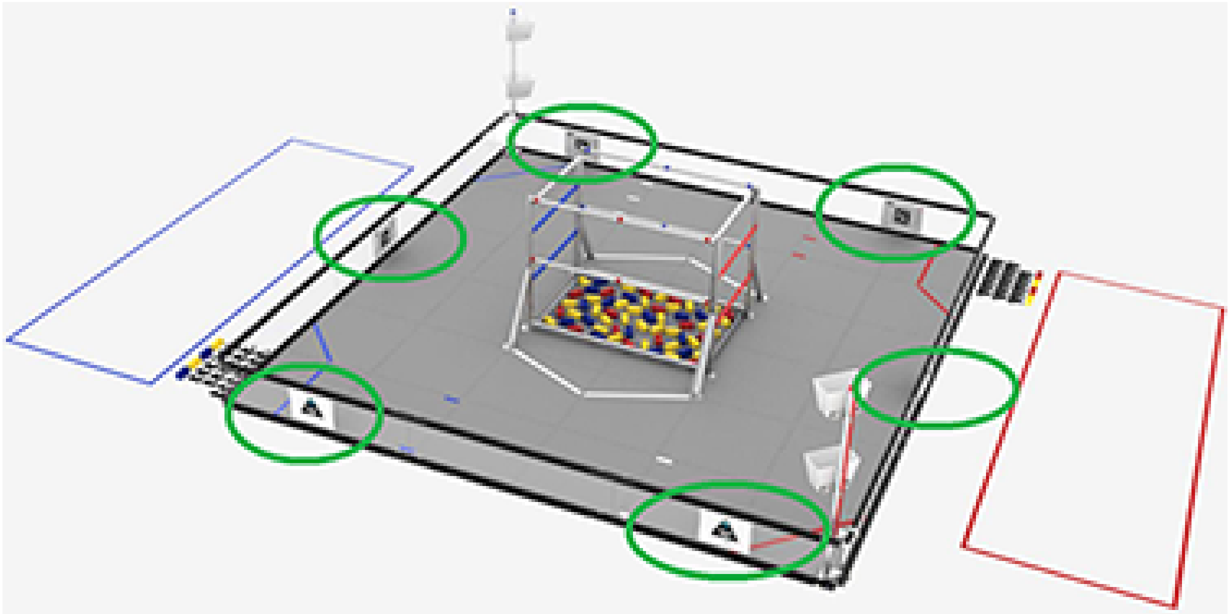


Fig. 89: Field locations of AprilTags in INTO THE DEEP

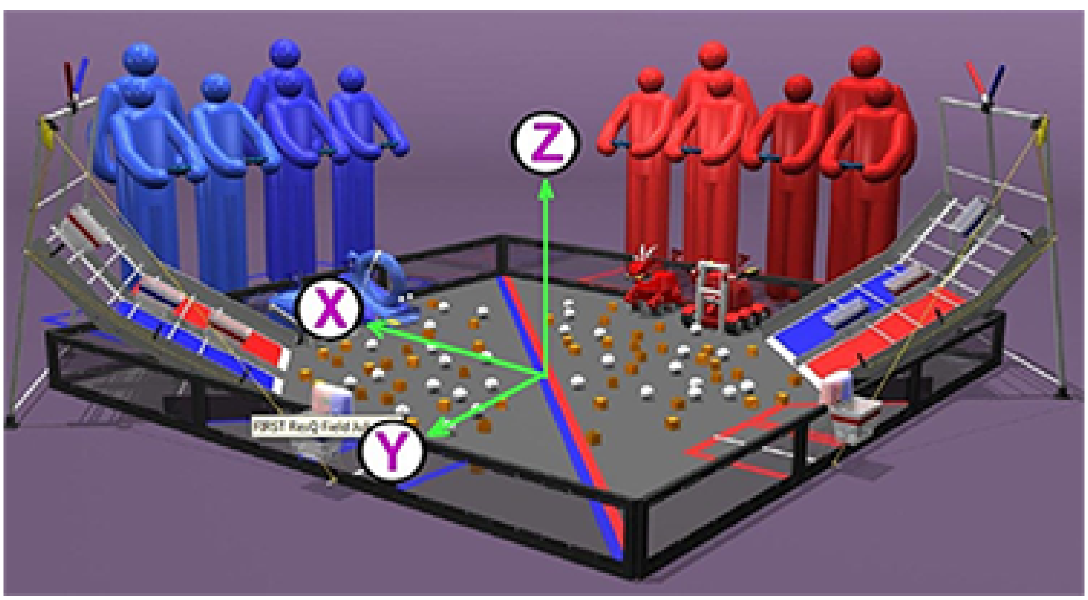


Fig. 90: FTC Field Coordinate System

Configuration

Skip this section if …

- *the active robot configuration already contains "Webcam 1", or*
- *using the built-in camera of an Android phone as Robot Controller.*

Before starting the programming, REV Control Hub users should make a robot configuration that includes the USB webcam to be used for AprilTag localization.

For now, use the default webcam name, "Webcam 1". If a different name is preferred, edit the Sample OpMode to agree with the exact webcam name in the robot configuration.

Save and activate that configuration; its name should appear on the paired Driver Station screen.

Open the Sample OpMode

To learn about opening the Sample OpMode, select and read the Blocks **or** Java section below:

Blocks

On a laptop or desktop computer connected via Wi-Fi to the Robot Controller, open the Chrome browser. Go to the REV Control Hub's address http://192.168.43.1:8080 (or http://192.168.49.1:8080 for Android RC phone) and click the Blocks tab.

Click `Create New OpMode`, enter a new name such as "AprilTagLocalization_Darlene_v01", and select the Sample OpMode `ConceptAprilTagLocalization`.

If using the built-in camera of an RC phone, change `true` to `false` at the OpMode's first Block called `set USE_WEBCAM`.

Save the OpMode, time to try it!

Java

Open your choice of OnBot Java or Android Studio.

In the `teamcode` folder, add/create a new OpMode with a name such as "AprilTagLocalization_Oscar_v01.java", and select the Sample OpMode `ConceptAprilTagLocalization.java`.

If using the built-in camera of an RC phone, change `true` to `false` at about line 71 (`USE_WEBCAM`).

Click "Build", time to try it!

Run the Sample OpMode

On the Driver Station, select the TeleOp OpMode that you just saved or built.

Aim the camera at an AprilTag from the current FTC game.

For real results, testing should be done on an FTC game field with one or more legal AprilTags posted in their correct positions.

For simulated/casual testing, use a loose paper AprilTag of the correct size. Or it may be on a computer screen, with the image zoomed to the **correct physical size** (4 x 4 inches, in this example):

Touch INIT only. No telemetry will appear, but at this moment the DS **Camera Stream** preview can be accessed. See the next section re. previews.

After using the preview to aim at the AprilTag, touch the DS Start arrow. The OpMode should give Telemetry showing the **localization results**:

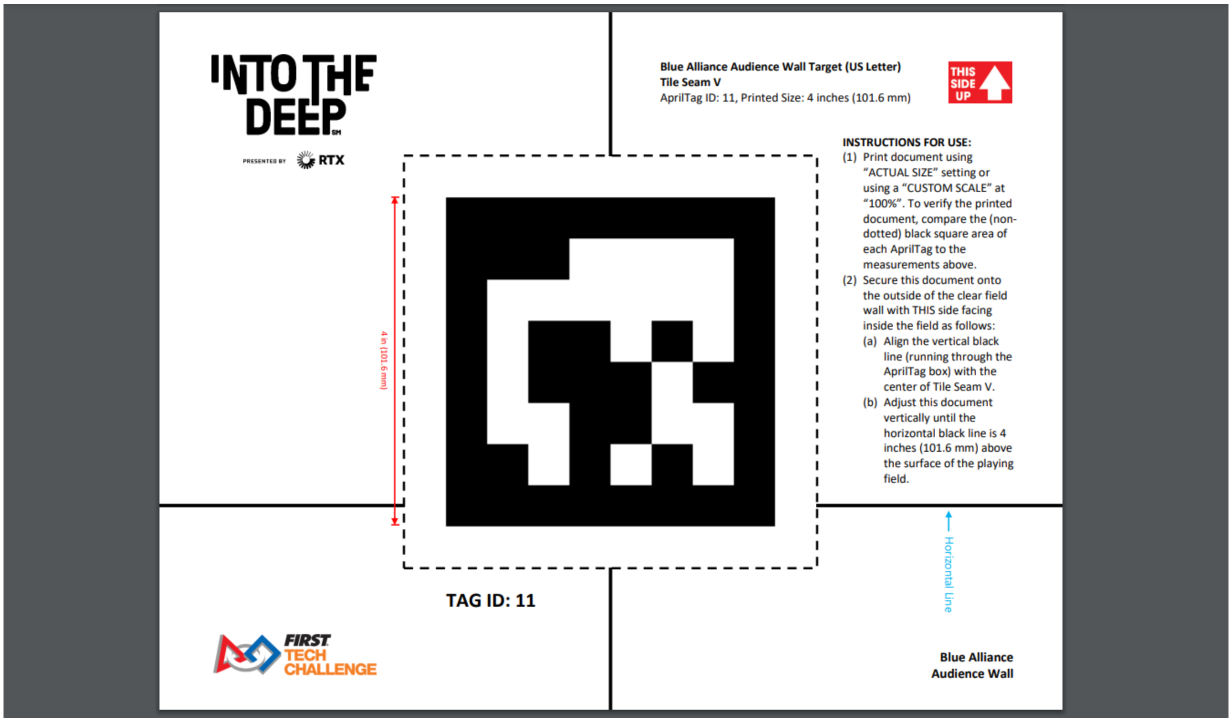


Fig. 91: Full AprilTag Image

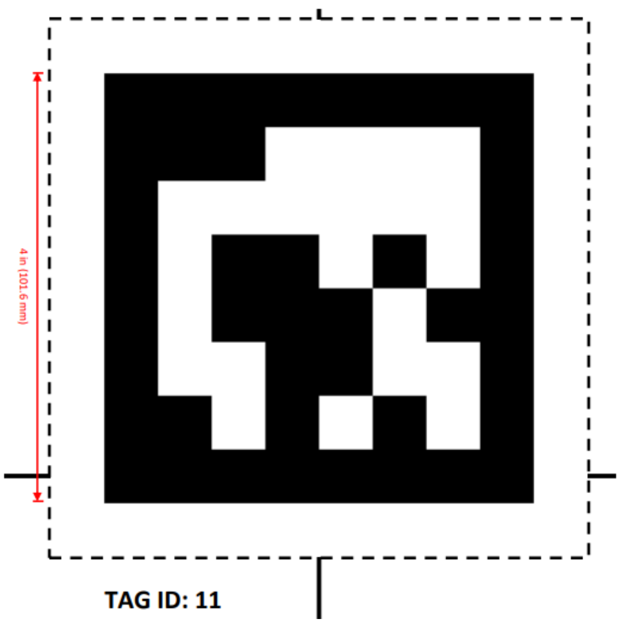


Fig. 92: Partial AprilTag Sheet

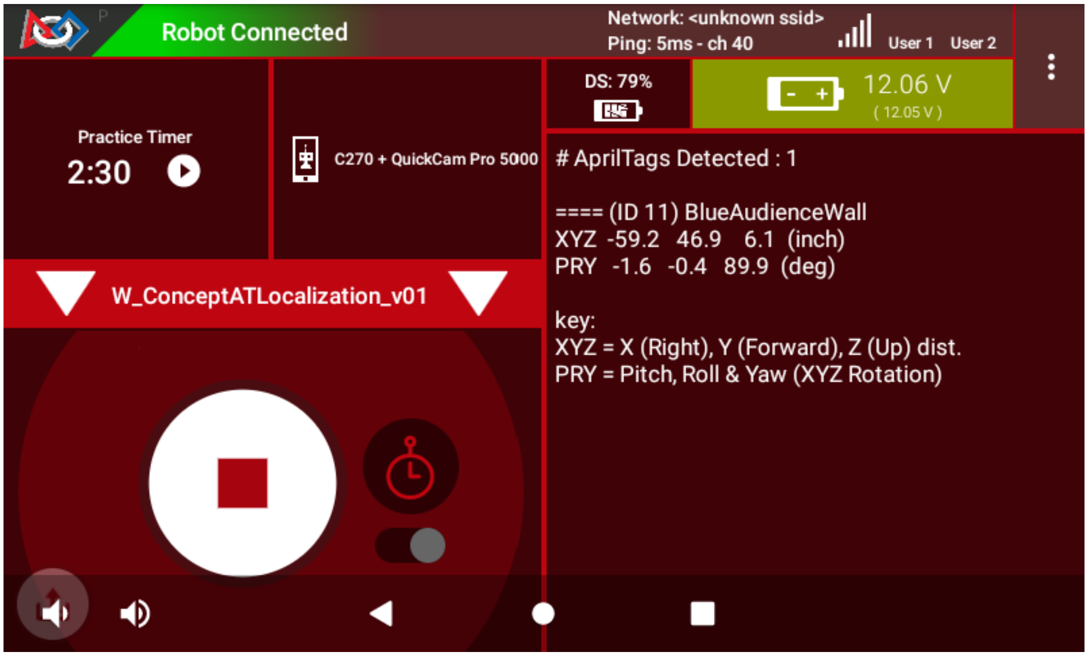


Fig. 93: Driver Station Sample Output

These details will be covered in a later section. In this example, the camera is 12 inches directly in front of AprilTag #11 from INTO THE DEEP.

Slowly move the camera around, keeping the AprilTag fully in the camera's view. The telemetry will update with the camera's location on the field.

It's working! Your OpMode can determine the **global pose** of the camera. A later section describes how to get the global **robot pose**, based on the camera's placement on the robot.

Skip the next two sections, if you already know how to use FTC previews.

DS Preview

Before describing the telemetry data, this page offers two sections on seeing the camera's view of the AprilTag with **previews**. Previewing is essential for working with robot vision.

On the Driver Station (DS), remain in INIT – don't touch the Start button.

At the top right corner, touch the 3-dots menu, then `Camera Stream`. This shows the camera's view; tap the image to refresh it.

For a BIG preview, touch the arrows at the bottom right corner.

Or, select Camera Stream again, to return to the previous screen and its Telemetry.

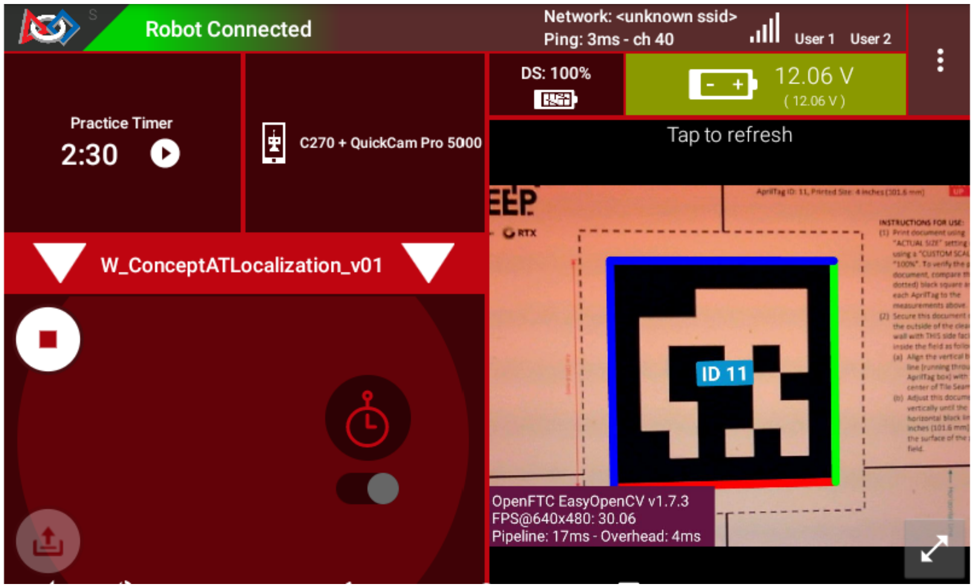


Fig. 94: Example of DS Camera Stream

RC Preview

The Robot Controller (RC) device also makes a preview, called `LiveView`. This is full video, and is shown automatically on the screen of an RC phone.

The above preview is from a REV Control Hub.

It has no physical screen, so you must plug in an HDMI monitor **or** use open-source scrcpy (called "screen copy") to see the preview on a laptop or computer that's connected via Wi-Fi to the Control Hub.

Basic Telemetry Data

Let's look closer at the DS telemetry:

In this example, the camera is 12 inches directly in front of AprilTag #11 from INTO THE DEEP.

The center of AprilTag #11 is at position X = -72 inches from the center of the field. This telemetry gives the camera's X position as (approximately) -60 inches, namely 12 inches in front of that tag.

The center of AprilTag #11 is at position Y = 48 inches from the center of the field. This telemetry gives the camera's Y position as (approximately) 48 inches, namely directly aligned (horizontally) with that tag.

The center of AprilTag #11 is at position Z = 5.9 inches (above the mat). This telemetry gives the camera's Z position as (approximately) 5.9 inches, namely directly aligned (vertically) with that tag.

The camera lens is parallel to the AprilTag, so the Pitch, Roll and Yaw values should be orthogonal (0 or a multiple of 90 degrees). This telemetry confirms the parallel orientation, with PRY values (approximately) 0 or 90 degrees.

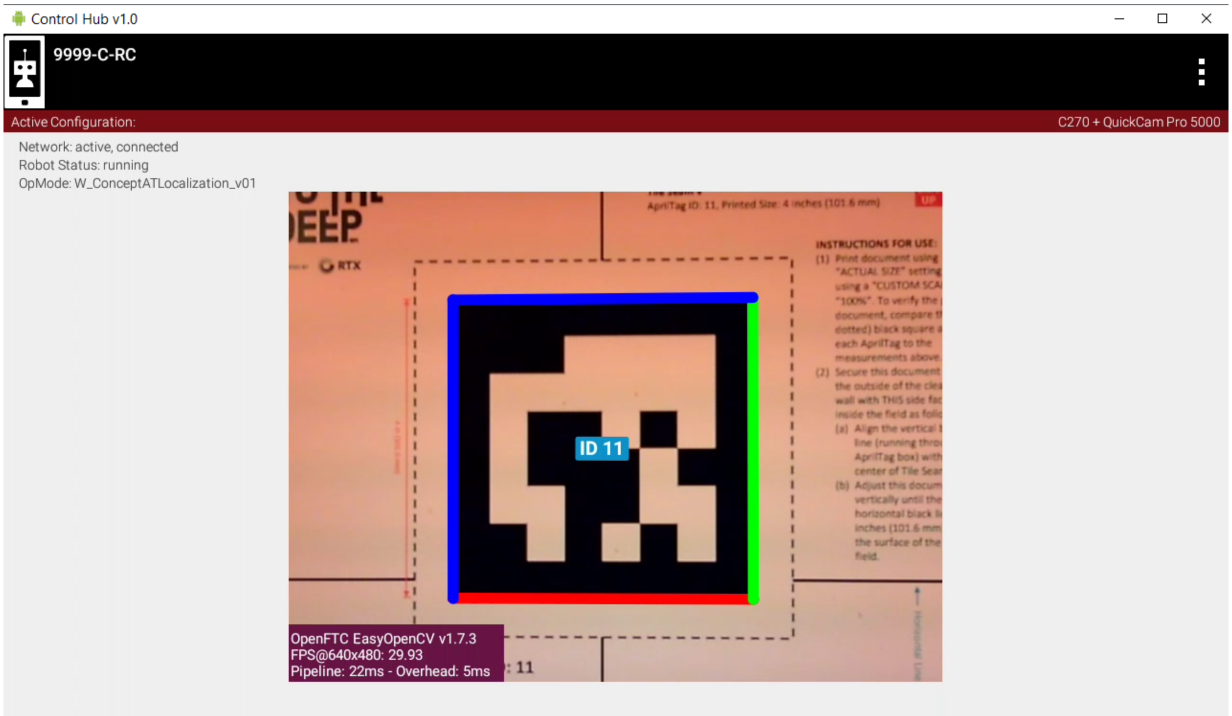


Fig. 95: Control Hub Preview

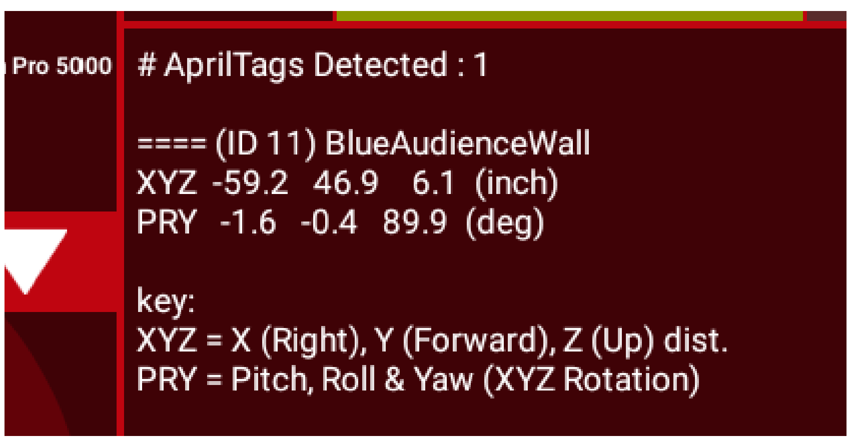


Fig. 96: DS Telemetry Example

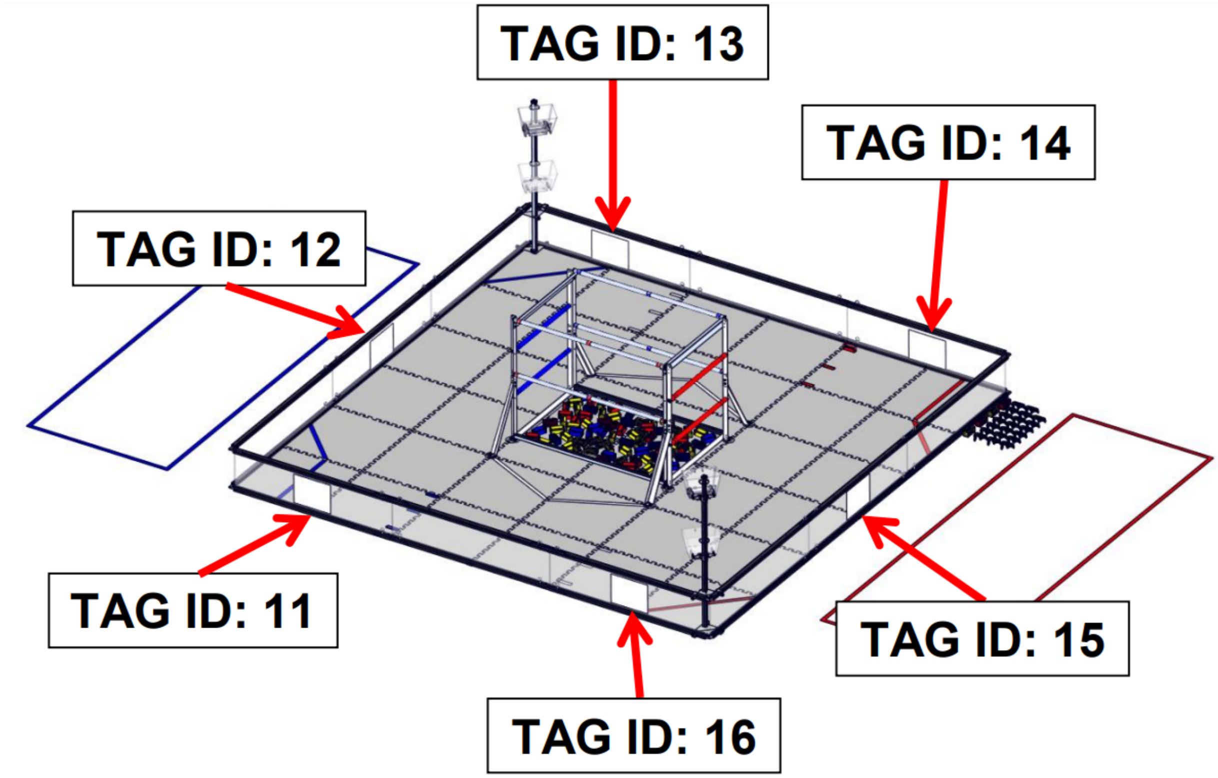


Fig. 97: Specific Tag Locations for INTO THE DEEP

Reference Frames

In the above example. the yaw angle is given as (approximately) -90 degrees. But the camera is facing in the negative X direction, thus has a heading or yaw angle of -180 degrees in the official FTC *field coordinate system* :

This sample OpMode uses a reference frame (coordinate system) that may be different than what you expect from other FTC navigation applications, including *IMU or robot axes* , odometry device axes, and the FTC field system (shown above). These differences typically result in basic and obvious changes in axis direction, axis swapping, and orthogonal angles (90-degree increments).

Learn and incorporate these differences into your OpMode, for the given scenario of your AprilTag localization. Manually adjust values as needed to accomplish your specific navigation goals.

Evaluate the accuracy and reliability of AprilTag navigation, with and without offsets, smoothing and other adjustments. Some FTC teams use multiple data sources for navigation. Base your robot strategy on capabilities **demonstrated** through extensive testing and refinement.

Camera Placement on Robot

The Sample OpMode provides fields to specify the location and orientation of the camera on the robot. The returned data will then represent the global **robot pose** rather than the camera's pose.

Subject to the reference frame caveat noted above, do your best to follow these commented instructions, in the Blocks and Java Sample OpModes:

> Setting these values requires a definition of the axes of the camera and robot:
>
> **Camera axes:**
>
> - *Origin location:* Center of the lens

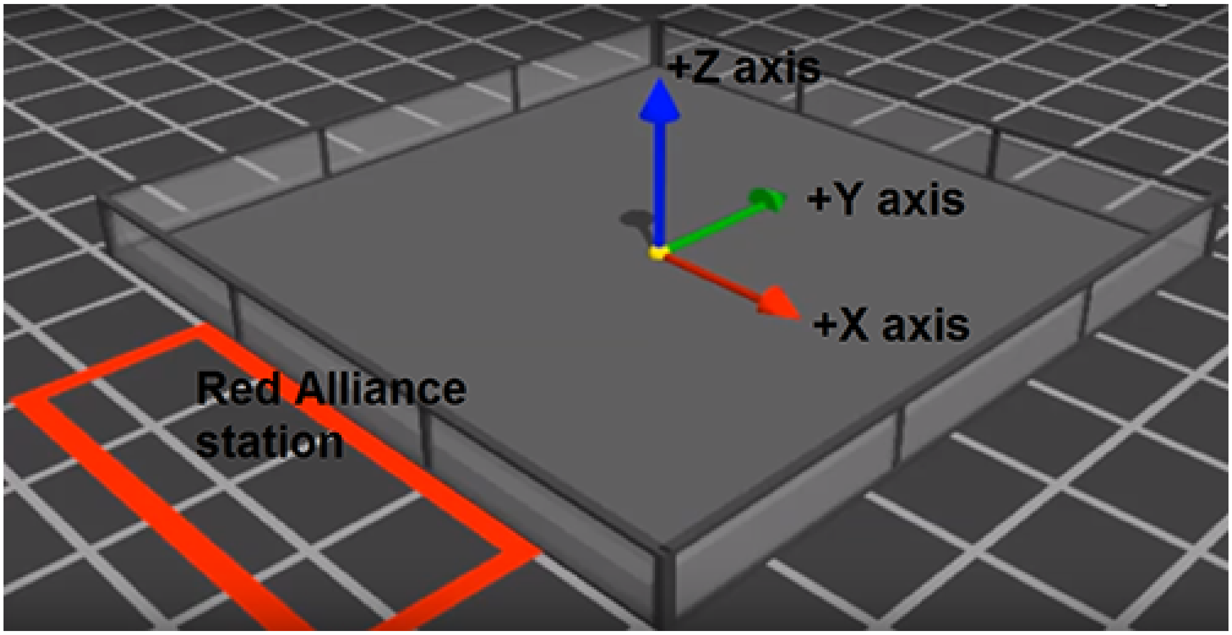


Fig. 98: FTC Field Coordinate System

- *Axes orientation:* +x right, +y down, +z forward (from camera's perspective)

Robot axes: (this is typical, but you can define this however you want)

- *Origin location:* Center of the robot at field height
- *Axes orientation:* +x right, +y forward, +z upward

Position:

- If all values are zero (no translation), that implies the camera is at the center of the robot. Suppose your camera is positioned 5 inches to the left, 7 inches forward, and 12 inches above the ground - you would need to set the position to (-5, 7, 12).

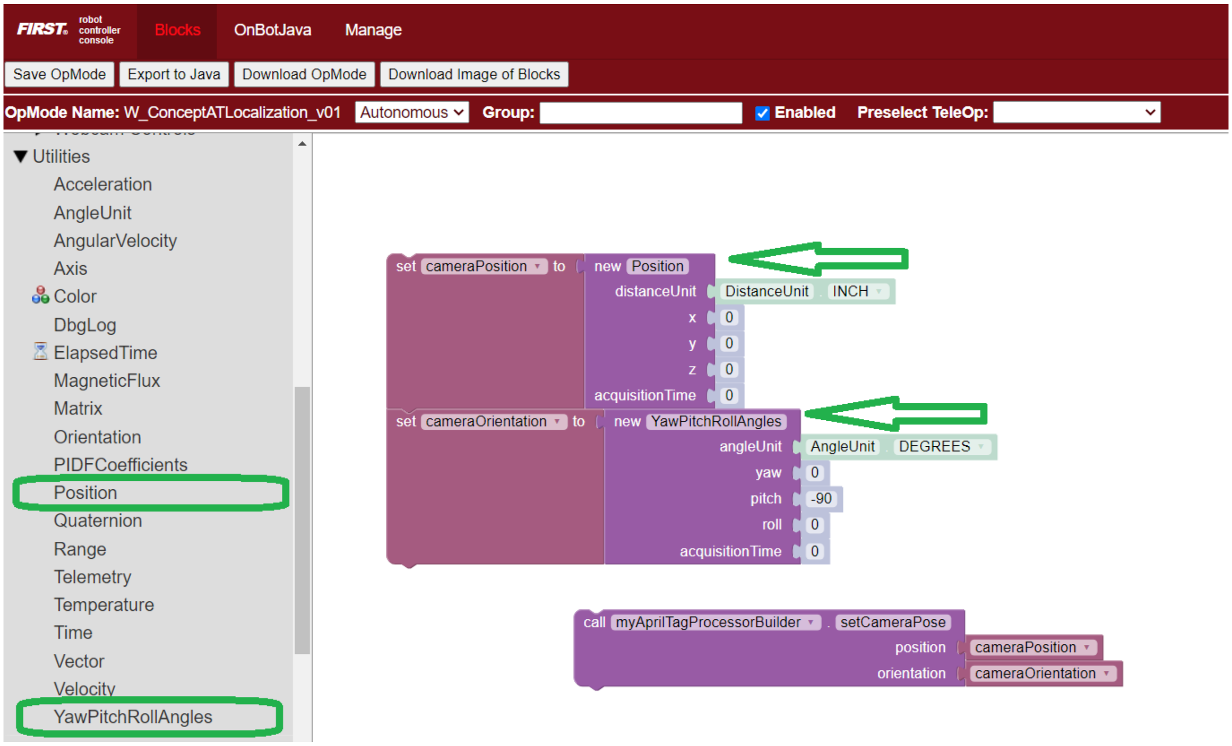
Orientation:

- If all values are zero (no rotation), that implies the camera is pointing straight up. In most cases, you'll need to set the pitch to -90 degrees (rotation about the x-axis), meaning the camera is horizontal. Use a yaw of 0 if the camera is pointing forwards, +90 degrees if it's pointing straight left, -90 degrees for straight right, etc. You can also set the roll to +/-90 degrees if it's vertical, or 180 degrees if it's upside-down.

To see the commands for setting **camera pose** (on the robot), select and read the Blocks **or** Java section below:

Blocks

The third Block called `.setCameraPose` can be found in the toolbox under Vision/AprilTag/AprilTagProcessor.Builder.



Java

These lines show that the camera placement on the robot becomes part of the AprilTag Processor, through the Java Builder pattern.

```
import org.firstinspires.ftc.robotcore.external.navigation.Position;
import org.firstinspires.ftc.robotcore.external.navigation.YawPitchRollAngles;
.
Position cameraPosition = new Position(DistanceUnit.INCH, 0, 0, 0, 0);
YawPitchRollAngles cameraOrientation = new YawPitchRollAngles(AngleUnit.DEGREES, 0, -90, 0, 0);
.
AprilTagProcessor aprilTag = new AprilTagProcessor.Builder()
      .setCameraPose(cameraPosition, cameraOrientation)
      .build();
```

Reading Global Pose

To see the commands for reading **global robot pose** data, select and read the Blocks **or** Java section below:

Blocks

These green Blocks can be assigned to position Variables, for later use.

These green Blocks can be assigned to orientation Variables, for later use.

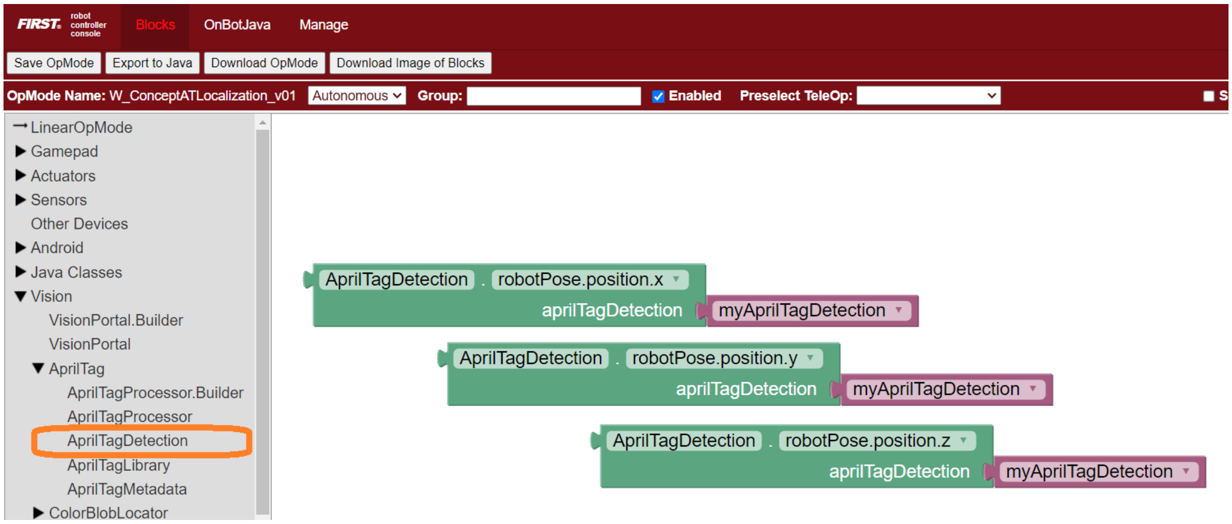


Fig. 99: Robot Position Blocks

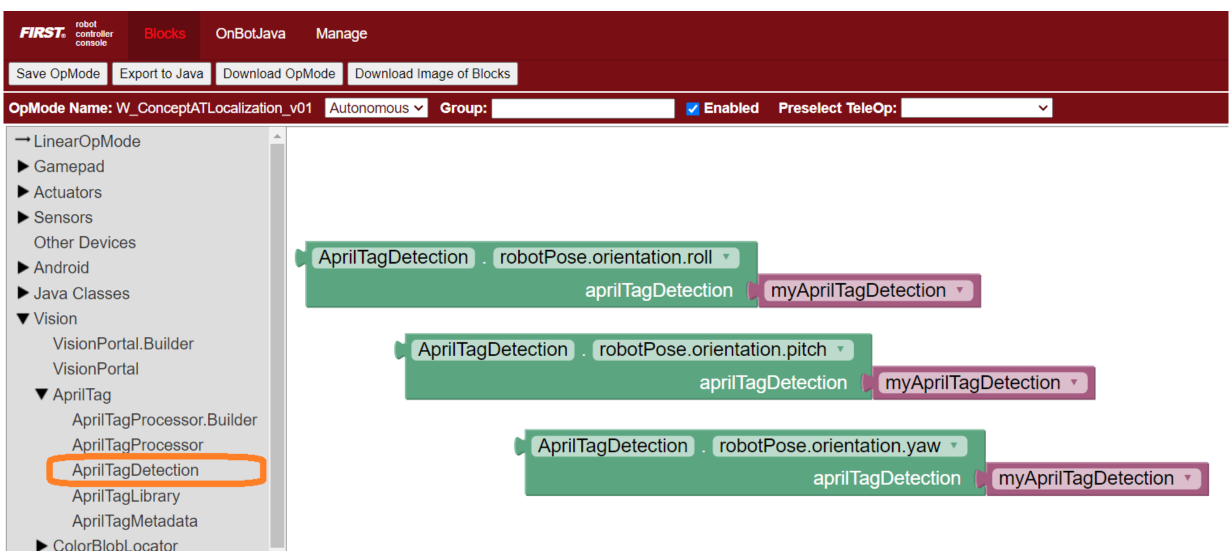


Fig. 100: Robot Orientation Blocks

Java

These lines demonstrate assigning position and orientation values to variables, for later use. These are typically "instant" values inside a `for` loop, as used in the Sample OpMode.

```
import org.firstinspires.ftc.vision.apriltag.AprilTagDetection;
.
AprilTagDetection detection;
.
double myX = detection.robotPose.getPosition().x;
double myY = detection.robotPose.getPosition().y;
double myZ = detection.robotPose.getPosition().z;
.
double myPitch = detection.robotPose.getOrientation().getPitch(AngleUnit.DEGREES);
double myRoll = detection.robotPose.getOrientation().getRoll(AngleUnit.DEGREES);
double myYaw = detection.robotPose.getOrientation().getYaw(AngleUnit.DEGREES);
```

Summary

The 2024 FTC software allows **robot localization** using a camera and fixed AprilTags on the field. This is done by combining three elements:

- basic AprilTag pose data,
- the tag's built-in metadata, and
- the camera's pose on the robot.

AprilTag localization uses a reference frame (coordinate system) that may differ from others, such as IMU or robot axes, odometry device axes, and the FTC field system. Adjust as needed.

Evaluate this navigation tool against other choices, and plan a robot strategy based on demonstrated capability.

Best of luck as you develop FTC robot navigation to reach your goals!

21.3.6 *FIRST* Tech Challenge AprilTag Testing Samples

Introduction

In the 2023-2024 season, FIRST Tech Challenge has introduced AprilTags into the season-unique competition. AprilTags were developed by the April Robotics Laboratory at the University of Michigan and are a visual fiducial tagging system, built on a similar concept as QR codes, useful for a wide variety of tasks including augmented reality, robotics, and camera calibration. A properly calibrated camera and tag library can be used to detect AprilTags and provide information such as range and orientation information (also known as **pose** data) about the tags with respect to the camera. The *FIRST* Tech Challenge Software Development Kit (SDK) has been updated to add AprilTag detection APIs to help teams make use of this resource.

This document contains examples of AprilTags that are intended to be used with the *FIRST* Tech Challenge AprilTag samples within the SDK. All AprilTags used in the 2023-2024 season are from the 36h11 tag family, which is a predetermined set of tags. The primary tag area is comprised of an 8x8 square matrix of black and white *pixels*. The size of the tag is measured based on the physical dimensions of the total black square portion of the tag – a 4 inch AprilTag has a black square portion that measures 4 inches on each side. Even though it is not used in measuring the size of an AprilTag, each tag also requires a white border one *pixel* thick surrounding the primary tag area (bringing the total tag size to 10x10 *pixels*). With the added white border, for example, a 4-inch AprilTag requires a footprint of 5 inches on each side.

The AprilTag API for *FIRST* Tech Challenge can handle multiple tag sizes; each individual tag can be sized independently, but there cannot be multiple sizes for an individual tag. Some pose information calculated for each tag, such as distance

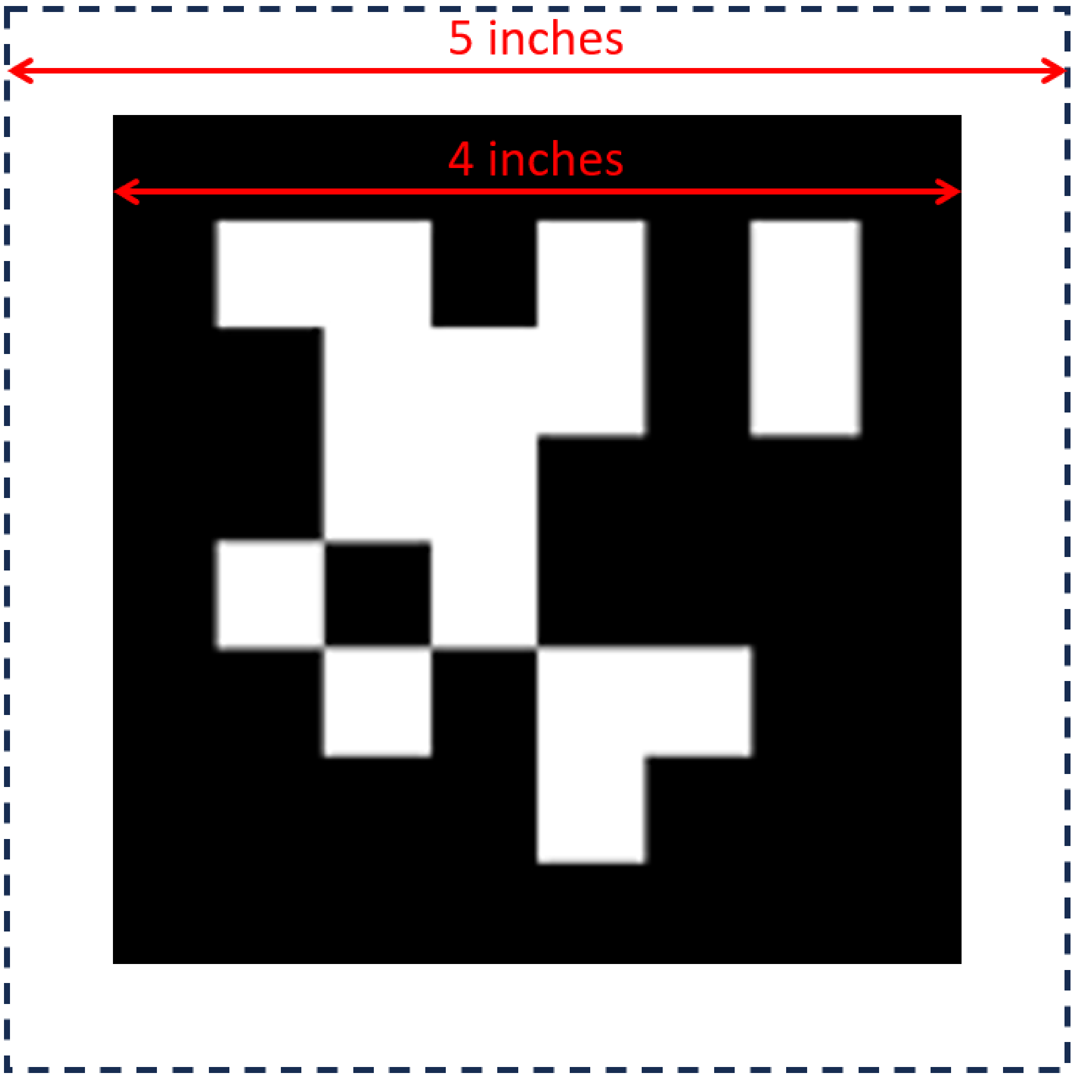


Fig. 101: Example sizing for 4-inch AprilTag

from camera to tag data, requires knowing the exact size of the tags being used. The default tag sizes used with the sample programs within the SDK are as follows:

Tag Description	Size of Tag in Inches (millimeters)
Tag ID: 583 (AKA "Nemo")	4 in (101.6 mm)
Tag ID: 584 (AKA "Jonah")	4 in (101.6 mm)
Tag ID: 585 (AKA "Cousteau")	6 in (152.4 mm)
Tag ID: 586 (AKA "Ariel")	6 in (152.4 mm)

When printing out the PDF `version of this document`, or portions thereof, please set the Page Size settings to "Actual Size" to ensure that the tags are printed properly. Every printer is slightly different, so it's also a good idea to measure the width and height of the black-square portion of the main tag area to verify that the page printed properly.

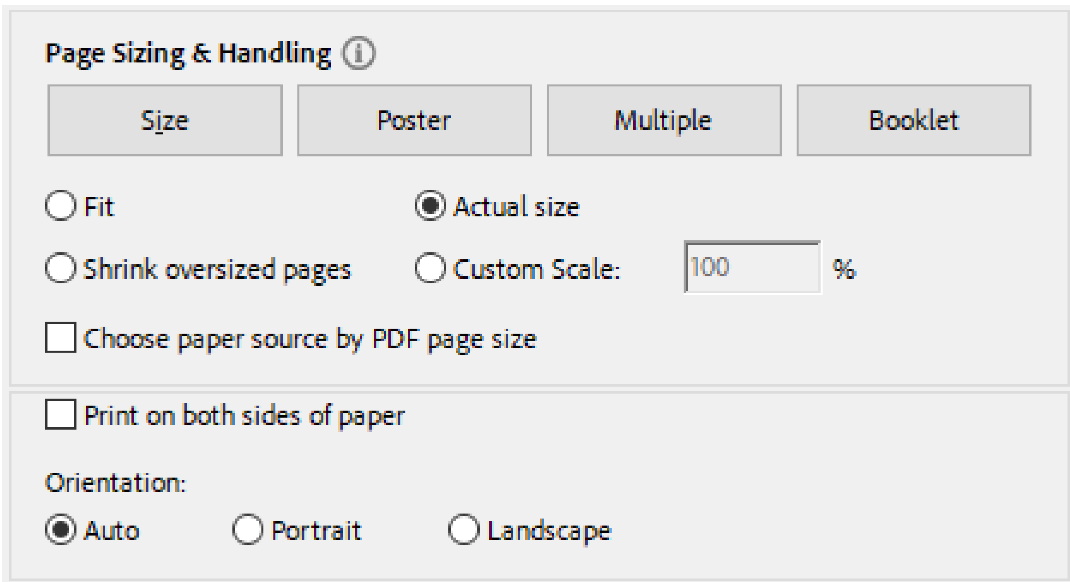


Fig. 102: Example printing settings for printing PDF at Actual Size

For more in-depth information about AprilTag detection values, and better understanding what they mean, please visit the following website:

Understanding AprilTag Detection Values - `Download and print the official PDF`

AprilTags

You can point your camera at these tags for recognition - ftc-docs does allow stretching of the image, so the image may not clearly and correctly be represented if the width of the display area is less than the width of the image. It is recommended to `download and print the official PDF`.

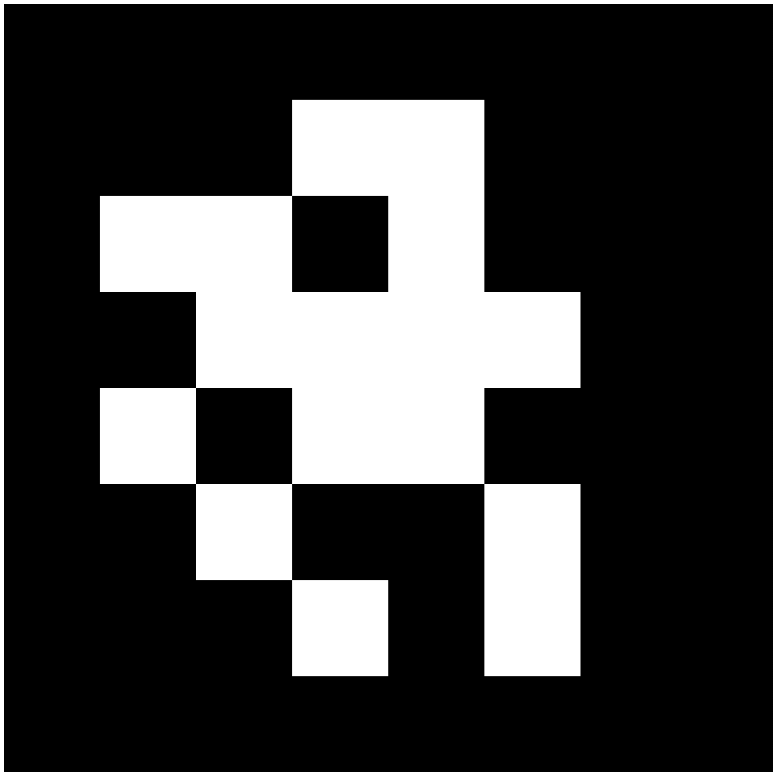


Fig. 103: Tag 583, "Nemo"

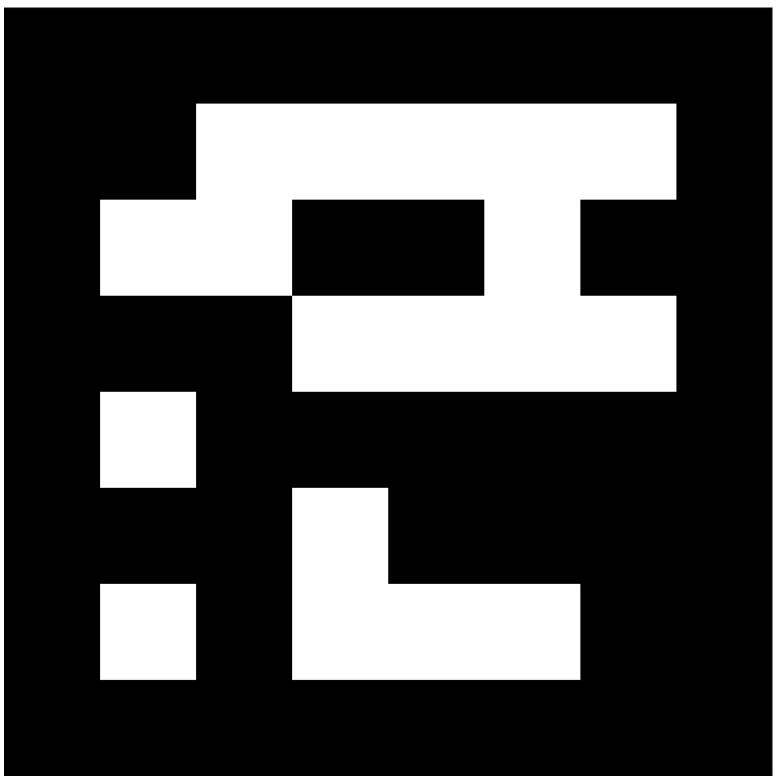


Fig. 104: Tag 584, "Jonah"

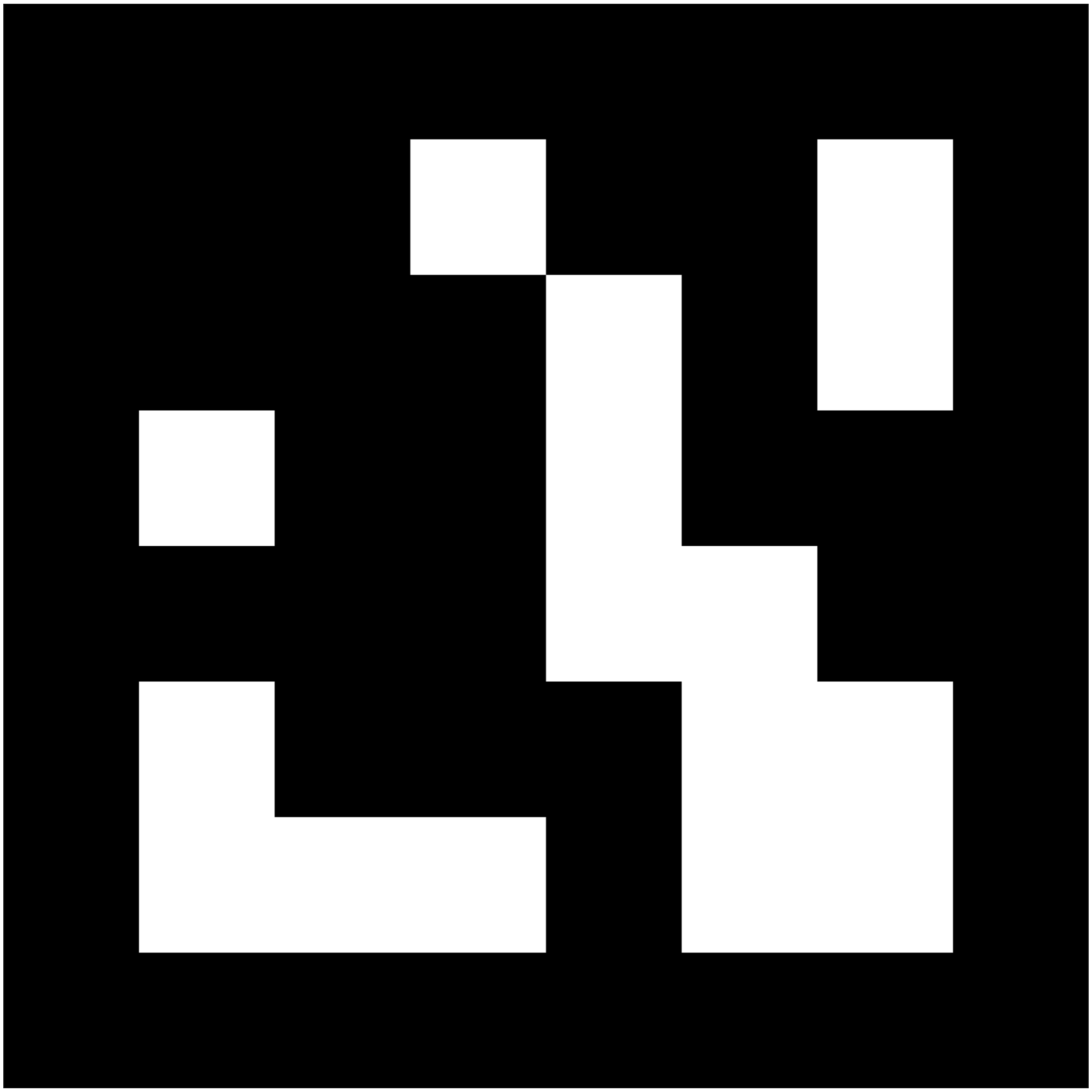


Fig. 105: Tag 585, “Cousteau”

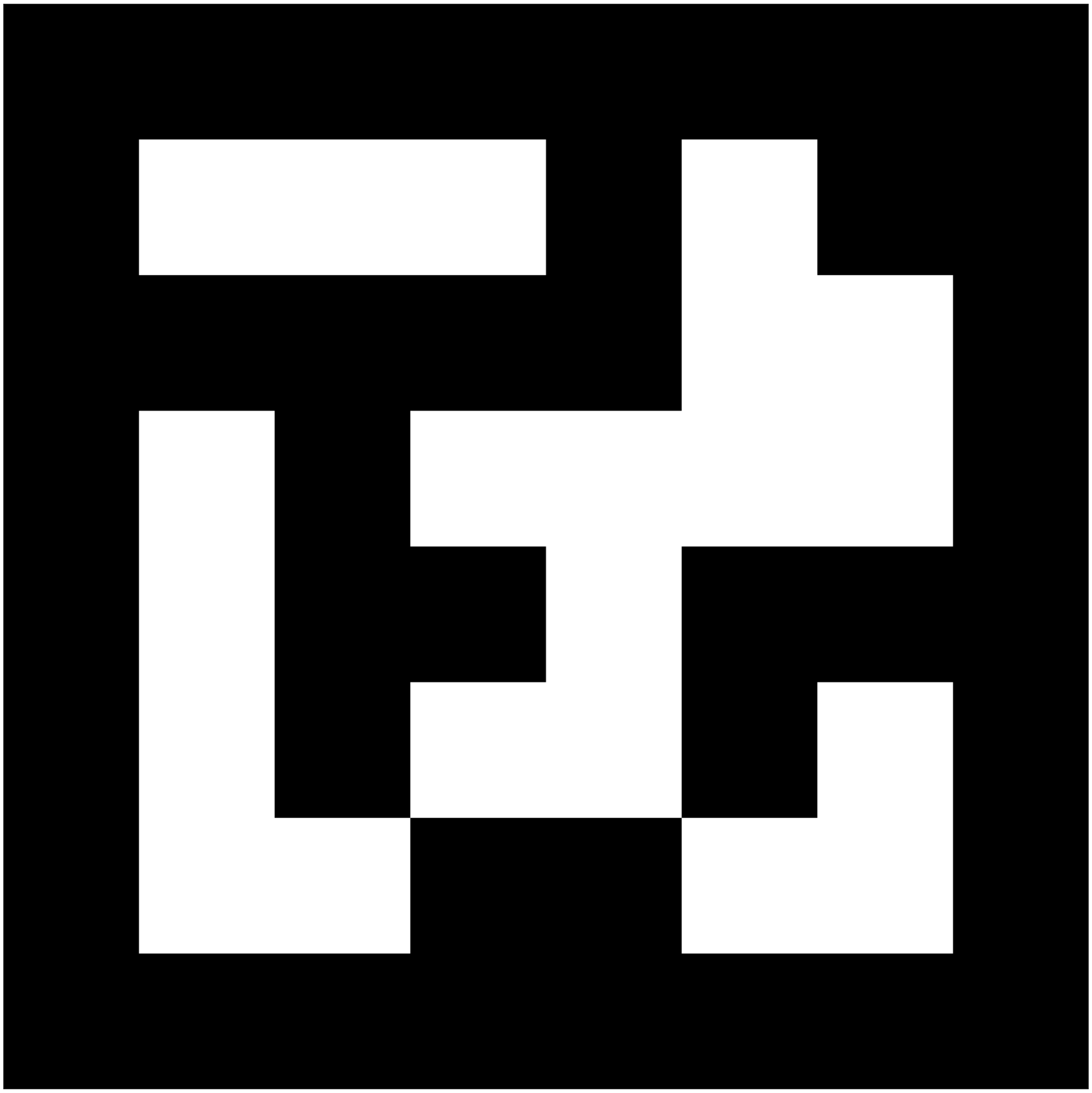


Fig. 106: Tag 586, “Ariel”

21.4 TensorFlow Programming

Topics for programming with TensorFlow Object Detection (TFOD)

21.4.1 TensorFlow for CENTERSTAGE presented by RTX

What is TensorFlow?

FIRST Tech Challenge teams can use TensorFlow Lite, a lightweight version of Google's TensorFlow machine learning technology that is designed to run on mobile devices such as an Android smartphone or the REV Control Hub. A *trained TensorFlow model* was developed to recognize the white `Pixel` game piece used in the **2023-2024 CENTERSTAGE presented by RTX** challenge.

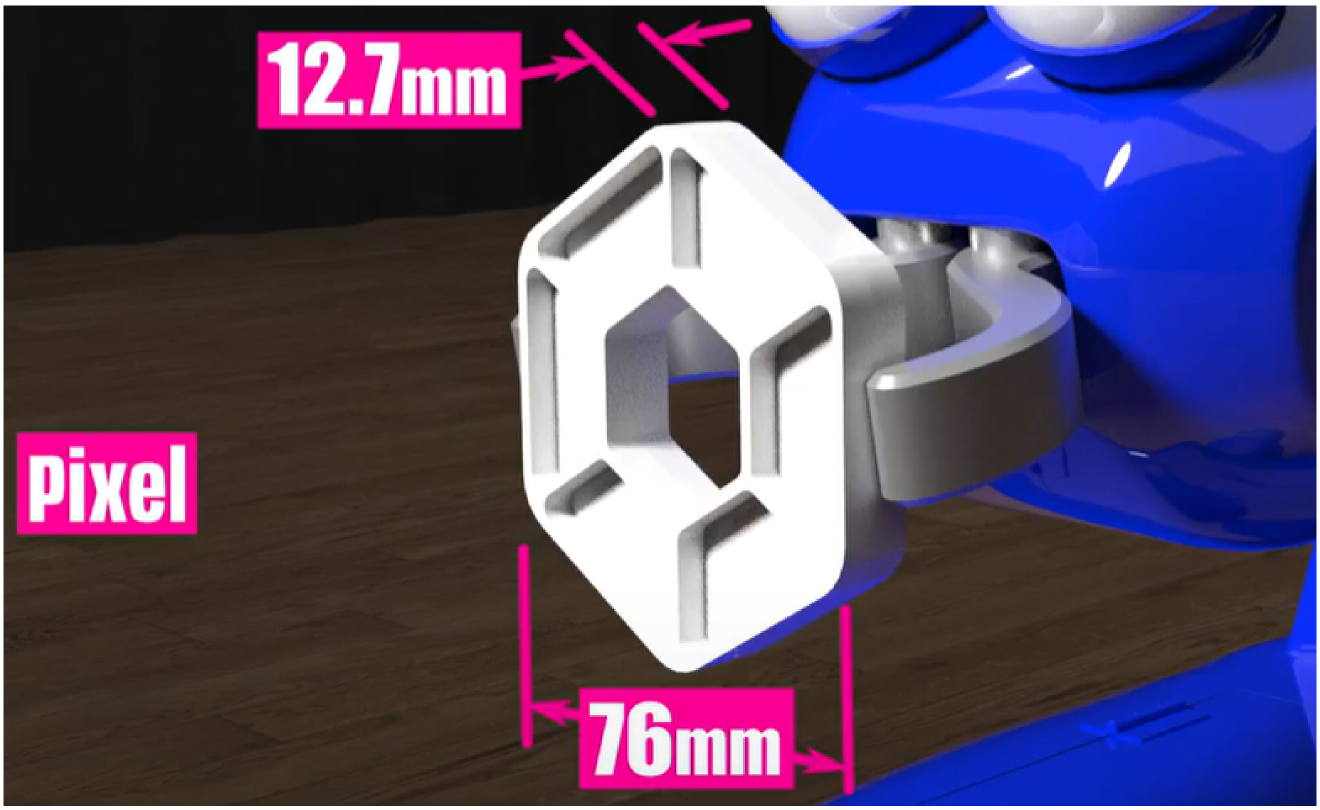


Fig. 107: This season's TFOD model can recognize a white Pixel

TensorFlow Object Detection (TFOD) has been integrated into the control system software to identify a white `Pixel` during a match. The SDK (SDK version 9.0) contains TFOD Sample OpModes and Detection Models that can recognize the white `Pixel` at various poses (but not all).

How Might a Team Use TensorFlow this season?

For this season's challenge the field is randomized during the Pre-Match stage. This randomization causes the white `Pixel` placed on Spike Marks to be placed on either the Left, Center, or Right Spike Mark. During Autonomous, Robots must independently determine which of the three Spike Marks (Left, Center, Right) the white `Pixel` was placed on. To do this, robots using a Webcam or a camera on a Robot Controller Smartphone can inspect Spike Mark locations to determine if a white `Pixel` is present. Once the robot has correctly identified which Spike Mark the white `Pixel` is present on, the robot can then perform additional actions based on that position that will yield additional points.

Teams also have the opportunity to replace the white `Pixel` with an object of their own creation, within a few guidelines specified in the Game Manual. This object, or Team Game Element, can be optimized to help the team identify it more easily and custom TensorFlow inference models can be created to facilitate recognition. As the field is randomized, the team's Team Game Element will be placed on the Spike Marks as the white `Pixel` would have, and the team must identify and use the Team Game Element the same as if it were a white `Pixel` on a Spike Mark.

Sample OpModes

Teams have the option of using a custom inference model with the *FIRST* Tech Challenge software or to use the game-specific default model provided. As noted above, the *FIRST* Machine Learning Toolchain is a streamlined tool for training your own TFOD models.

The FIRST Tech Challenge software (Robot Controller App and Android Studio Project) includes sample OpModes (Blocks and Java versions) that demonstrate how to use **the default inference model**. These tutorials show how to use the sample OpModes, using examples from previous *FIRST* Tech Challenge seasons, but demonstrate the process for use in any season.

- *Blocks Sample OpMode for TensorFlow Object Detection*
- *Java Sample OpMode for TFOD*

Using the sample OpModes, teams can practice identifying white `Pixels` placed on Spike Marks. The sample OpMode `ConceptTensorFlowObjectDetectionEasy` is a simple OpMode to use to detect a `Pixel` - it is a very basic OpMode simplified for beginner teams to perform basic `Pixel` detection.

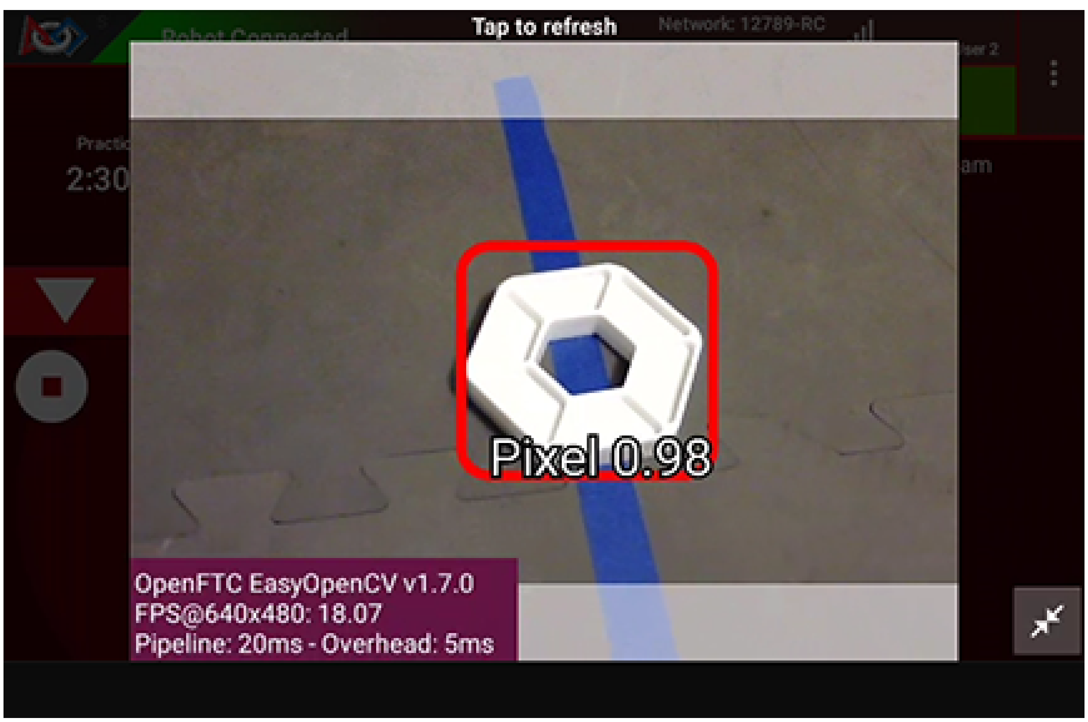


Fig. 108: Example Detection of a Pixel

It is important to note that if the detection of the object is below the minimum confidence threshold, the detection will not be shown - it is important to set the minimum detection threshold appropriately.

Note: The default minimum confidence threshold provided in the Sample OpMode (75%) is only provided as an example; depending on local conditions (lighting, image wear, etc...) it may be necessary to lower the minimum confidence in order to increase TensorFlow's likelihood to see all possible image detections. However, due to its simplified nature it is not possible to change the minimum confidence using the `Easy` OpMode. Instead, you will have to use the normal OpMode.

Notes on Training the CENTERSTAGE Model

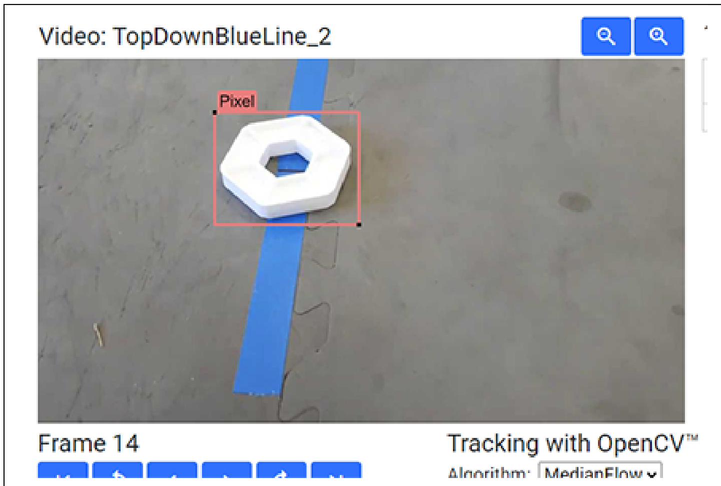
The `Pixel` game piece posed an interesting challenge for TensorFlow Object Detection (TFOD). As is warned in the Machine Learning Toolkit documentation, TFOD is not very good with recognizing and differentiating simple geometric shapes, nor distinguishing between specific colors; instead, TFOD is good at detecting *patterns*. TFOD needs to be able to recognize a unique *pattern*, and while there is a small amount of patterning in the ribbing of the `Pixel`, in various lighting conditions it's dubious how much the ribbing will be able to be seen. Even in the image at the top of this document, the ribbing can only be seen due to the specific shadows that the game piece has been provided. Even in optimal testing environments, it was difficult to capture video of the object that nicely highlighted the ribbing enough for TensorFlow to use for pattern recognition. This highlighted the inability to guarantee optimal `Pixel` characteristics in unknown lighting environments for TFOD.

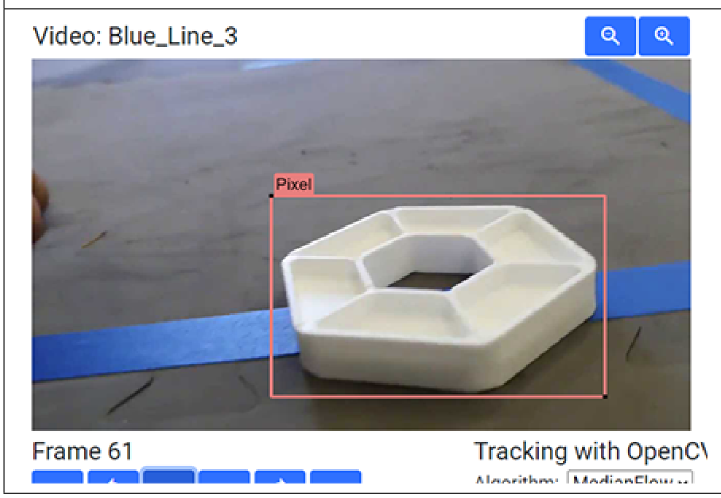
Another challenge with training the model had to do with how the `Pixel` looks at different pose angles. When the camera is merely a scant few inches from the floor, the `Pixel` can almost look like a solid object; at times there may be sufficient shadows to see that there is a hole in the center of the object, but not always. However, if the camera was several inches off the floor the `Pixel` looked differently, as the mat or colored tape could be seen through the hole in the middle of the object. This confused the neural network and made it extremely difficult to train, and the resulting models eventually recognized any "sufficiently light colored blob" as a `Pixel`. This was not exactly ideal.

Even with the best of images, the Machine Learning algorithms had a difficult time determining what *was* a `Pixel` and what wasn't. What ended up working was providing NOT ONLY images of the `Pixel` in different poses, but also several white objects that WERE NOT a `Pixel`. This was fundamental to helping TensorFlow train itself to understand that "All `Pixels` are White Objects, but not all White Objects are `Pixels`."

To provide some additional context on this, here are a few examples of labeled frames that illustrate the challenges and techniques in dealing with the `Pixel` game piece.

Table 7: Examples of Challenging Scenarios





Using the Default CENTERSTAGE Model

In the previous section it's described how the height of the camera from the floor has a huge effect on how the `Pixel` is seen; too low and the object can look like a single "blob" of color, and too high and the object will look similar to a white donut. When training the model, it was decided that the Donut approach was the best - train the model to recognize the `Pixel` from above to provide a clear and consistent view of the `Pixel`. Toss in some angled shots as well, along with some additional extra objects just to give TensorFlow some perspective, and a model is born. **But wait, how does that affect detection of the Pixel from the robot's starting configuration?**

In CENTERSTAGE, using the default CENTERSTAGE model, it is unlikely that a robot will be able to get a consistent detection of a White `Pixel` from the starting location. In order to get a good detection, the robot's camera needs to be placed fairly high up, and angled down to be able to see the gray tile, blue tape, or red tape peeking out of the center of the `Pixel`. Thanks to the center structure on the field this season, it's doubtful that a team will want to have an exceptionally tall robot - likely no more than 14 inches tall, but most will want to be under 12 inches to be safe (depending on your strategy - please don't let this article define your game strategy!). The angle that your robot's camera will have with the Pixel in the starting configuration makes this seem unlikely.

Here are several images of detected and non-detected `Pixels`. Notice that the center of the object must be able to see through to what's under the `Pixel` in order for the object to be detected as a `Pixel`.

Table 8: Examples of Detected and Non-Detected Pixels

Therefore, there are two options for detecting the `Pixel`:

1. The camera can be on a retractable/moving system, so that the camera is elevated to a desirable height during the start of Autonomous, and then retracts before moving around.
2. The robot will have to drive closer to the Spike Marks in order to be able to properly detect the `Pixels`.

For the second option (driving closer), the camera's field of view might pose a challenge if it's desirable for all three Spike Marks to be always in view. If using a Logitech C270 camera, perhaps using a Logitech C920 with a wider field of view might help to some degree. This completely depends on the height of the camera and how far the robot must be driven in order to properly recognize a `Pixel`. Teams can also simply choose to point their webcam to the CENTER and LEFT Spike Marks, for example, and drive closer to those targets, and if a `Pixel` is not detected then by process of elimination it must be on the RIGHT Spike Mark.

Selecting objects for the Team Prop

Selecting objects to use for your custom Team Prop can seem daunting. Questions swirl like "What shapes are going to be recognized best?", "If I cannot have multiple colors, how do I make patterns?", and "How do I make this easier on myself?". Hopefully this section will help you understand a little more about TensorFlow and how to get the most out of it.

First, it's important to note that TensorFlow has the following quirks/behaviors:

- In order to run TensorFlow on mobile phones, *FIRST* Tech Challenge uses a very small core model resolution. This means the image is downscaled from the high definition webcam image to one that is only 300x300 pixels. This means that medium and small objects within the webcam images may be reduced to very small indistinguishable clusters of pixels in the target image. Keep the objects in the view of the camera large, and train for a wide range of image sizes.
- TensorFlow is not really good at differentiating simple geometric shapes. TensorFlow Object Detection is an object classifier, and similar geometric shapes will classify similarly. Humans are much better at differentiating geometric shapes than neural net algorithms, like TensorFlow, at the present.
- TensorFlow is great at pattern detection, but that means that within the footprint of the object you need one or more repeating or unique patterns. The larger the pattern the easier it will be for TensorFlow to detect the pattern at a distance.

So what kinds of patterns are good for TensorFlow? Let's explore a few examples:

1. Consider the shape of a chess board Rook. The Rook itself is mostly uniform all around, no matter how you rotate the object it more or less looks the same. Not much patterning there. However, the top of the Rook is very unique and patterned. Exaggerating the "battlements", the square-shaped parts of the top of the Rook, can provide unique patterning that TensorFlow can distinguish.
2. Consider the outline of a chess Knight, as the "head" of the Knight is facing to the right or to the left. That profile is very distinguishable as the head of a horse. That specific animal is one that model zoos have been optimized for, so it's definitely a shape that TensorFlow can be trained to recognize.
3. Consider the patterning in a fancy wrought-iron fence. If made thick enough, those repeating patterns can be recognized by a TensorFlow model. Like the Chess Board Rook, it might be wise to make the object round so that the pattern is similar and repeats now matter how the object is rotated. If allowed, having multiple shades of color can also help make a more-unique patterning on the object (e.g. multiple shades of red, likely must consult the Q&A).
4. TensorFlow can be used to Detect Plants and all of the plants are a single color. Similar techniques can be reverse-engineered (make objects of different "patterns" similar to plants) to create an object that can be detected and differentiated from other objects on the game field.

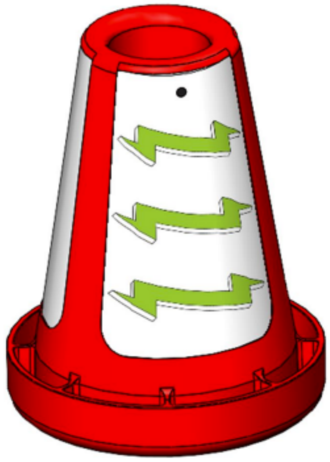
Hopefully this gives you quite a few ideas for how to approach this challenge!

21.4.2 TensorFlow for POWERPLAY presented by Raytheon Technologies

What is TensorFlow?

FIRST Tech Challenge teams can use TensorFlow Lite, a lightweight version of Google's TensorFlow machine learning technology that is designed to run on mobile devices such as an Android smartphone. A *trained TensorFlow model* was developed to recognize the three game-defined images on the Signal element used in the **2022-2023 POWERPLAY presented by Raytheon Technologies** challenge.

TensorFlow Object Detection (TFOD) has been integrated into the control system software to identify these Signal images during a match. The SDK (SDK version 8.0) contains TFOD Sample Op Modes and Detection Models that can recognize and differentiate between the Signal images: Bolt (green lightning bolt), Bulb (4 yellow light bulbs), and Panel (purple solar panels).



E-1 – *Signal*

Fig. 109: This season's TFOD model can recognize Signal image elements

Note: TensorFlow Lite runs on Android 6.0 (Marshmallow) or higher, a requirement met by all currently allowed devices. If you are a Blocks programmer using an older/disallowed Android device that is not running Marshmallow or higher, TFOD Blocks will automatically be missing from the Blocks toolbox or design palette.

How Might a Team Use TensorFlow this season?

For this season's challenge, during the pre-Match stage a single die is rolled and the field is randomized. The random value of the die determines how field reset staff will rotate the Signal to show one of the three images on the Signal to the robot - Signal images are offset 120 degrees on the Signal to occlude all images other than the chosen one. Robots must independently determine which of the three images (Image 1, Image 2, or Image 3, indicated by the number of dots above the image either on the Signal stickers or on the Team Specific Signal Sleeve) is showing. Once the robot has correctly identified the Image being shown, the robot can then know in which zone to end the Autonomous Period for additional points.

Sample Op Modes

Teams have the option of using a custom inference model with the *FIRST* Tech Challenge software or to use the game-specific default model provided. As noted above, the *FIRST* Machine Learning Toolchain is a streamlined tool for training your own TFOD models.

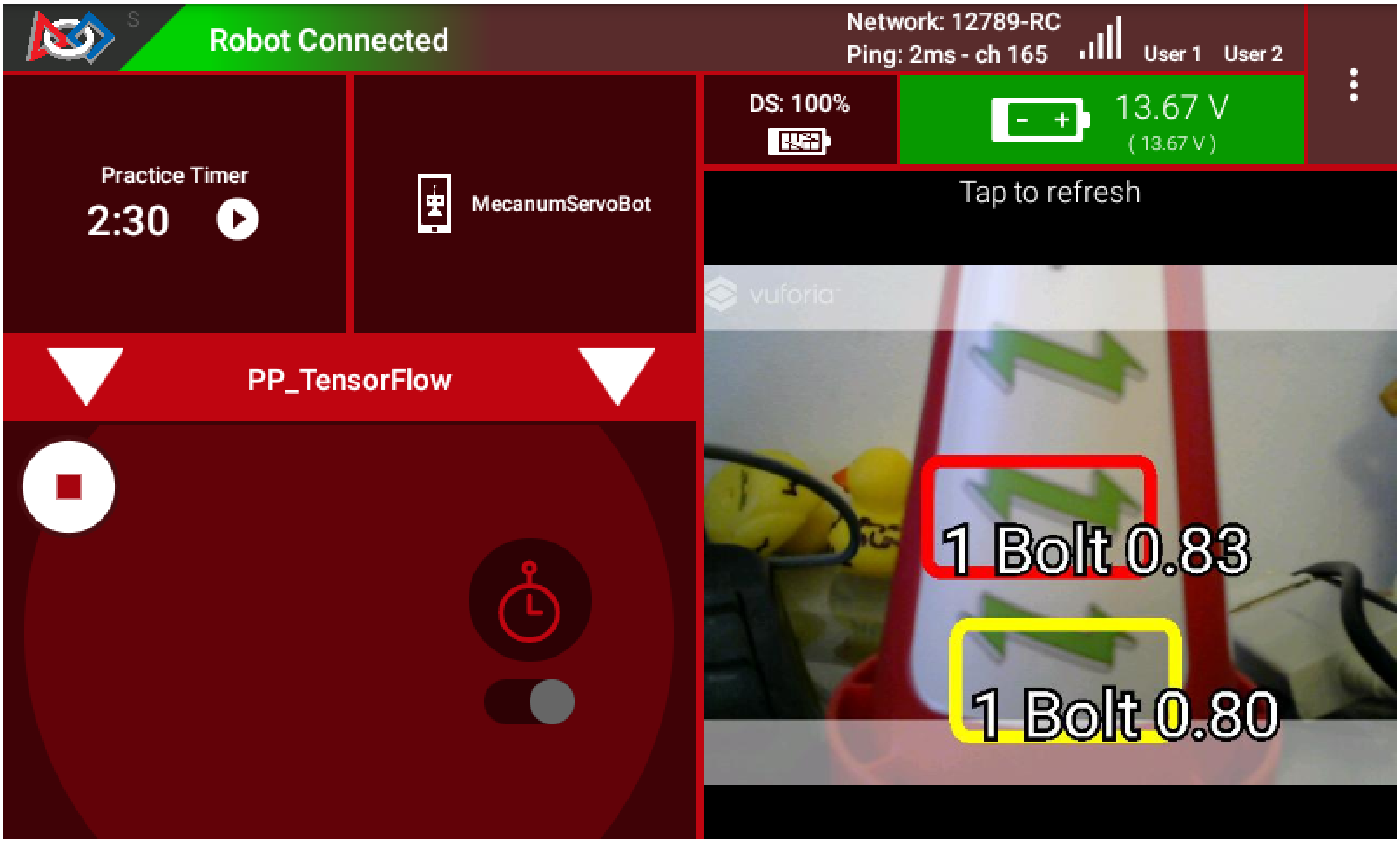
The FIRST Tech Challenge software (Robot Controller App and Android Studio Project) includes sample op modes (Blocks and Java versions) that demonstrate how to use **the default inference model**. These tutorials show how to use the sample op modes, using examples from previous *FIRST* Tech Challenge seasons, but demonstrate the process for use in any season.

- *Blocks Sample Op Mode for TensorFlow Object Detection*

- *Java Sample Op Mode for TFOD*

Using the sample Op Modes, each of the Signal images can be detected. Here are a few examples of detecting the images.

Example Image 1



Example Detection of a Bolt

Example Image 2

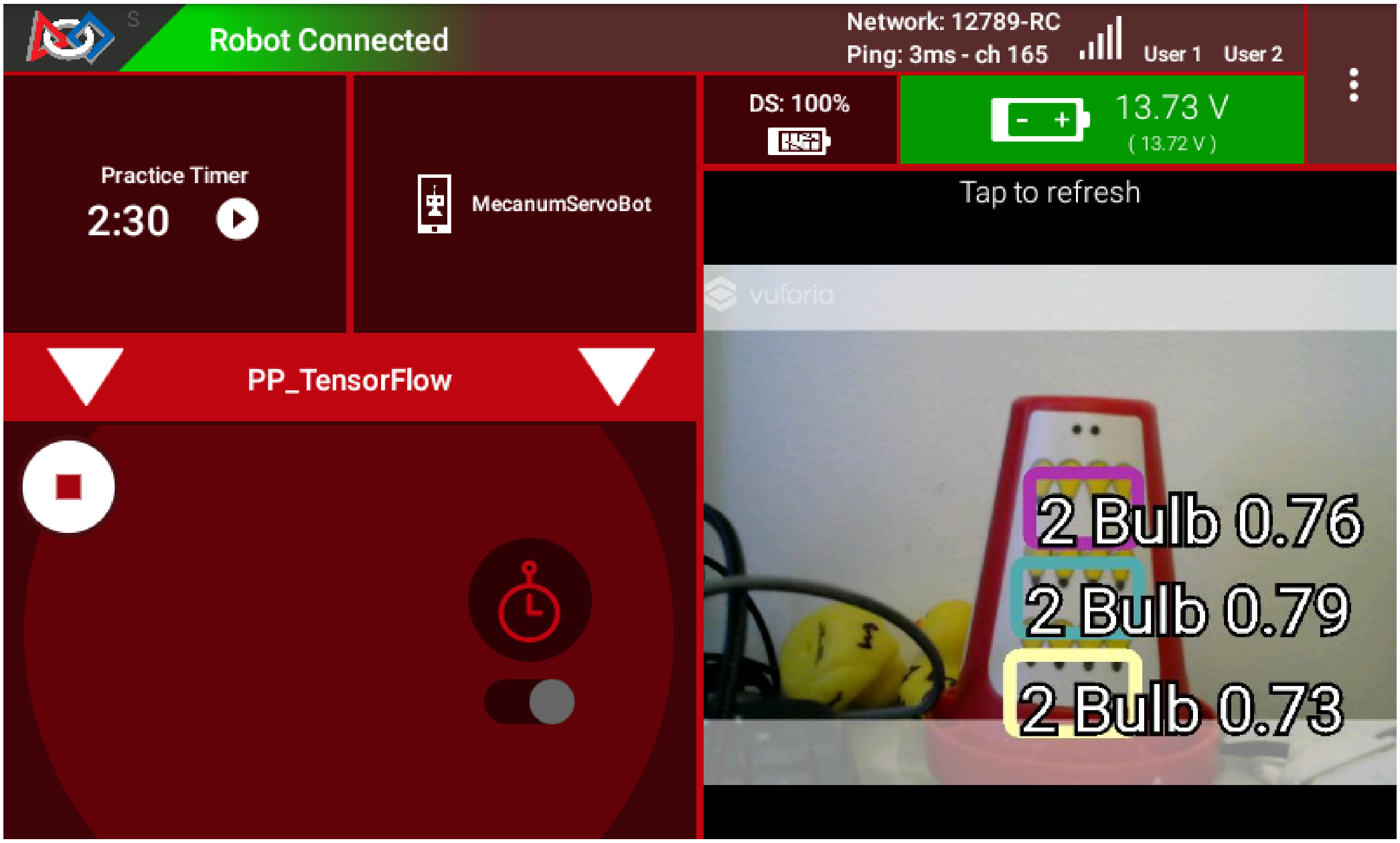
Example Detection of a Bulb

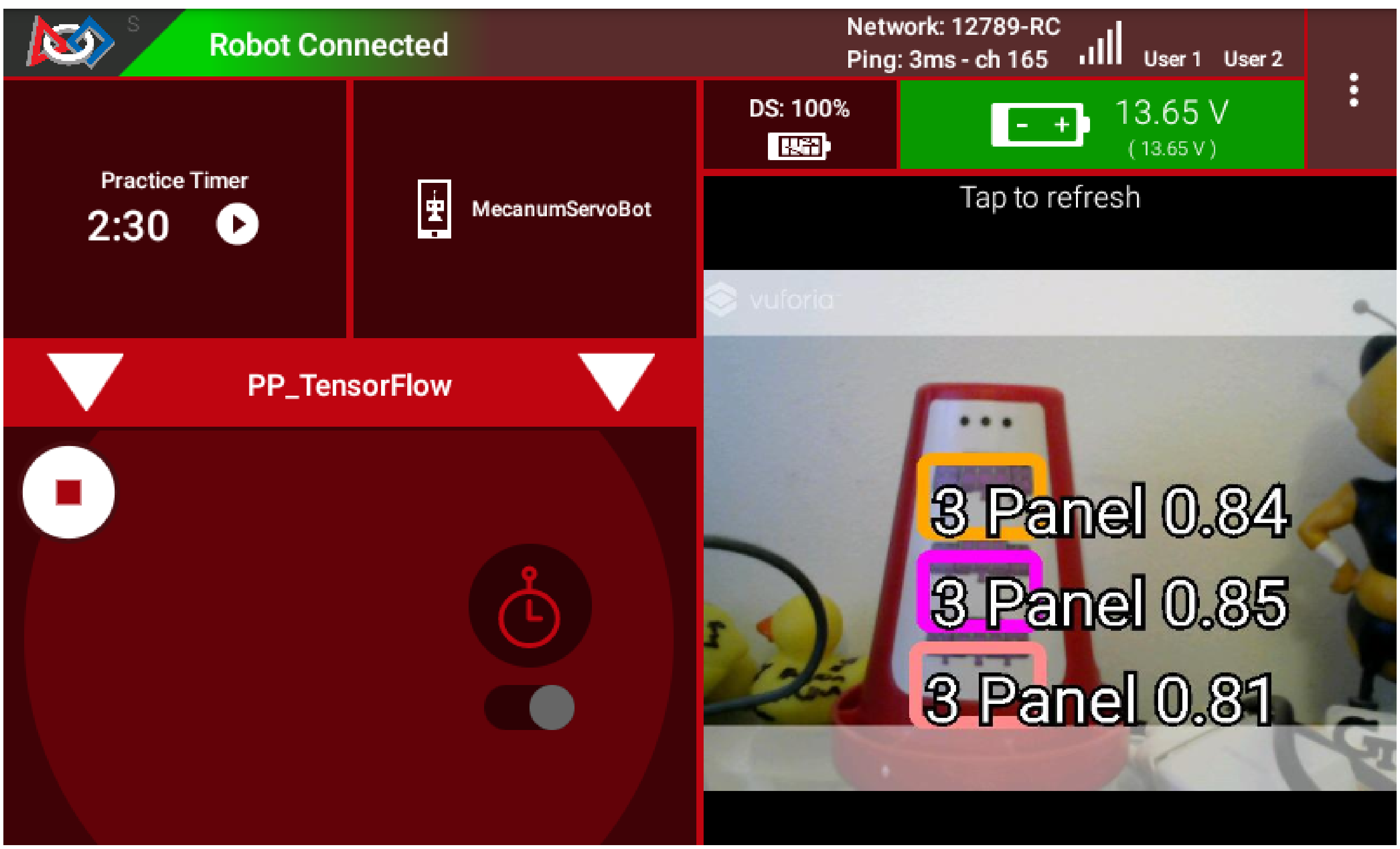
Example Image 3

Example Detection of a Panel

It is important to note that if the detection of the object is below the minimum confidence threshold, the detection will not be shown - it is important to set the minimum detection threshold appropriately.

Note: The default minimum confidence threshold provided in the Sample Op Mode is only provided as an example; depending on local conditions (lighting, image wear, etc…) it may be necessary to lower the minimum confidence in order to increase TensorFlow's likelihood to see all possible image detections.


Robot Connected
Network: 12789-RC
Ping: 3ms - ch 165
User 1
User 2
DS: 100%
13.73 V
(13.72 V)
Practice Timer
2:30
MecanumServoBot
Tap to refresh
PP_TensorFlow
2 Bulb 0.76
2 Bulb 0.79
2 Bulb 0.73


Robot Connected
Network: 12789-RC
Ping: 3ms - ch 165
User 1
User 2
DS: 100%
13.65 V
(13.65 V)
Practice Timer
2:30
MecanumServoBot
Tap to refresh
PP_TensorFlow
3 Panel 0.84
3 Panel 0.85
3 Panel 0.81

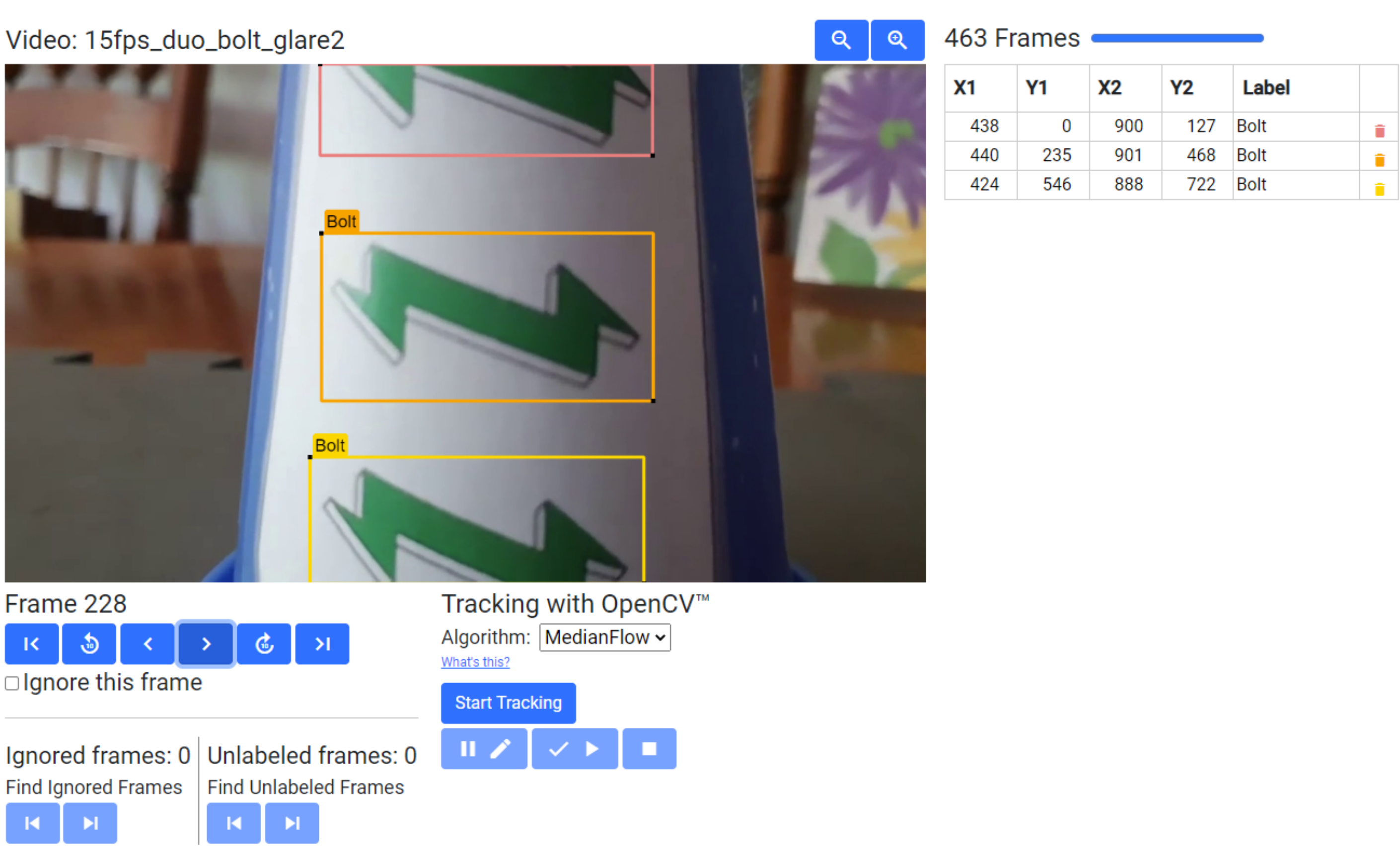
Default POWERPLAY Model Detection Notes

As shown in the previous examples, with the default POWERPLAY TensorFlow model it is sometimes more common for TensorFlow to recognize/label partial image areas (upper or lower portions of the images) than whole images themselves. This is likely due to how the training set was developed during training of the TensorFlow model.

In order to try to ensure that there would be as many detections for a given set of images as possible, the training set included frames that contained both complete and partial images; it just happened that the way the frames were developed there were more upper and lower partial images than whole images, and it appears that TensorFlow's neural network seems to almost "prefer" to recognize partial images rather than whole images. Such biases are common.

To provide some additional context on this, here are a few examples of labeled frames that were used to train the default TensorFlow model.

Example Training Frame 1



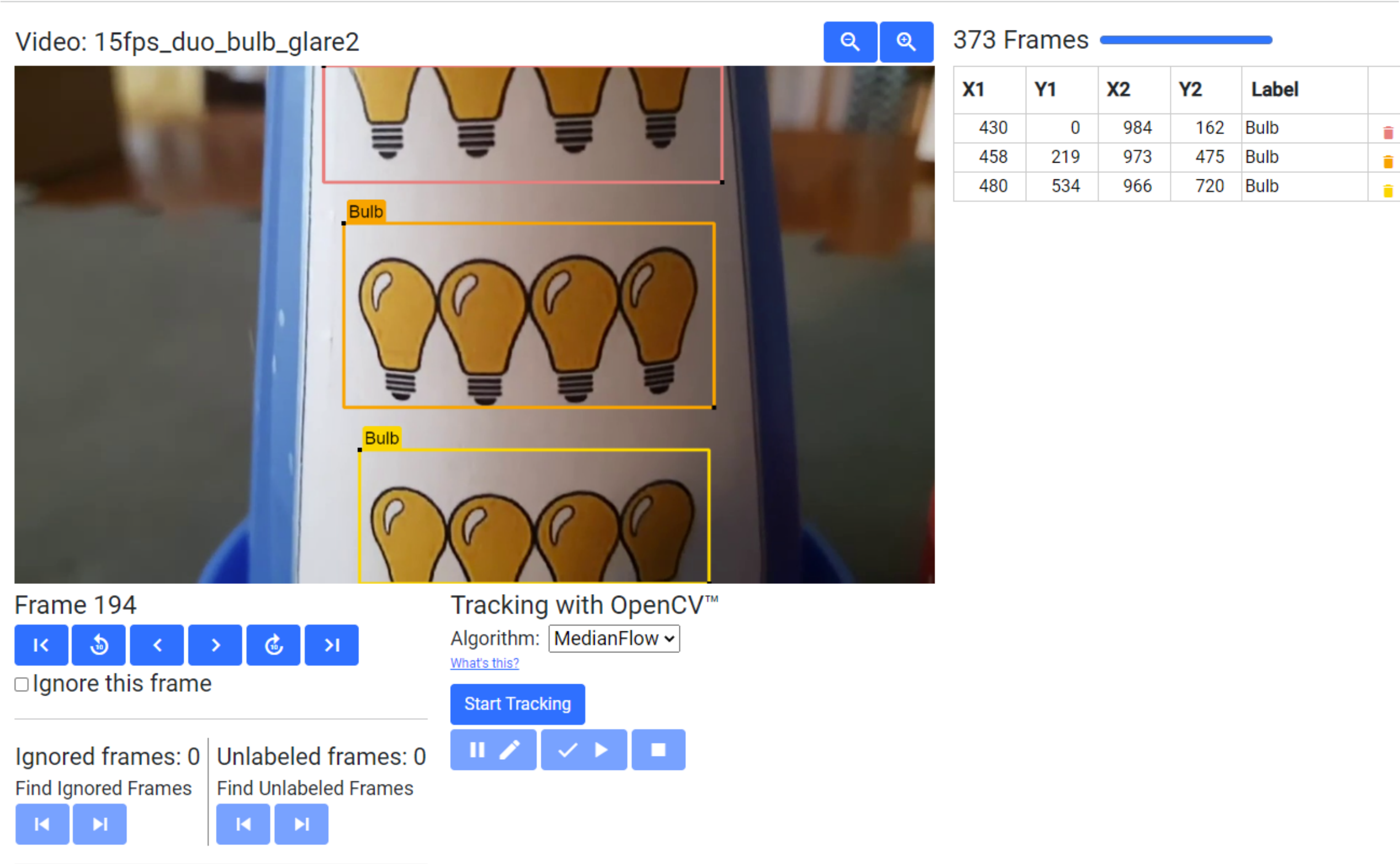
Example Training for a Bolt

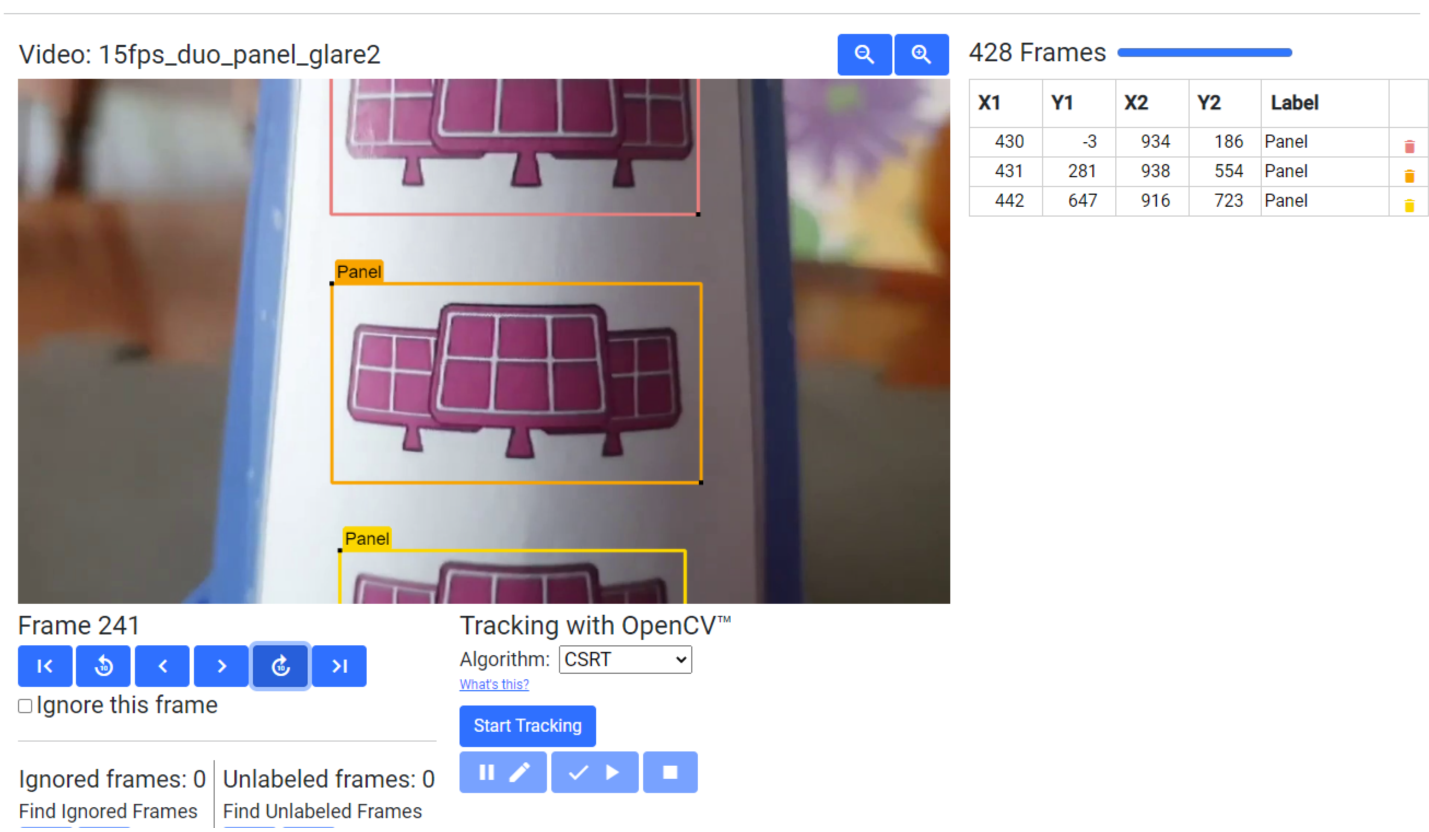
Example Training Frame 2

Example Training for a Bulb

Example Training Frame 3

Example Training for a Panel


Video: 15fps_duo_bulb_glare2
373 Frames
X1 Y1 X2 Y2 Label
430 0 984 162 Bulb
458 219 973 475 Bulb
480 534 966 720 Bulb
Bulb
Bulb
Frame 194
Ignore this frame
Tracking with OpenCV™
Algorithm: MedianFlow
What's this?
Start Tracking
Ignored frames: 0
Find Ignored Frames
Unlabeled frames: 0
Find Unlabeled Frames


Video: 15fps_duo_panel_glare2
428 Frames
X1 Y1 X2 Y2 Label
430 -3 934 186 Panel
431 281 938 554 Panel
442 647 916 723 Panel
Panel
Panel
Frame 241
Ignore this frame
Tracking with OpenCV™
Algorithm: CSRT
What's this?
Start Tracking
Ignored frames: 0
Find Ignored Frames
Unlabeled frames: 0
Find Unlabeled Frames

Understanding Backgrounds For Signal Sleeves

When thinking about how to develop a custom Signal Sleeve, it's easy to overlook one of the most important elements that may make or break your ability to detect objects - your image background. TensorFlow attempts to identify common background material and "ignore" the backgrounds for detecting labeled objects; a great example of this is the white background on the sticker. It should be known that the white background on the stickers posed quite a challenge, one that teams should be aware of when/if attempting to develop their own images for their Signal Sleeves.

If the same background is always present, and always has similar characteristics in the training data, TensorFlow may assume the background isn't actually a background and is really a part of the image. TensorFlow may then expect to see the specific background with the objects always. If the background of the image then varies for whatever reason, TensorFlow may not recognize the image with the new background.

A great example of this occurred in 2021 Freight Frenzy; the duck model was trained to recognize a rubber duck, and the rubber duck just happened to always be present on a gray mat tile within the training frames. The model happened to "expect" a gray mat tile in the background, and rubber ducks seen without the gray mat tile had a significantly reduced detection rate.

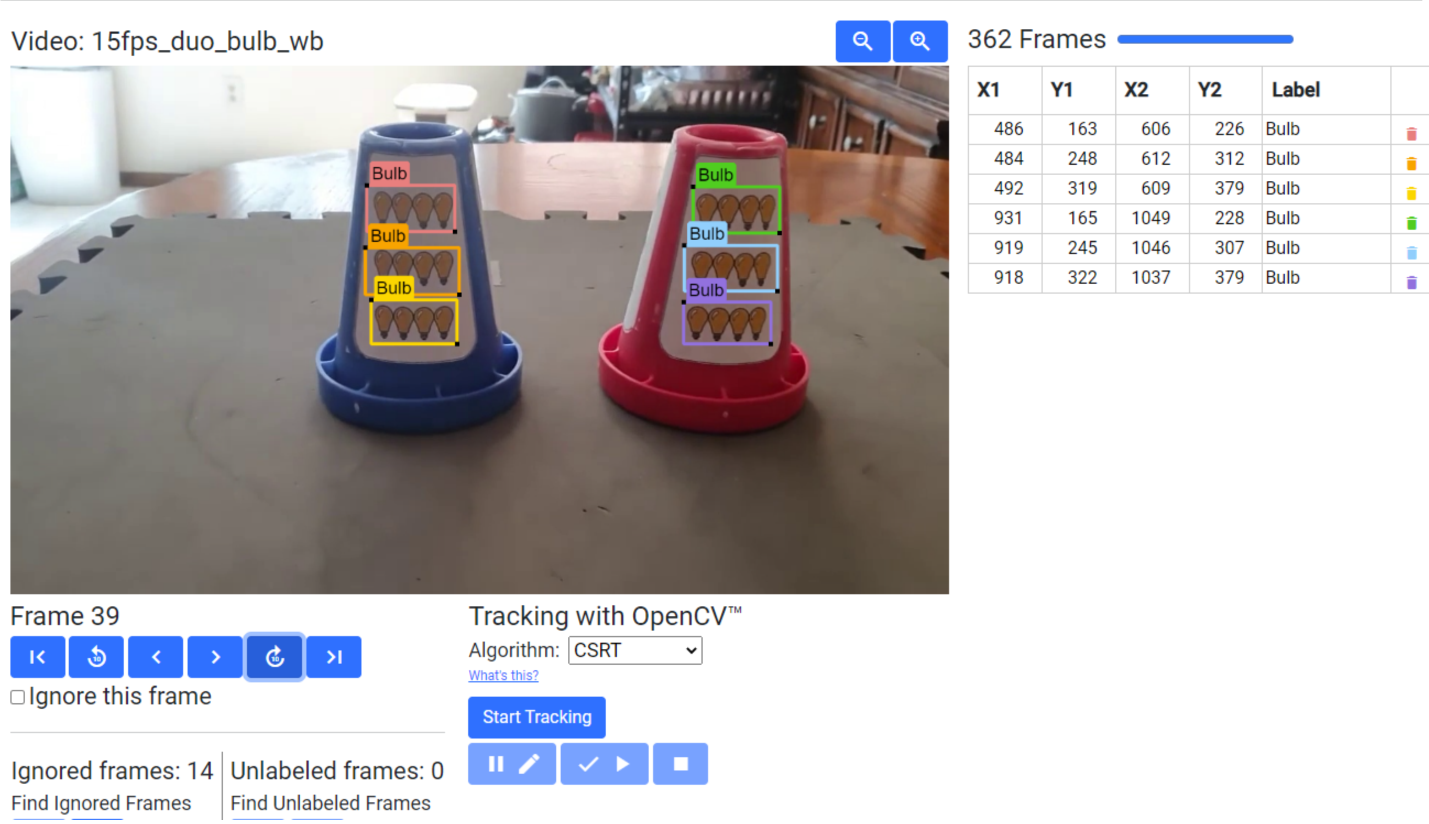
In POWERPLAY, the white sticker background is always present, except the white color of the background can be unintentionally altered based on the lighting being used in the room; warmer lights cause the white to turn yellow or orange, cooler lights cause the white to turn more blue, and glare causes a gradient of colors to appear across the white background. Sometimes algorithms can adjust the color scheme to provide a "white balance" to adjust the colors correctly, but requiring such tools and adjustments might be beyond the grasp for the average user. (See *White Balance Control* and *White Balance Control Mode* for more information about adjusting white balance programmatically within the SDK's Java language libraries).

In order to get TensorFlow to become less sensitive to the need for "white balance" within the frame, and ignore the white altogether, a suite of different lighting scenarios were replicated and used to train the model with the hopes that TensorFlow would eventually see the "areas of changing colors" (due to the different lighting situations) as background and ignore it altogether to focus more on the images themselves. This is ultimately what was successful for the default model. Below are some examples of the lighting conditions used to train the model.

Lighting Scenario 1

Example Lighting Scenario #1

Lighting Scenario 2



Example Lighting Scenario #2

Lighting Scenario 3

Example Lighting Scenario #3

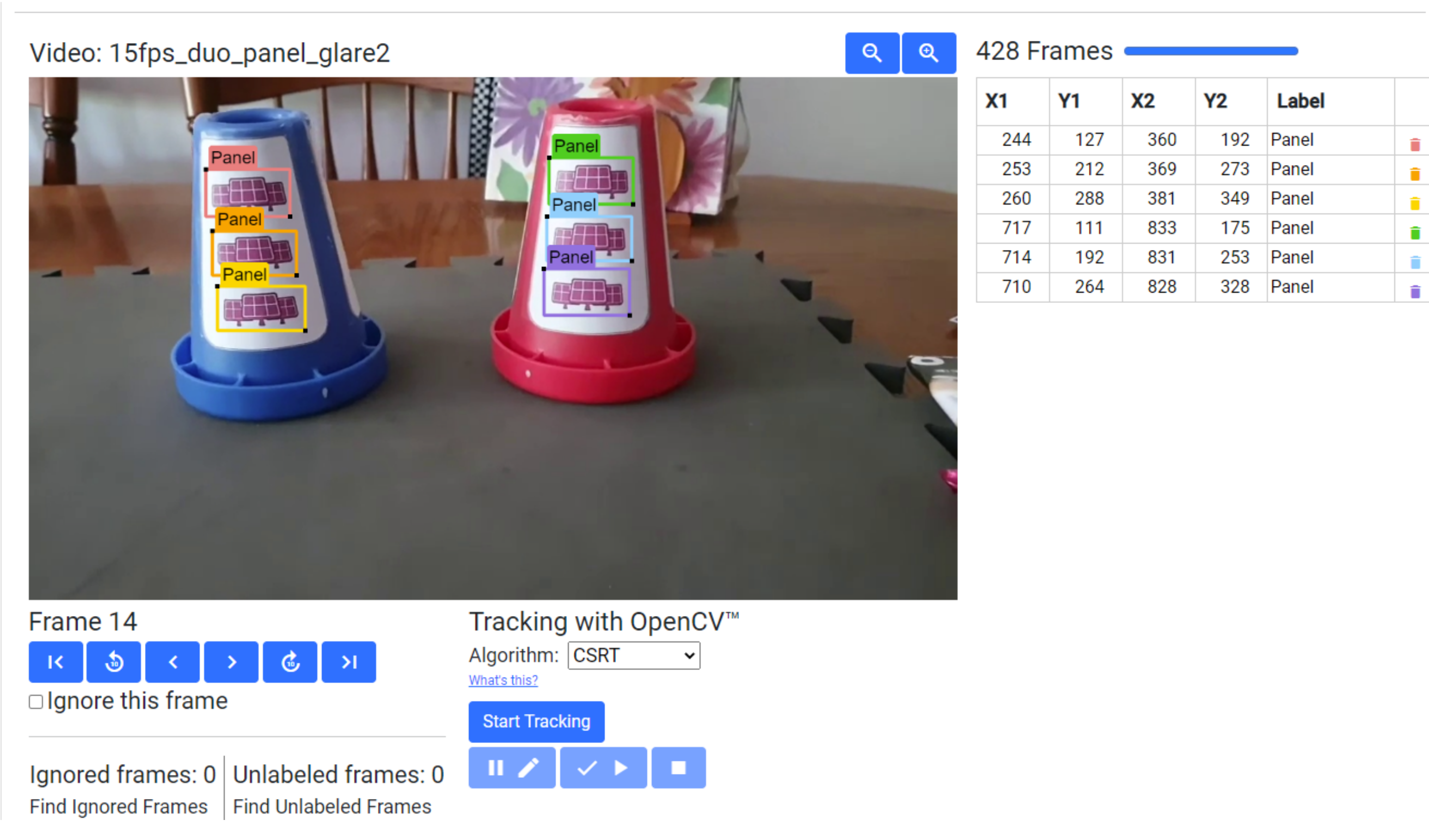
It is recommended that teams choose a background that is more resistant to being "altered" by lighting conditions, and doesn't exist anywhere else on the game field, or try adjusting the *White Balance Control* via programming if you're a Java language user.

Selecting Images For Signal Sleeves

Selecting images to use for your custom Signal Sleeve can seem daunting. Questions swirl like "What images are going to be recognized best?", "Why were the images used in the Default Model chosen?", and "How do I make this easier on myself?". Hopefully this section will help you understand the image selection used for the Default Model, and that will help inform your own decisions for your Signal Sleeve.

First, it's important to note that TensorFlow has the following quirks/behaviors:

- In order to run TensorFlow on mobile phones, *FIRST* Tech Challenge uses a very small core model resolution. This means the image is downscaled from the high definition webcam image to one that is only 300x300 pixels. This means that medium and small objects within the webcam images may be reduced to very small indistinguishable clusters of pixels in the target image. Keep the objects in the view of the camera large, and train for a wide range of image sizes.
- TensorFlow is not really good at differentiating geometric shapes. TensorFlow Object Detection is an object classifier, and similar geometric shapes will classify similarly. Humans are much better at differentiating geometric shapes than neural net algorithms, like TensorFlow, at the present.



- TensorFlow is great at pattern detection, color differentiation, and image textures. For instance, TensorFlow can be easily trained to recognize the difference between Zebras and Horses, but it would not be able to differentiate between specific Zebra patterns to be able to identify, for example, "Carl the Zebra."

The default images were chosen for several design factors:

- Images needed to be vertically short and horizontally long. When setting the TensorFlow zoom factor above 1.0, the aspect ratio causes the zoom window to be wider horizontally than vertically; even at modest zoom factors the zoom window shrinks to be vertically smaller than the sticker itself at even the minimim distance from the robot (18 inches). In order to have more than one detection within the window, and to aid in providing wide margins for adjusting the camera during robot setup, images that are horizontally wide and vertically short were desired. Thanks to the season theme, the green lightning bolt from the *FIRST* Energize season logo was chosen first. The green color and the zig-zag pattern on the top and bottom of the bolt were desired elements for TensorFlow.
- TensorFlow's ability to detect patterns better than shapes was utilized in two ways in the "Bulb" image; first the repeated bulb image created a repeating pattern that TensorFlow could recognize, and the image itself was colored differently than other colors it may have seen on the sticker background, the cones themselves, or on the green lightning bolt. Yellow was selected as the color within the repeating light bulbs. It helped that the light bulb had a similar art style to the lightning bolt and even fit the theme, even though that wasn't a hard requirement.
- Finally, the solar panels were selected similarly to the bulbs. The grid pattern within the solar panels made for a unique pattern element not present in the other images, and the purple color helped offset it as well.

With the images selected, there were only basic tweaks made to the images for use in POWERPLAY. For example, the images were modified to have relatively similar aspect ratios and sizes to aid in uniformity of setup, and it was determined that TensorFlow could be trained to recognize elements of each image fairly well.

When selecting images for use with TensorFlow, keep in mind the elements of pattern, color, and size. For example, a donut can be a great image for use by TensorFlow; not because of the circular shape, but because of the frosting and the sprinkles on top which creates a very unique pattern for TensorFlow to recognize. Be creative!

21.4.3 TensorFlow for FREIGHT FRENZY presented by Raytheon Technologies

What is TensorFlow?

FIRST Tech Challenge teams can use TensorFlow Lite, a lightweight version of Google's TensorFlow machine learning technology that is designed to run on mobile devices such as an Android smartphone. A *trained TensorFlow model* was developed to recognize game elements for the 2021-2022 Freight Frenzy challenge.

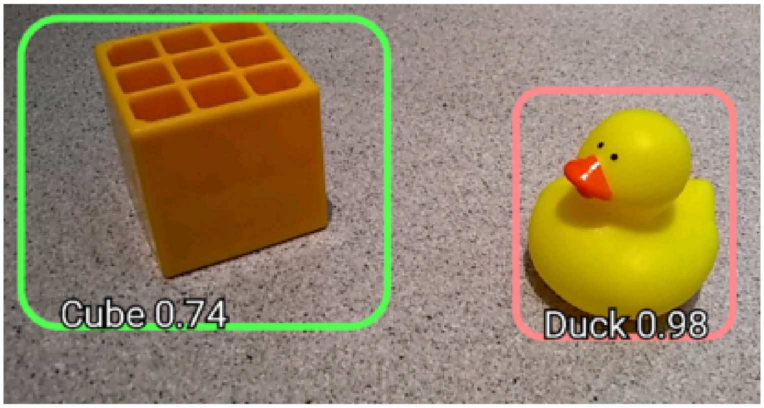


Fig. 110: This season's TFOD model can recognize Freight elements

TensorFlow Object Detection (TFOD) has been integrated into the control system software, to identify and track these game pieces during a match. The software (SDK version 7.0) contains TFOD Sample Op Modes that can recognize the Freight elements Duck, Box (or Cube), and Cargo (or Ball).

How Might a Team Use TensorFlow in Freight Frenzy?

For this season's challenge, during the pre-Match stage a single die is rolled and the field is randomized.

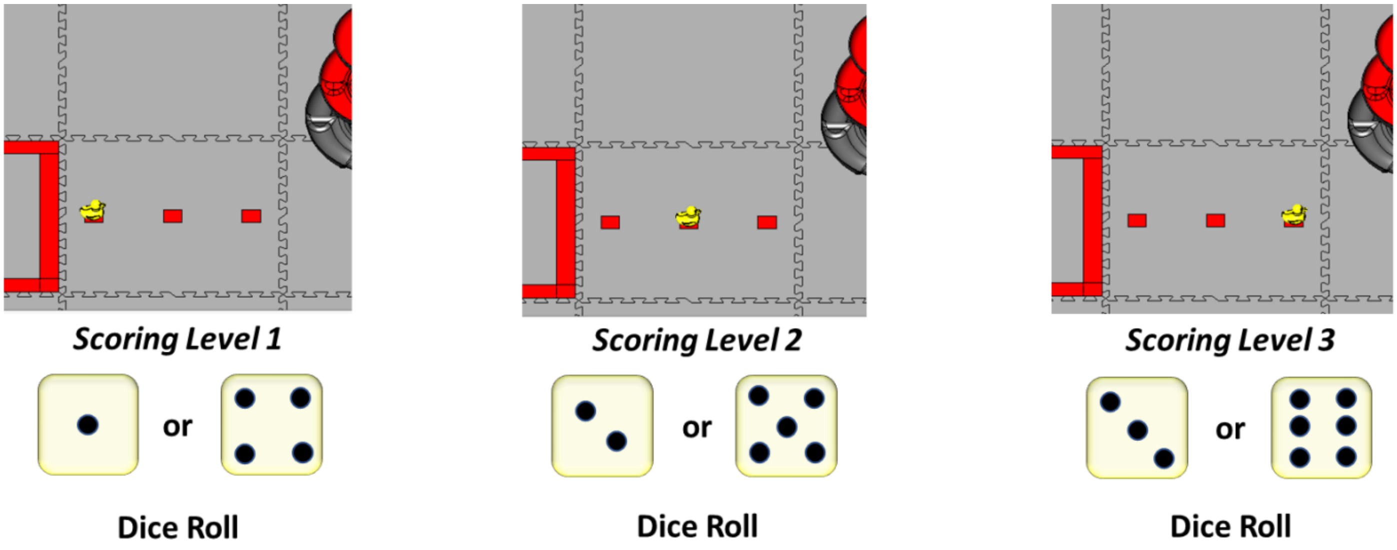


Fig. 111: Randomization

At the beginning of the match's Autonomous period, a robot can use TensorFlow to "look" at the **Barcode** area and determine whether the Duck or optional Team Shipping Element (TSE) is in position 1, 2 or 3. This indicates the preferred scoring level on the **Alliance Shipping Hub**. A bonus is available for using the TSE instead of a Duck.

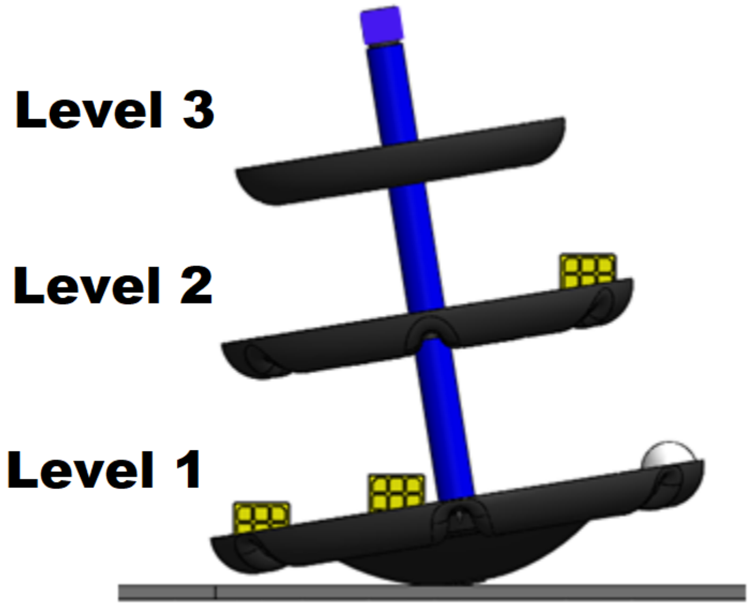


Fig. 112: Alliance Shipping Hub

Important Note on Phone Compatibility

TensorFlow Lite runs on Android 6.0 (Marshmallow) or higher, a requirement met by all currently allowed devices. If you are a Blocks programmer using an older/disallowed Android device that is not running Marshmallow or higher, TFOD Blocks will automatically be missing from the Blocks toolbox or design palette.

Sample Op Modes

The software (SDK version 7.0 and higher) contains sample Blocks and Java op modes that demonstrate TensorFlow **recognition** of Freight elements Duck, Box (cube) and Cargo (ball). The sample op modes also show **where** in the camera's field of view a detected object is located.

Click on the following links to learn more about these sample Op Modes.

- *Blocks TensorFlow Object Detection Example*
- *Java TensorFlow Object Detection Example*

Using a Custom Inference Model

Teams have the option of using a custom inference model with the FIRST Tech Challenge software. As noted above, the **Machine Learning toolchain** is a streamlined tool for training your own TFOD models. An alternate would be to use the TensorFlow Object Detection API to create an enhanced model of the Freight elements or TSE, or to create a custom model to detect other entirely different objects. Other teams might also want to use an available pre-trained model to build a robot that can detect common everyday objects (for demo or outreach purposes, for example).

The software includes sample op modes (Blocks and Java versions) that demonstrate how to use a **custom inference model**:

- Using a Custom TensorFlow Model with Blocks
- Using a Custom TensorFlow Model with Java

These tutorials use examples from a previous season (Skystone), but the process remains generally valid for Freight Frenzy.

Detecting Everyday Objects

You can use a pretrained TensorFlow Lite model to detect **everyday objects**, such as a clock, person, computer mouse, or cell phone. The following advanced tutorial shows how you can use a free, pretrained model to recognize numerous everyday objects.

- Using a TensorFlow Pretrained Model to Detect Everyday Objects

---

Updated 11/19/21

21.4.4 Blocks Sample OpMode for TFOD

Introduction

This tutorial describes the FTC Blocks Sample OpMode for TensorFlow Object Detection (TFOD). This Sample, called "ConceptTensorFlowObjectDetection", can recognize one or more official game elements and provide their visible size and position.

For the 2023-2024 game CENTERSTAGE, the game element is a hexagonal white **Pixel**. The FTC SDK software contains a TFOD model of this object, ready for recognition.

For extra points, teams may instead use their own custom TFOD models of **Team Props**. That option is described here:

- *Blocks Custom Model Sample OpMode for TFOD*

Creating the OpMode

At the FTC Blocks browser interface, click on the "Create New OpMode" button to display the Create New OpMode dialog box.

Specify a name for your new OpMode. Select "ConceptTensorFlowObjectDetection" as the Sample OpMode that will be the template for your new OpMode.

If no webcam is configured for your REV Control Hub, the dialog box will display a warning message (shown here). You can ignore this warning message if you will use the built-in camera of an Android RC phone. Click "OK" to create your new OpMode.

The new OpMode should appear in edit mode in your browser.

By default, the Sample OpMode assumes you are using a webcam, configured as "Webcam 1". If you are using the built-in camera on your Android RC phone, change the USE_WEBCAM Boolean from `true` to `false` (green arrow above).

Adjusting the Zoom Factor

If the object to be recognized will be more than roughly 2 feet (61 cm) from the camera, you might want to set the digital zoom factor to a value greater than 1. This tells TensorFlow to use an artificially magnified portion of the image, which may offer more accurate recognitions at greater distances.

Pull out the **``setZoom``** Block, found in the toolbox or palette called "Vision", under "TensorFlow" and "TfodProcessor" (see green oval above). Change the magnification value as desired (green arrow).

On REV Control Hub, the "Vision" menu appears only when the active robot configuration contains a webcam, even if not plugged in.

This `setZoom` Block can be placed in the INIT section of your OpMode,

- immediately after the call to the `initTfod` Function, or

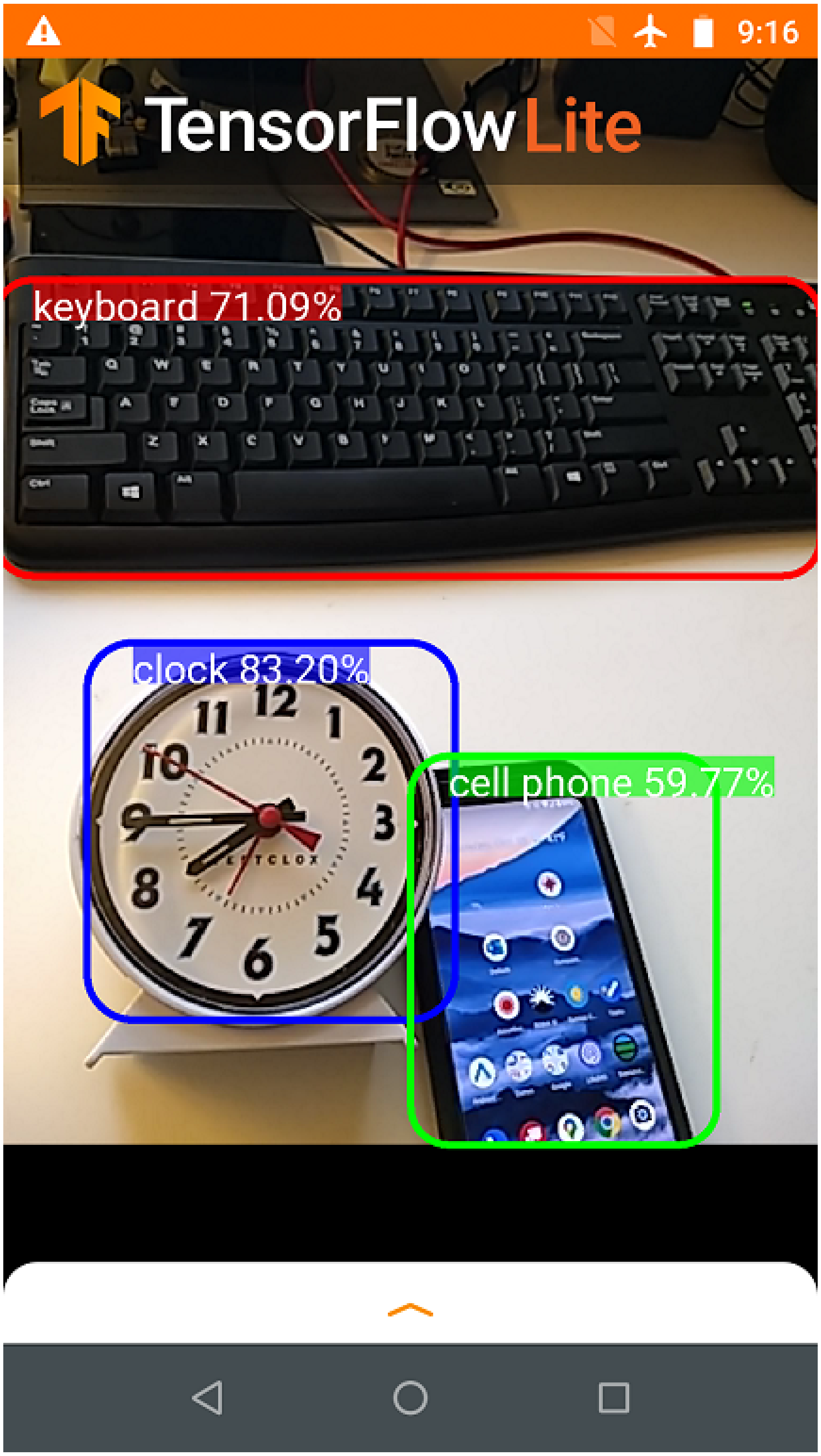


Fig. 113: TensorFlow can recognize everyday objects

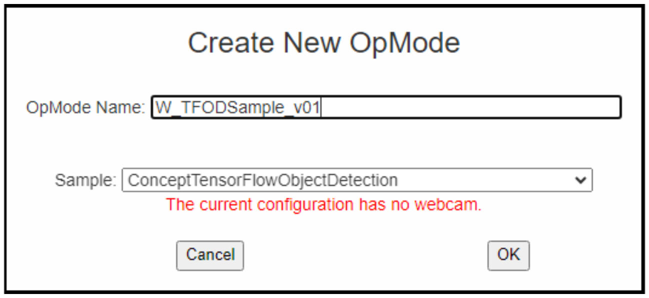


Fig. 114: Creating a New OpMode

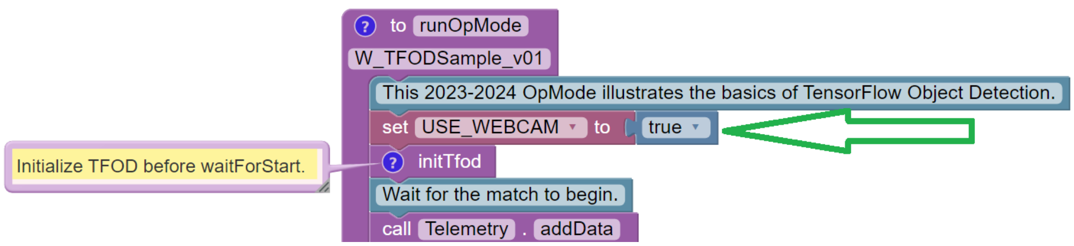


Fig. 115: Sample OpMode

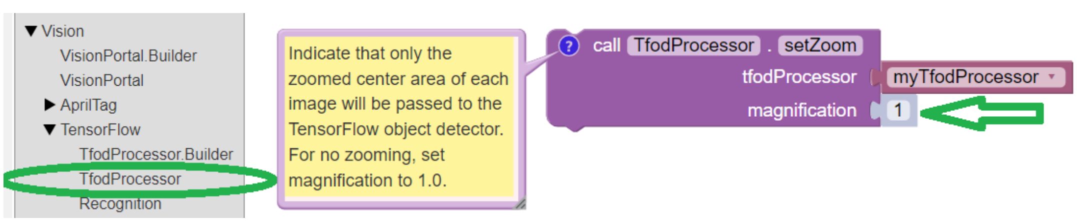


Fig. 116: Setting the Zoom Factor

- as the very last Block inside the `initTfod` Function.

This Block is **not** part of the Processor Builder pattern, so the Zoom factor can be set to other values during the OpMode, if desired.

The "zoomed" region can be observed in the DS preview (Camera Stream) and the RC preview (LiveView), surrounded by a greyed-out area that is **not evaluated** by the TFOD Processor.

Other Adjustments

The Sample OpMode uses a default **minimum confidence** level of 75%. The TensorFlow Processor needs to have a confidence level of 75% or higher, to consider an object as "recognized" in its field of view.

You can see the object name and actual confidence (as a **decimal**, e.g. 0.75) near the Bounding Box, in the Driver Station preview (Camera Stream) and Robot Controller preview (Liveview).

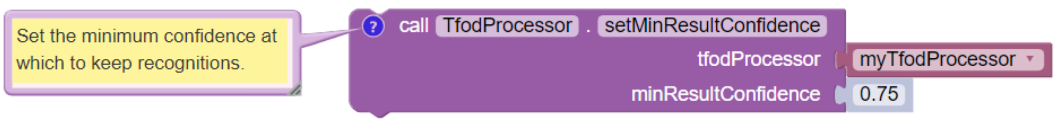


Fig. 117: Setting the Minimum Confidence

Pull out the **``setMinResultConfidence``** Block, found in the toolbox or palette called "Vision", under "TensorFlow" and "TfodProcessor". Adjust this parameter to a higher value if you would like the processor to be more selective in identifying an object.

Another option is to define, or clip, a **custom area for TFOD evaluation**, unlike `setZoom` which is always centered.

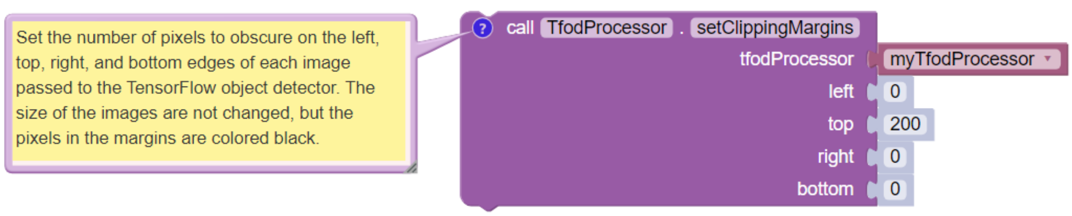


Fig. 118: Setting Clipping Margins

From the same Blocks palette, pull out the **``setClippingMargins``** Block. Adjust the four margins as desired, in units of pixels.

These Blocks can be placed in the INIT section of your OpMode,

- immediately after the call to the `initTfod` Function, or
- as the very last Blocks inside the `initTfod` Function.

As with `setZoom`, these Blocks are **not** part of the Processor Builder pattern, so they can be set to other values during the OpMode, if desired.

Command Flow in this Sample

After the `waitForStart` Block, this OpMode contains the main program loop:

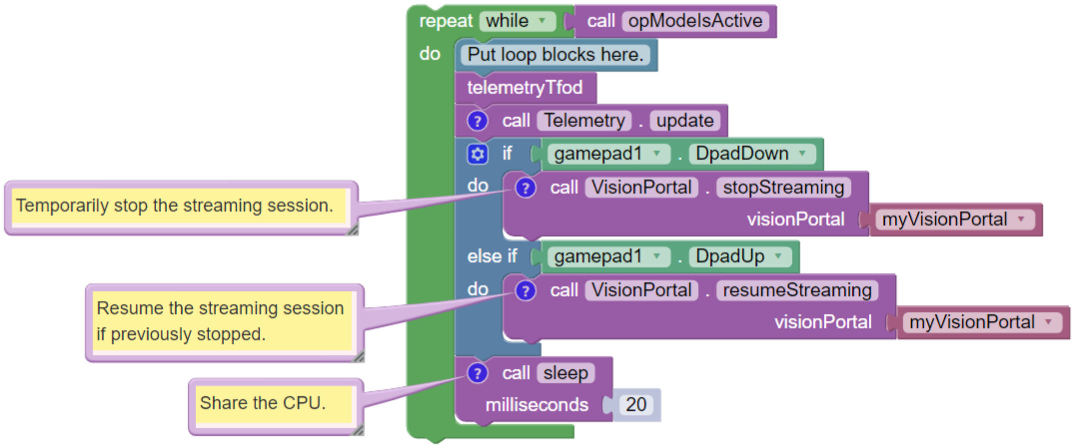


Fig. 119: OpMode Main Loop

This loop repeatedly calls a Blocks Function called ``**telemetryTfod**``. That Function is the heart of the OpMode, seeking and evaluating recognized TFOD objects, and displaying DS Telemetry about those objects. It will be discussed below, in the next section.

The main loop also allows the user to press the `Dpad Down` button on the gamepad, to temporarily stop the streaming session. This `.stopStreaming` Block pauses the flow and processing of camera frames, thus **conserving CPU resources**.

Pressing the `Dpad Up` button (`.resumeStreaming`) allows the processing to continue. The on-and-off actions can be observed in the RC preview (LiveView), described further below.

These two commands appear here in this Sample OpMode, to spread awareness of one tool for managing CPU and bandwidth resources. The FTC VisionPortal offers over 10 such controls, *described here*.

Processing TFOD Recognitions

The Function called ``**telemetryTfod**`` is the heart of the OpMode, seeking and evaluating recognized TFOD objects, and displaying DS Telemetry about those objects.

The first Block uses the TFOD Processor to gather and store all recognitions in a List, called `myTfodRecognitions`.

The green "FOR Loop" iterates through that List, handling each item, one at a time. Here the "handling" is simply displaying certain TFOD fields to DS Telemetry.

For competition, you want to do more than display Telemetry, and you want to exit the main loop at some point. These code modifications are discussed in another section below.

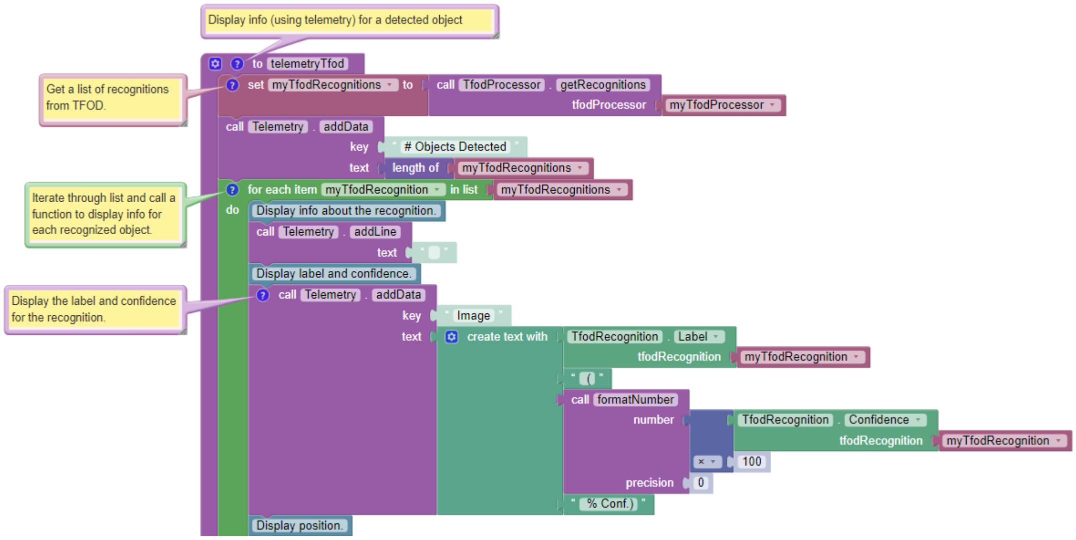


Fig. 120: Telemetry TFOD

Testing the OpMode

Click the "Save OpMode" button, then run the OpMode from the Driver Station. The Robot Controller should use the CENTERSTAGE TFOD model to recognize and track the white Pixel.

For a preview during the INIT phase, touch the Driver Station's 3-dot menu and select **Camera Stream**.

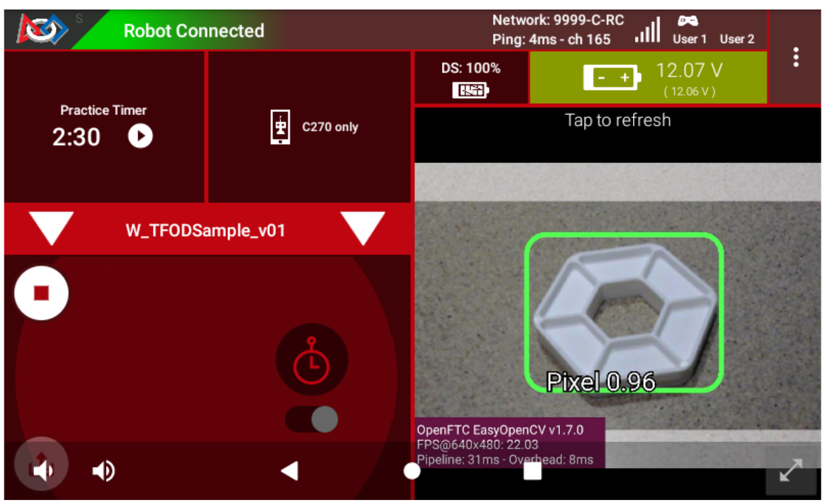


Fig. 121: Sample DS Camera Stream

Camera Stream is not live video; tap to refresh the image. Use the small white arrows at lower right to expand or revert the preview size. To close the preview, choose 3-dots and Camera Stream again.

After touching the DS START button, the OpMode displays Telemetry for any recognized Pixel(s):

The above Telemetry shows the label name, and TFOD confidence level. It also gives the **center location** and **size** (in pixels) of the Bounding Box, which is the colored rectangle surrounding the recognized object.

The pixel origin (0, 0) is at the top left corner of the image.

Before and after touching DS START, the Robot Controller provides a video preview called **LiveView**.

For Control Hub (with no built-in screen), plug in an HDMI monitor or learn about `scrcpy` (https://github.com/Genymobile/scrcpy). The above image is a LiveView screenshot via `scrcpy`.

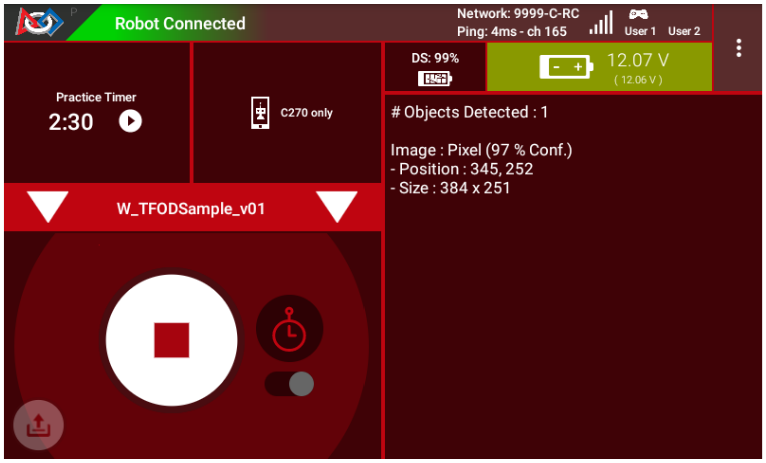


Fig. 122: Sample DS Telemetry

Fig. 123: Sample RC LiveView

If you don't have a physical Pixel on hand, try pointing the camera at this image:

Fig. 124: Sample Pixel

Modifying the Sample

In this Sample OpMode, the main loop ends only upon touching the DS Stop button. For competition, teams should **modify this code** in at least two ways:

- for a significant recognition, take action or store key information – inside the FOR loop
- end the main loop based on your criteria, to continue the OpMode

As an example, you might set a Boolean variable `isPixelDetected` to `true`, if a significant recognition has occurred.

You might also evaluate and store which randomized Spike Mark (red or blue tape stripe) holds the white Pixel.

Regarding the main loop, it could end after the camera views all three Spike Marks, or after your code provides a high-confidence result. If the camera's view includes more than one Spike Mark position, perhaps the white Pixel's **Bounding Box** size and location could be useful. Teams should consider how long to seek an acceptable recognition, and what to do otherwise.

In any case, the OpMode should exit the main loop and continue running, using any stored information.

Best of luck this season!

Questions, comments and corrections to westsiderobotics@verizon.net

21.4.5 Blocks Custom Model Sample OpMode for TFOD

Introduction

This tutorial uses an FTC Blocks Sample OpMode to load and recognize a **custom TensorFlow inference model**.

- In this example, the "custom model" is actually the standard trained model of the 2023-2024 CENTERSTAGE game element called a **Pixel**. This does not affect the process described for a custom model.

Downloading the Model

The Robot Controller allows you to load a trained inference model in the form of a TensorFlow Lite (`.tflite`) file.

Here we use the standard FTC `.tflite` file from CENTERSTAGE (2023-2024), available on GitHub at the following link:

- CENTERSTAGE TFLite File

Note: Very advanced teams could use Google's TensorFlow Object Detection API (https://github.com/tensorflow/models/tree/master/research/object_detection) to create their own custom inference model.

Click the "Download Raw File" button to download the `CenterStage.tflite` file from GitHub to your local device (e.g. laptop). See the green arrow.

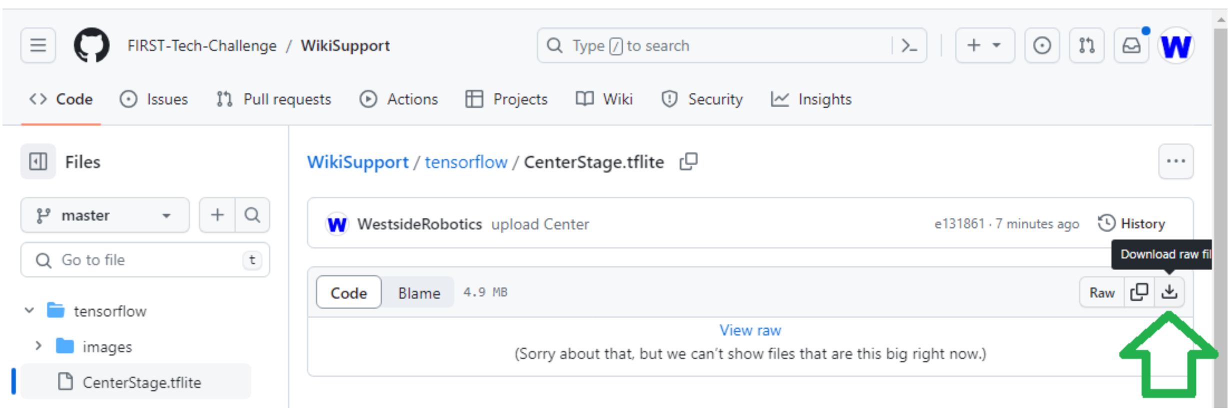


Fig. 125: Public repo for CenterStage tflite file

Uploading to the Robot Controller

After downloading the file to your laptop, you need to upload it to the Robot Controller. Connect your laptop to your Robot Controller's wireless network and navigate to the FTC "Manage" page:

Scroll down and click on "Manage TensorFlow Lite Models".

Now click the "Upload Models" button.

Click "Choose Files", and use the dialog box to find and select the downloaded `CenterStage.tflite` file.

Now the file will upload to the Robot Controller. The file will appear in the list of TensorFlow models available for use in OpModes.

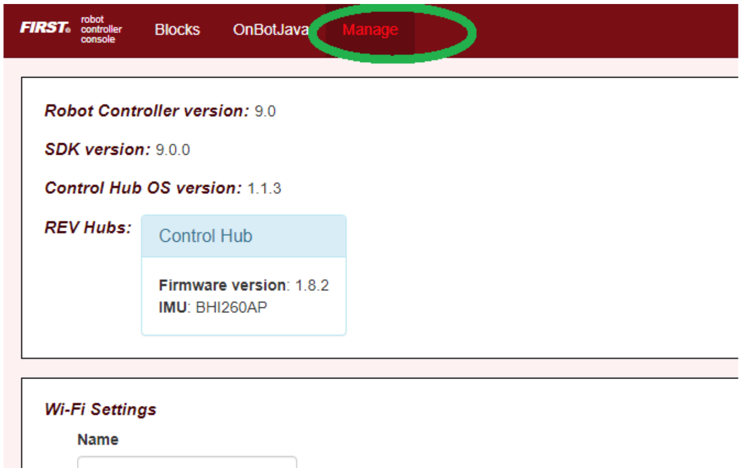


Fig. 126: Example of the Manage Page

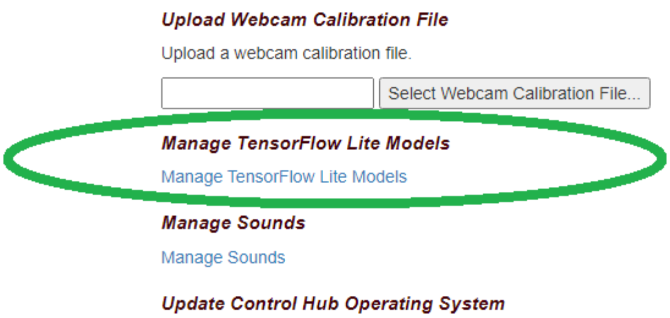


Fig. 127: Manage TFLITE Models Link

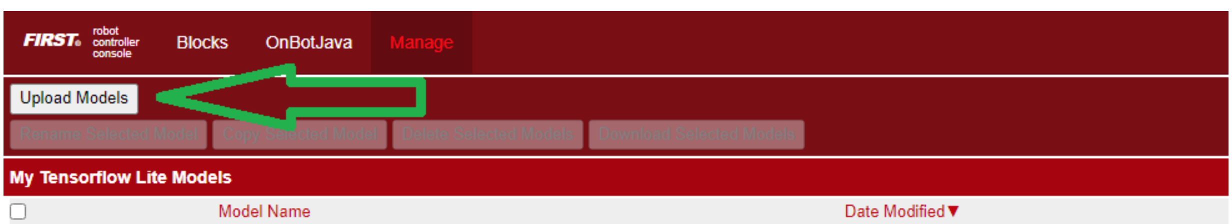


Fig. 128: Upload TFLITE Models Button

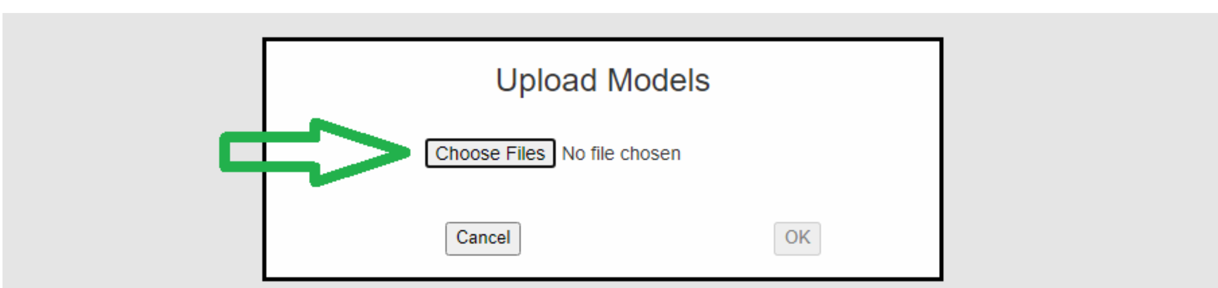


Fig. 129: Upload TFLITE Models Button

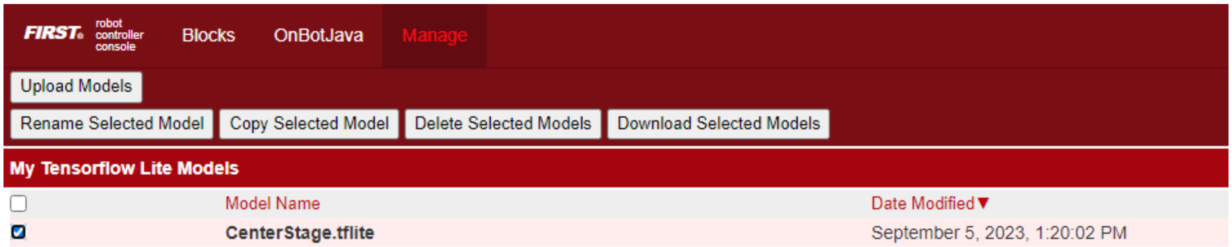


Fig. 130: TFLITE Model Listed

Creating the OpMode

Click on the “Blocks” tab at the top of the screen to navigate to the Blocks Programming page. Click on the “Create New OpMode” button to display the Create New OpMode dialog box.

Specify a name for your new OpMode. Select “ConceptTensorFlowObjectDetectionCustomModel” as the Sample OpMode that will be the template for your new OpMode.

If no webcam is configured for your REV Control Hub, the dialog box will display a warning message (shown here). You can ignore this warning message if you will use the built-in camera of an Android RC phone. Click “OK” to create your new OpMode.

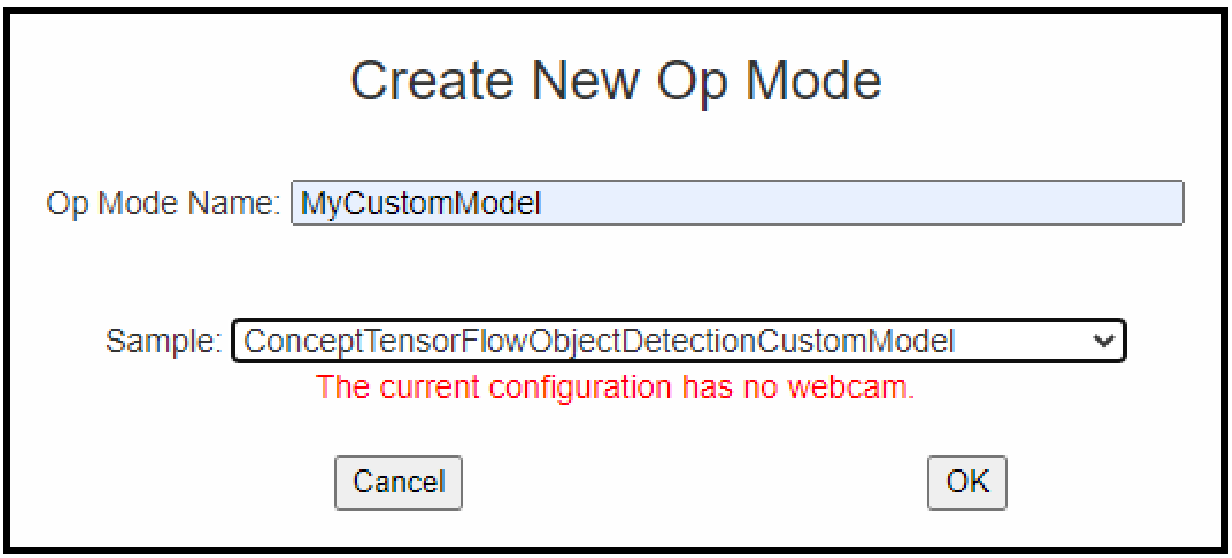


Fig. 131: Create New OpMode

The new OpMode should appear in edit mode in your browser.

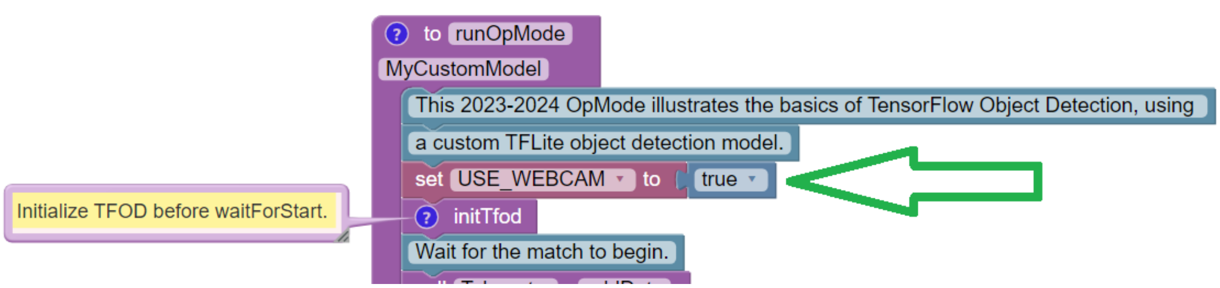


Fig. 132: Sample OpMode

By default, the Sample OpMode assumes you are using a webcam, configured as "Webcam 1". If you are using the built-in camera on your Android RC phone, change the USE_WEBCAM Boolean from `true` to `false` (green arrow above).

Loading the Custom Model

Scroll down in the OpMode, to the Blocks Function called "initTfod".

In the Block with ".setModelFileName", change the filename from "MyCustomModel.tflite" to `CenterStage.tflite` – or other filename that you uploaded to the Robot Controller. The filename must be an exact match. See green oval below.

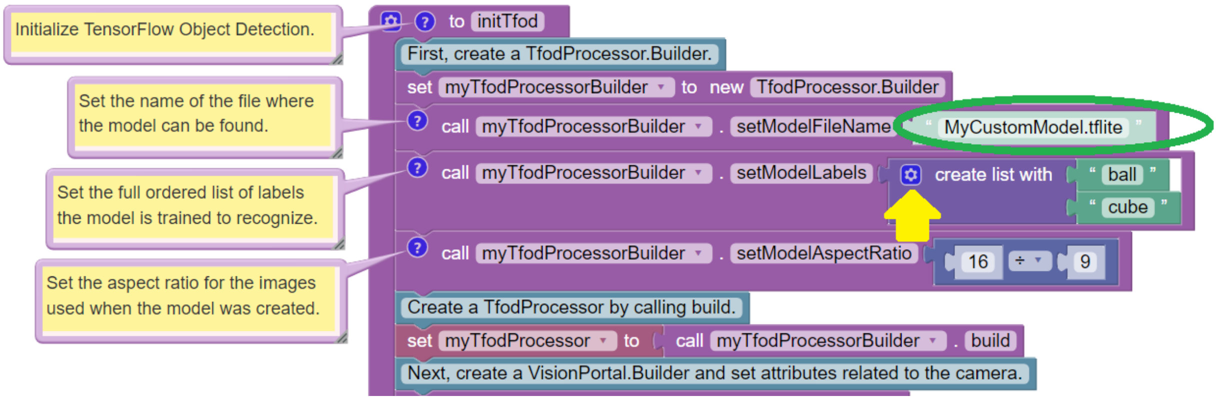


Fig. 133: Init TFOD Function

When loading an inference model, you must specify a list of **labels** that describe the known objects in the model. This is done in the next Block, with ".setModelLabels".

This Sample OpMode assumes a default model with two known objects, labeled "ball" and "cube". The CENTERSTAGE model contains only one object, labeled "Pixel".

For competition, the **Team Prop** label names might be `myTeamProp_Red` and/or `myTeamProp_Blue`.

The number of labels can be changed by clicking the small blue gear icon for the "create list with" Block (see yellow arrow).

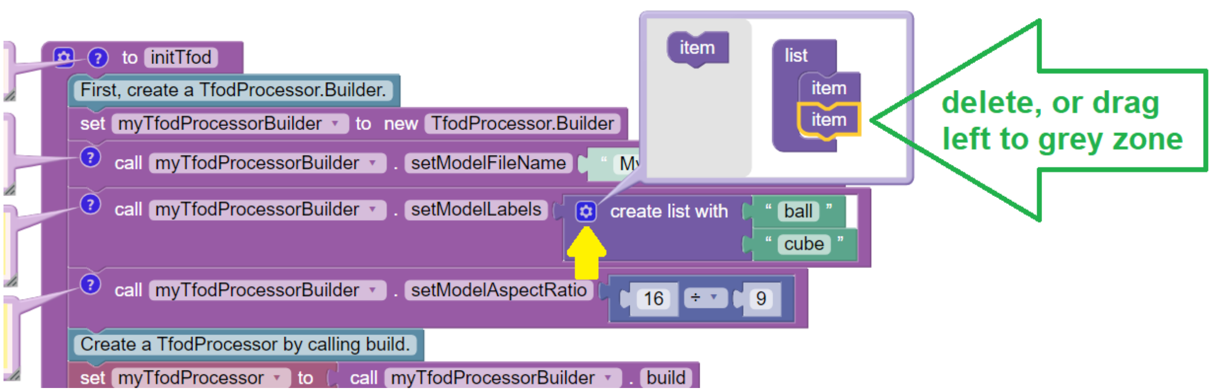


Fig. 134: Blue Gear Delete

In the pop-up layout balloon, click on one of the list items to select it (green arrow above). Then remove it, by pressing Delete (on keyboard), or by dragging it to the balloon's left-side grey zone.

After editing that purple "list" structure, click the blue gear icon again to close the layout balloon. Edit the remaining label to "Pixel".

When complete, the edited Blocks should look like this:

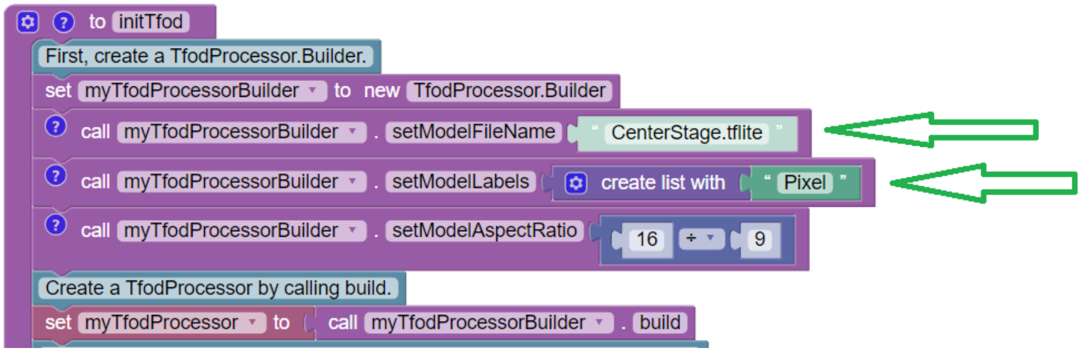


Fig. 135: Adding Pixel Label

Adjusting the Zoom Factor

If the object to be recognized will be more than roughly 2 feet (61 cm) from the camera, you might want to set the digital zoom factor to a value greater than 1. This tells TensorFlow to use an artificially magnified portion of the image, which may offer more accurate recognitions at greater distances.

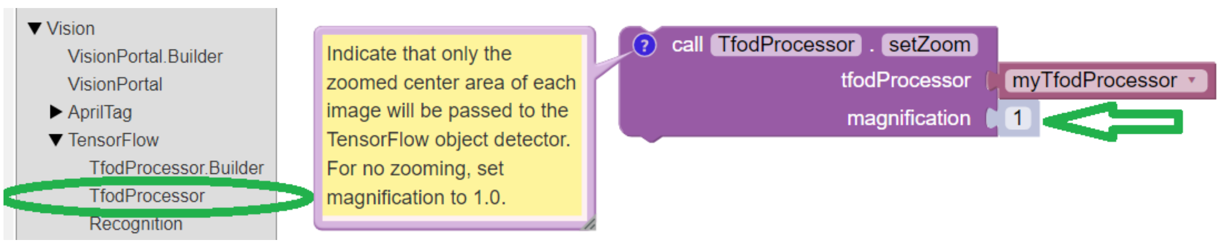


Fig. 136: Set Zoom

Pull out the **"setZoom" Block**, found in the toolbox or palette called "Vision", under "TensorFlow" and "TfodProcessor" (see green oval above). Change the magnification value as desired (green arrow).

On REV Control Hub, the "Vision" menu appears only when the active robot configuration contains a webcam, even if not plugged in.

Place this Block immediately after the Block `set myTfodProcessor to call myTfodProcessorBuilder.build`. This Block is **not** part of the Processor Builder pattern, so the Zoom factor can be set to other values during the OpMode, if desired.

The "zoomed" region can be observed in the DS preview (Camera Stream) and the RC preview (LiveView), surrounded by a greyed-out area that is **not evaluated** by the TFOD Processor.

Testing the OpMode

Click the "Save OpMode" button, then run the OpMode from the Driver Station. The Robot Controller should use the new CENTERSTAGE inference model to recognize and track the Pixel game element.

For a preview during the INIT phase, touch the Driver Station's 3-dot menu and select **Camera Stream**.

Camera Stream is not live video; tap to refresh the image. Use the small white arrows at lower right to expand or revert the preview size. To close the preview, choose 3-dots and Camera Stream again.

After touching the DS START button, the OpMode displays Telemetry for any recognized Pixel(s):

The above Telemetry shows the label name, and TFOD confidence level. It also gives the **center location** and **size** (in pixels) of the Bounding Box, which is the colored rectangle surrounding the recognized object.

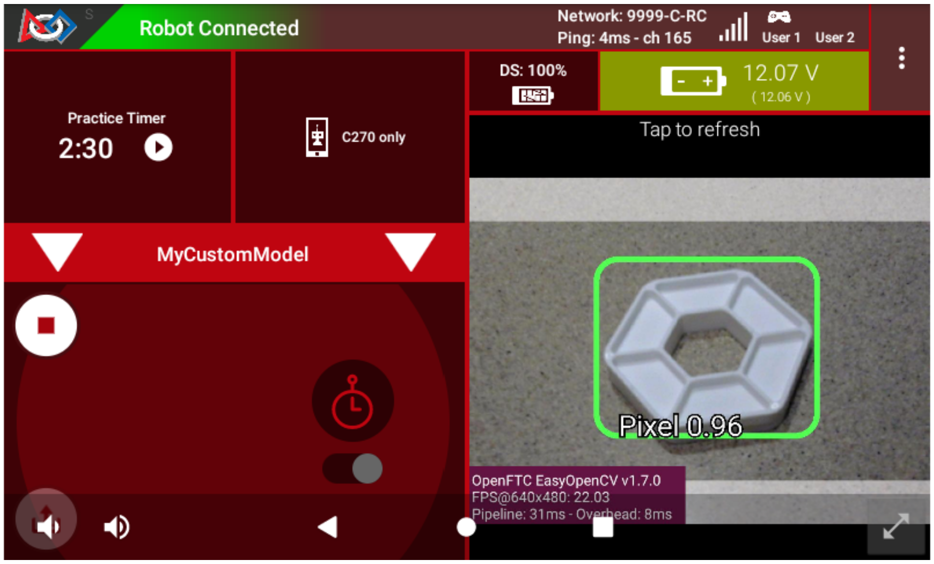


Fig. 137: DS Camera Stream

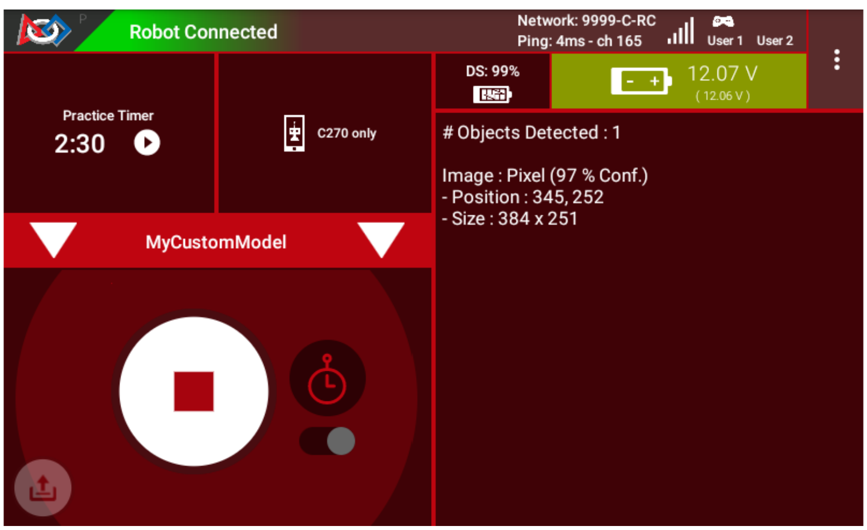


Fig. 138: DS Telemetry

The pixel origin (0, 0) is at the top left corner of the image.

Before and after touching DS START, the Robot Controller provides a video preview called **LiveView**.

Fig. 139: RC LiveView

For Control Hub (with no built-in screen), plug in an HDMI monitor or learn about `scrcpy` (https://github.com/Genymobile/scrcpy). The above image is a LiveView screenshot via `scrcpy`.

If you don't have a physical Pixel on hand, try pointing the camera at this image:

Modifying the Sample

In this Sample OpMode, the main loop ends only upon touching the DS Stop button. For competition, teams should **modify this code** in at least two ways:

- for a significant recognition, take action or store key information – inside the FOR loop
- end the main loop based on your criteria, to continue the OpMode

As an example, you might set a Boolean variable `isTeamPropDetected` to `true`, if a significant recognition has occurred.

You might also evaluate and store which randomized Spike Mark (red or blue tape stripe) holds the Team Prop.

Regarding the main loop, it could end after the camera views all three Spike Marks, or after your code provides a high-confidence result. If the camera's view includes more than one Spike Mark position, perhaps the Team Prop's **Bounding Box** size and location could be useful. Teams should consider how long to seek an acceptable recognition, and what to do otherwise.

In any case, the OpMode should exit the main loop and continue running, using any stored information.

Best of luck this season!

Questions, comments and corrections to westsiderobotics@verizon.net

Fig. 140: Sample Pixel

21.4.6 Java Easy Sample OpMode for TFOD

Introduction

This tutorial describes the "Easy" version of the FTC Java Sample OpMode for TensorFlow Object Detection (TFOD).

This Sample, called "ConceptTensorFlowObjectDetectionEasy.java", can recognize official FTC game elements and provide their visible size and position. It uses standard/default TFOD settings.

For the 2023-2024 game CENTERSTAGE, the game element is a hexagonal white **Pixel**. The FTC SDK software contains a TFOD model of this object, ready for recognition.

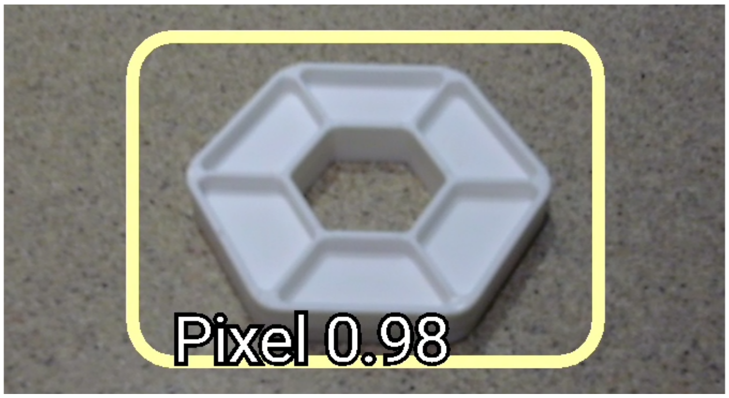


Fig. 141: Sample TFOD Recognition

For extra points, teams may instead use their own custom TFOD models of **Team Props**. That option is described here:

- *Java Custom Model Sample OpMode for TFOD*

This tutorial shows **OnBot Java** screens. Users of **Android Studio** can follow along, since the Sample OpMode is exactly the same.

A different Sample OpMode shows how to set **TFOD options**, unlike the "Easy" version which uses only standard/default TFOD settings. That version, called "ConceptTensorFlowObjectDetection.java" has good commenting to guide users in the Java **Builder pattern** for custom settings.

The "Easy" OpMode covered here does not require the user to work with the Builder pattern, although the SDK does use it internally.

Creating the OpMode

At the FTC OnBot Java browser interface, click on the large black **plus-sign icon** "Add File", to open the New File dialog box.

Specify a name for your new OpMode. Select "ConceptTensorFlowObjectDetectionEasy" as the Sample OpMode that will be the template for your new OpMode.

This Sample has optional gamepad inputs, so it could be designated as a **TeleOp** OpMode (see above).

Click "OK" to create your new OpMode.

Android Studio users should follow the commented instructions to copy this class from the Samples folder to the Teamcode folder, with a new name. Also remove the `@Disabled` annotation, to make the OpMode visible in the Driver Station list.

The new OpMode should appear in edit mode in your browser.

By default, the Sample OpMode assumes you are using a webcam, configured as "Webcam 1". If you are using the built-in camera on your Android RC phone, change the USE_WEBCAM Boolean from `true` to `false` (orange oval above).

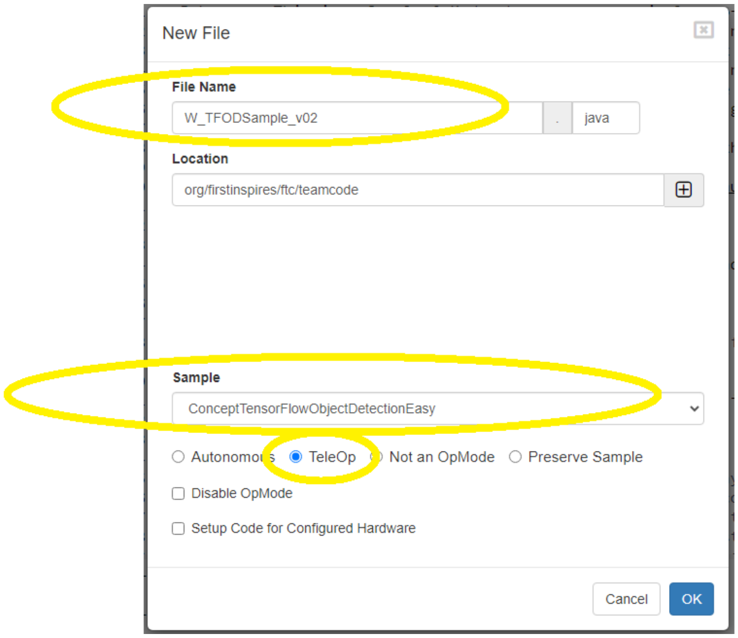


Fig. 142: New File Dialog

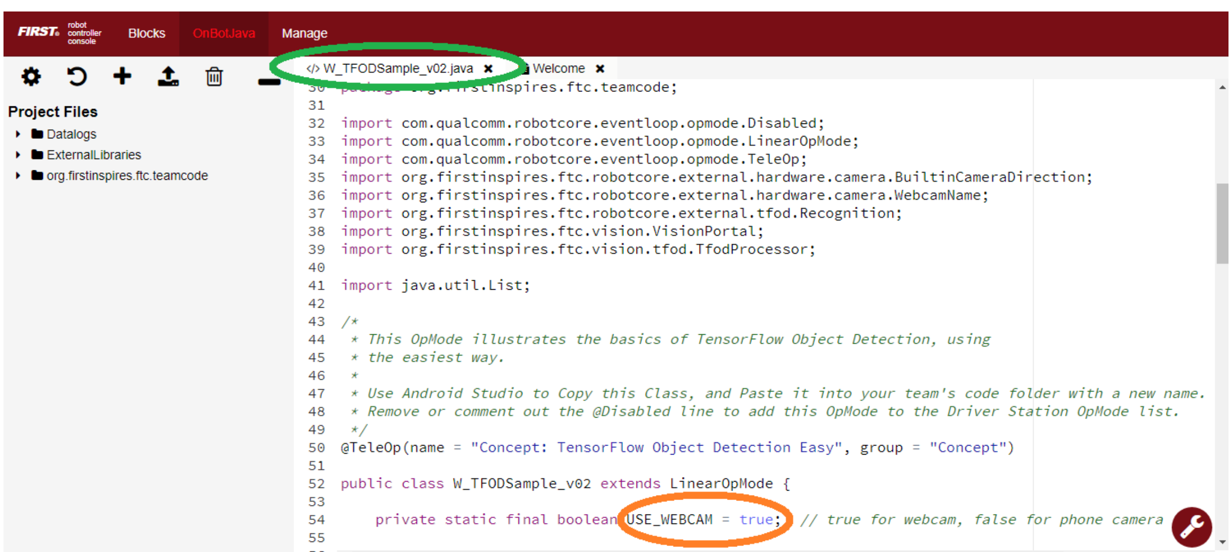


Fig. 143: Opening New Sample

Preliminary Testing

This OpMode is ready to use – it's the "Easy" version!

Click the "Build Everything" button (wrench icon at lower right), and wait for confirmation "BUILD SUCCESSFUL".

If Build is prevented by some other OpMode having errors/issues, they must be fixed before your new OpMode can run. For a quick fix, you could right-click on that filename and choose "Disable/Comment". This "comments out" all lines of code, effectively removing that file from the Build. That file can be re-activated later with "Enable/Uncomment".

In Android Studio (or OnBot Java), you can open a problem class/OpMode and type **CTRL-A** and **CTRL-/** to select and "comment out" all lines of code. This is reversible with **CTRL-A** and **CTRL-/** again.

Now run your new OpMode from the Driver Station (on the TeleOp list, if so designated). The OpMode should recognize any CENTERSTAGE white Pixel within the camera's view, based on the trained TFOD model in the SDK.

For a **preview** during the INIT phase, touch the Driver Station's 3-dot menu and select **Camera Stream**.

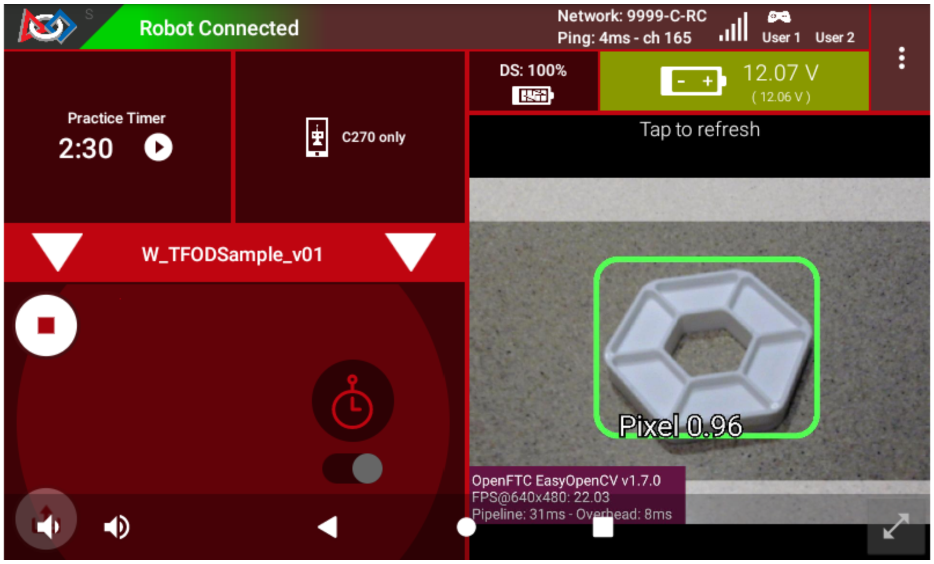


Fig. 144: DS Camera Stream

Camera Stream is not live video; tap to refresh the image. Use the small white arrows at lower right to expand or revert the preview size. To close the preview, choose 3-dots and Camera Stream again.

After the DS START button is touched, the OpMode displays Telemetry for any recognized Pixel(s):

The above Telemetry shows the Label name, and TFOD recognition confidence level. It also gives the **center location** and **size** (in pixels) of the Bounding Box, which is the colored rectangle surrounding the recognized object.

The pixel origin (0, 0) is at the top left corner of the image.

Before and after DS START is touched, the Robot Controller provides a video preview called **LiveView**.

For Control Hub (with no built-in screen), plug in an HDMI monitor or learn about `scrcpy` (https://github.com/Genymobile/scrcpy). The above image is a LiveView screenshot via `scrcpy`.

If you don't have a physical Pixel on hand, try pointing the camera at this image:

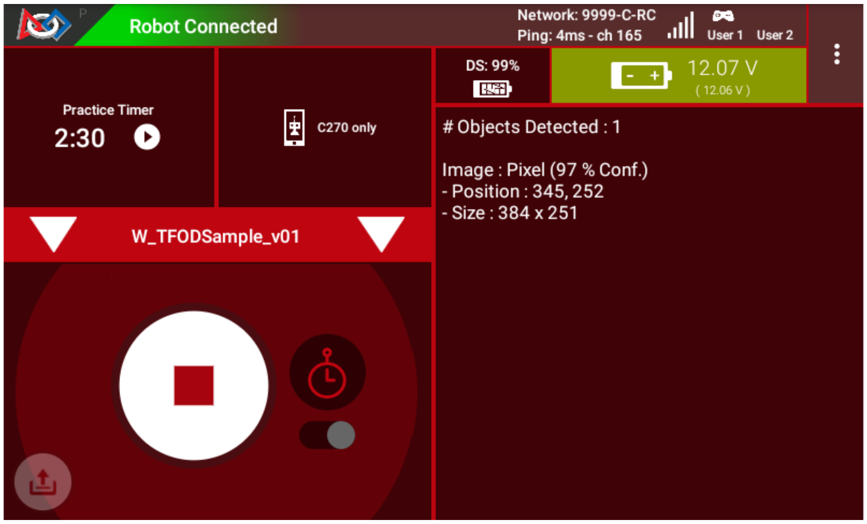


Fig. 145: DS Telemetry Display

Fig. 146: Sample RC LiveView

Fig. 147: Example of a Pixel

Program Logic and Initialization

During the INIT stage (before DS START is touched), this OpMode calls a **method to initialize** the TFOD Processor and the FTC VisionPortal. After DS START is touched, the OpMode runs a continuous loop, calling a **method to display telemetry** about any TFOD recognitions. The OpMode also contains two optional features to remind teams about **CPU resource management**, useful in vision processing.

Here's the first method, to initialize the TFOD Processor and the FTC VisionPortal.

```
/**
 * Initialize the TensorFlow Object Detection processor.
 */
private void initTfod() {

    // Create the TensorFlow processor the easy way.
    tfod = TfodProcessor.easyCreateWithDefaults();

    // Create the vision portal the easy way.
    if (USE_WEBCAM) {
        visionPortal = VisionPortal.easyCreateWithDefaults(
            hardwareMap.get(WebcamName.class, "Webcam 1"), tfod);
    } else {
        visionPortal = VisionPortal.easyCreateWithDefaults(
            BuiltinCameraDirection.BACK, tfod);
    }

}   // end method initTfod()
```

For the **TFOD Processor**, the method `easyCreateWithDefaults()` uses standard default settings. Most teams don't need to modify these, especially for the built-in TFOD model (white Pixel).

For the **VisionPortal**, the method `easyCreateWithDefaults()` requires parameters for camera name and processor(s) used, but otherwise uses standard default settings such as:

- camera resolution 640 x 480
- non-compressed streaming format YUY2
- enable RC preview (called LiveView)
- if TFOD and AprilTag processors are disabled, still display LiveView (without annotations)

These are good starting values for most teams.

Telemetry Method

After DS START is touched, the OpMode continuously calls this method to display telemetry about any TFOD recognitions:

```
/**
 * Add telemetry about TensorFlow Object Detection (TFOD) recognitions.
 */
private void telemetryTfod() {

    List<Recognition> currentRecognitions = tfod.getRecognitions();
    telemetry.addData("# Objects Detected", currentRecognitions.size());

    // Step through the list of recognitions and display info for each one.
    for (Recognition recognition : currentRecognitions) {
        double x = (recognition.getLeft() + recognition.getRight()) / 2 ;
        double y = (recognition.getTop()  + recognition.getBottom()) / 2 ;
```

(continues on next page)

(continued from previous page)

```
        telemetry.addData(""," ");
        telemetry.addData("Image", "%s (%.0f %% Conf.)", recognition.getLabel(), recognition.
↪getConfidence() * 100);
        telemetry.addData("- Position", "%.0f / %.0f", x, y);
        telemetry.addData("- Size", "%.0f x %.0f", recognition.getWidth(), recognition.getHeight());
    }   // end for() loop

}   // end method telemetryTfod()
```

In the first line of code, **all TFOD recognitions** are collected and stored in a List variable. The camera might "see" more than one game element in its field of view, even if not intended (i.e. for CENTERSTAGE with 1 game element).

The `for() loop` then iterates through that List, handling each item, one at a time. Here the "handling" is simply processing certain TFOD fields for DS Telemetry.

The `for() loop` calculates the pixel coordinates of the **center** of each Bounding Box (the preview's colored rectangle around a recognized object).

Telemetry is created for the Driver Station, with the object's name (Label), recognition confidence level (percentage), and the Bounding Box's location and size (in pixels).

For competition, you want to do more than display Telemetry, and you want to exit the main OpMode loop at some point. These code modifications are discussed in another section below.

Resource Management

Vision processing is "expensive", using much **CPU capacity and USB bandwidth** to process millions of pixels streaming in from the camera.

This Sample OpMode contains two optional features to remind teams about resource management. Overall, the SDK provides *over 10 tools* to manage these resources, allowing your OpMode to run effectively.

As the first example, streaming images from the camera can be paused and resumed. This is a very fast transition, freeing CPU resources (and potentially USB bandwidth).

```
// Save CPU resources; can resume streaming when needed.
if (gamepad1.dpad_down) {
    visionPortal.stopStreaming();
} else if (gamepad1.dpad_up) {
    visionPortal.resumeStreaming();
}
```

Pressing the Dpad buttons, you can observe the off-and-on actions in the RC preview (LiveView), described above. In your competition OpMode, these streaming actions would be programmed, not manually controlled.

The second example: after exiting the main loop, the VisionPortal is closed.

```
// Save more CPU resources when camera is no longer needed.
visionPortal.close();
```

Teams may consider this at any point when the VisionPortal is no longer needed by the OpMode, freeing valuable CPU resources for other tasks.

Adjusting the Zoom Factor

If the object to be recognized will be more than roughly 2 feet (61 cm) from the camera, you might want to set the digital Zoom factor to a value greater than 1. This tells TensorFlow to use an artificially magnified portion of the image, which may offer more accurate recognitions at greater distances.

```
// Indicate that only the zoomed center area of each
// image will be passed to the TensorFlow object
// detector. For no zooming, set magnification to 1.0.
tfod.setZoom(2.0);
```

This `setZoom()` method can be placed in the INIT section of your OpMode,

- immediately after the call to the `initTfod()` method, or
- as the very last command inside the `initTfod()` method.

This method is **not** part of the Processor Builder pattern (used in other TFOD Sample OpModes), so the Zoom factor can be set to other values during the OpMode, if desired.

The “zoomed” region can be observed in the DS preview (Camera Stream) and the RC preview (LiveView), surrounded by a greyed-out area that is **not evaluated** by the TFOD Processor.

Other Adjustments

The Sample OpMode uses a default **minimum confidence** level of 75%. This means the TensorFlow Processor needs a confidence level of 75% or higher, to consider an object as “recognized” in its field of view.

You can see the object name and actual confidence (as a **decimal**, e.g. 0.96) near the Bounding Box, in the Driver Station preview (Camera Stream) and Robot Controller preview (Liveview).

```
// Set the minimum confidence at which to keep recognitions.
tfod.setMinResultConfidence((float) 0.75);
```

Adjust this parameter to a higher value if you would like the processor to be more selective in identifying an object.

Another option is to define, or clip, a **custom area for TFOD evaluation**, unlike `setZoom` which is always centered.

```
// Set the number of pixels to obscure on the left, top,
// right, and bottom edges of each image passed to the
// TensorFlow object detector. The size of the images are not
// changed, but the pixels in the margins are colored black.
tfod.setClippingMargins(0, 200, 0, 0);
```

Adjust the four margins as desired, in units of pixels.

These methods can be placed in the INIT section of your OpMode,

- immediately after the call to the `initTfod()` method, or
- as the very last commands inside the `initTfod()` method.

As with `setZoom`, these methods are **not** part of the Processor Builder pattern (used in other TFOD Sample OpModes), so they can be set to other values during the OpMode, if desired.

Modifying the Sample

In this Sample OpMode, the main loop ends only when the DS STOP button is touched. For competition, teams should **modify this code** in at least two ways:

- for a significant recognition, take action or store key information – inside the `for() loop`
- end the main loop based on your criteria, to continue the OpMode

As an example, you might set a Boolean variable `isPixelDetected` to `true`, if a significant recognition has occurred.

You might also evaluate and store which randomized Spike Mark (red or blue tape stripe) holds the white Pixel.

Regarding the main loop, it could end after the camera views all three Spike Marks, or after your code provides a high-confidence result. If the camera's view includes more than one Spike Mark position, perhaps the white Pixel's **Bounding Box** size and location could be useful. Teams should consider how long to seek an acceptable recognition, and what to do otherwise.

In any case, the OpMode should exit the main loop and continue running, using any stored information.

Best of luck this season!

Questions, comments and corrections to westsiderobotics@verizon.net

21.4.7 Java Custom Model Sample OpMode for TFOD

Introduction

This tutorial describes the regular, or **Builder**, version of the FTC Java **Sample OpMode** for TensorFlow Object Detection (TFOD).

This Sample, called **"ConceptTensorFlowObjectDetection.java"**, can recognize **official or custom** FTC game elements and provide their visible size and position. It uses the Java **Builder pattern** to customize standard/default TFOD settings.

This is **not the same** as the "Easy" version, which uses only default settings and official/built-in TFOD model(s), described here:

- *Java Easy Sample OpMode for TFOD*

For the 2023-2024 game CENTERSTAGE, the official game element is a hexagonal white **Pixel**. The FTC SDK software contains a TFOD model of this object, ready for recognition.

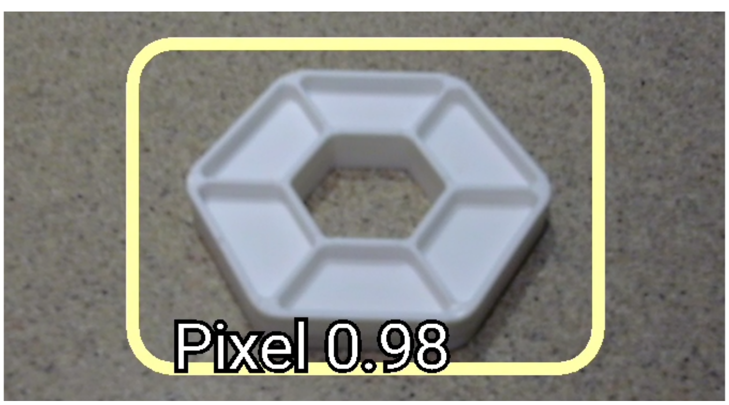


Fig. 148: Example Pixel Recognition using TFOD

For extra points, FTC teams may instead use their own custom TFOD models of game elements, called **Team Props** in CENTERSTAGE.

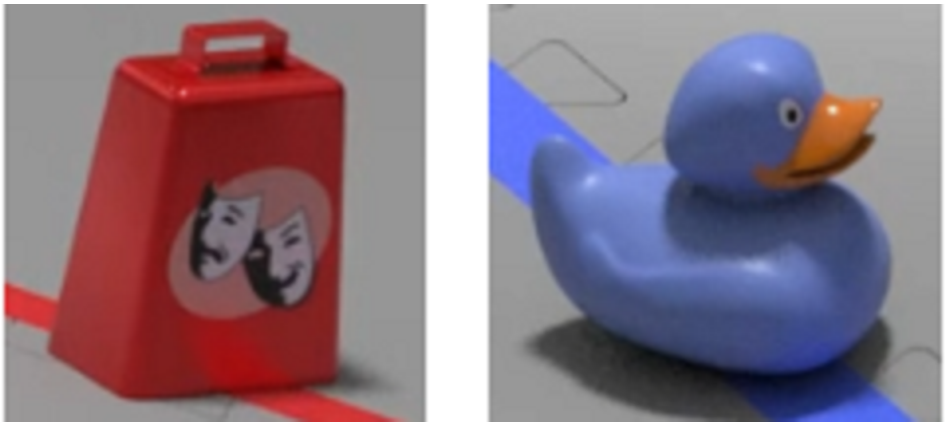


Fig. 149: Example Team Props

This tutorial shows **OnBot Java** screens. Users of **Android Studio** can follow along with a few noted exceptions, since the Sample OpMode is exactly the same.

Creating the OpMode

At the FTC **OnBot Java** browser interface, click on the large black **plus-sign icon** "Add File", to open the New File dialog box.

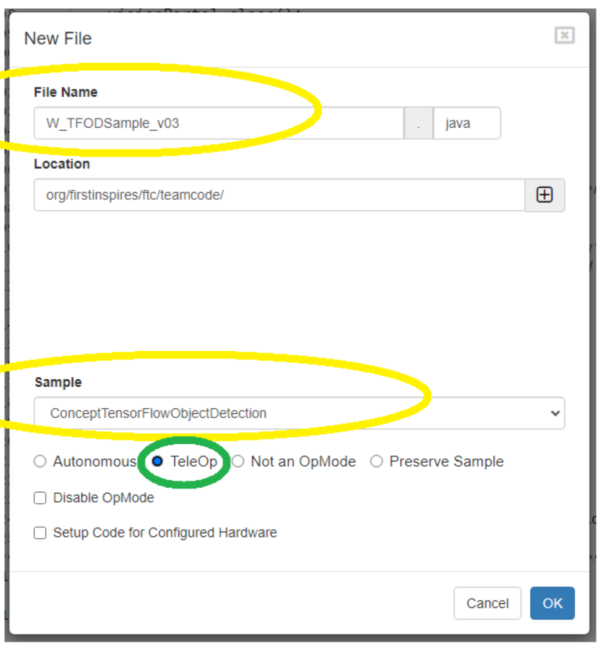


Fig. 150: Example New File Dialog

Specify a name for your new OpMode. Select **"ConceptTensorFlowObjectDetection"** as the Sample OpMode to be the template for your new OpMode.

This Sample has optional gamepad inputs, so it could be designated as a **TeleOp** OpMode (see green oval above).

Click "OK" to create your new OpMode.

Android Studio users should follow the commented instructions to copy this class from the Samples folder to the Teamcode folder, with a new name. Also remove the `@Disabled` annotation, to make the OpMode visible in the Driver Station list.

The new OpMode should appear in the editing window of OnBot Java.

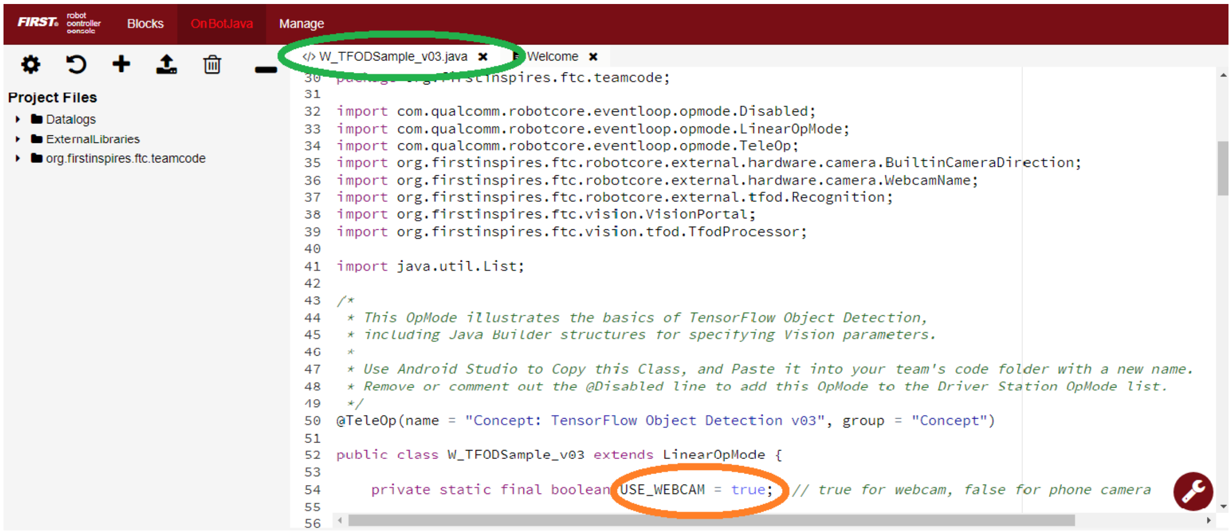


Fig. 151: Sample Open Dialog

By default, the Sample OpMode assumes you are using a webcam, configured as “Webcam 1”. If instead you are using the built-in camera on your Android RC phone, change the USE_WEBCAM Boolean from `true` to `false` (orange oval above).

Preliminary Testing

This Sample OpMode is **ready to use**, for detecting the default/built-in model (white Pixel for CENTERSTAGE).

If **Android Studio** users get a DS error message “Loading model from asset failed”, skip to the next section “Downloading the Model”.

Click the “Build Everything” button (wrench icon at lower right), and wait for confirmation “BUILD SUCCESSFUL”.

If Build is prevented by some other OpMode having errors/issues, they must be fixed before your new OpMode can run. For a quick fix, you could right-click on that filename and choose “Disable/Comment”. This “comments out” all lines of code, effectively removing that file from the Build. That file can be re-activated later with “Enable/Uncomment”.

In Android Studio (or OnBot Java), you can open a problem class/OpMode and type **CTRL-A** and **CTRL-/** to select and “comment out” all lines of code. This is reversible with **CTRL-A** and **CTRL-/** again.

Now run your new OpMode from the Driver Station (in the TeleOp list, if so designated). The OpMode should recognize any CENTERSTAGE white Pixel within the camera’s view, based on the trained TFOD model.

For a **preview** during the INIT phase, touch the Driver Station’s 3-dot menu and select **Camera Stream**.

Camera Stream is not live video; tap to refresh the image. Use the small white arrows at bottom right to expand or revert the preview size. To close the preview, choose 3-dots and Camera Stream again.

After the DS START button is touched, the OpMode displays Telemetry for any recognized Pixel(s):

The above Telemetry shows the Label name, and TFOD recognition confidence level. It also gives the **center location** and **size** (in pixels) of the Bounding Box, which is the colored rectangle surrounding the recognized object.

The pixel origin (0, 0) is at the top left corner of the image.

Before and after DS START is touched, the Robot Controller provides a video preview called **LiveView**.

For Control Hub (with no built-in screen), plug in an HDMI monitor or learn about `scrcpy` (https://github.com/Genymobile/scrcpy). The above image is a LiveView screenshot via `scrcpy`.

If you don’t have a physical Pixel on hand, try pointing the camera at this image:

Congratulations! At this point the Sample OpMode and your camera are working properly. Ready for a custom model?

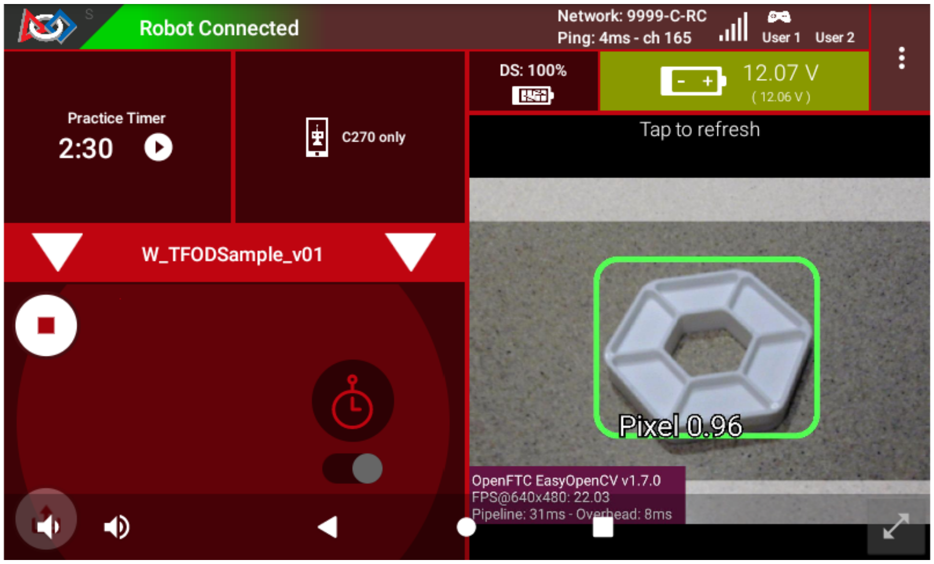


Fig. 152: Sample DS Camera Stream

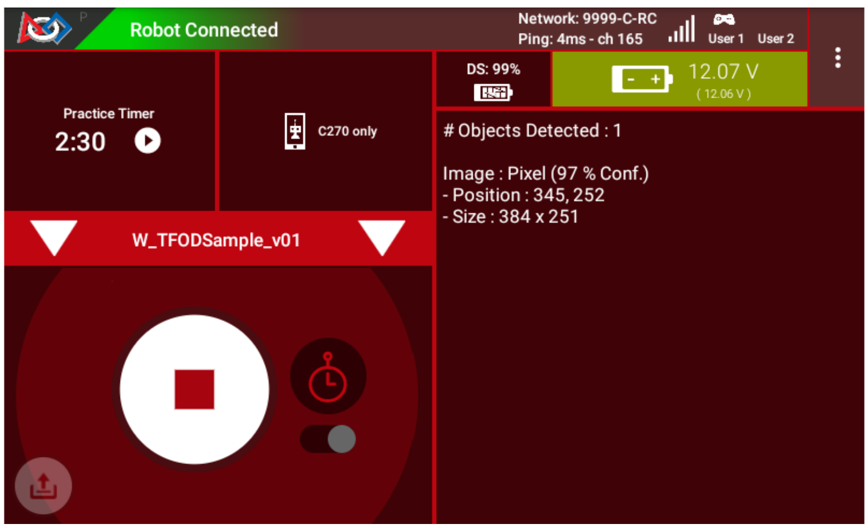


Fig. 153: Sample DS Telemetry

Fig. 154: Sample RC LiveView

Fig. 155: Sample Pixel

Downloading the Model

Now we describe how to load a trained inference model in the form of a TensorFlow Lite (`.tflite`) file.

Instead of an **actual custom model**, here we use the standard FTC model of the white Pixel from CENTERSTAGE (2023-2024). Later, your team will follow this **same process** with your custom TFOD model, specifying its filename and labels (objects to recognize).

The standard `.tflite` file (white Pixel) is available on GitHub at the following link:

- CENTERSTAGE TFLite File (https://github.com/FIRST-Tech-Challenge/WikiSupport/blob/master/tensorflow/CenterStage.tflite)

Note: Very advanced teams could use Google's TensorFlow Object Detection API (https://github.com/tensorflow/models/tree/master/research/object_detection) to create their own custom inference model.

Click the "Download Raw File" button to download the `CenterStage.tflite` file from GitHub to your local device (e.g. laptop). See the green arrow.

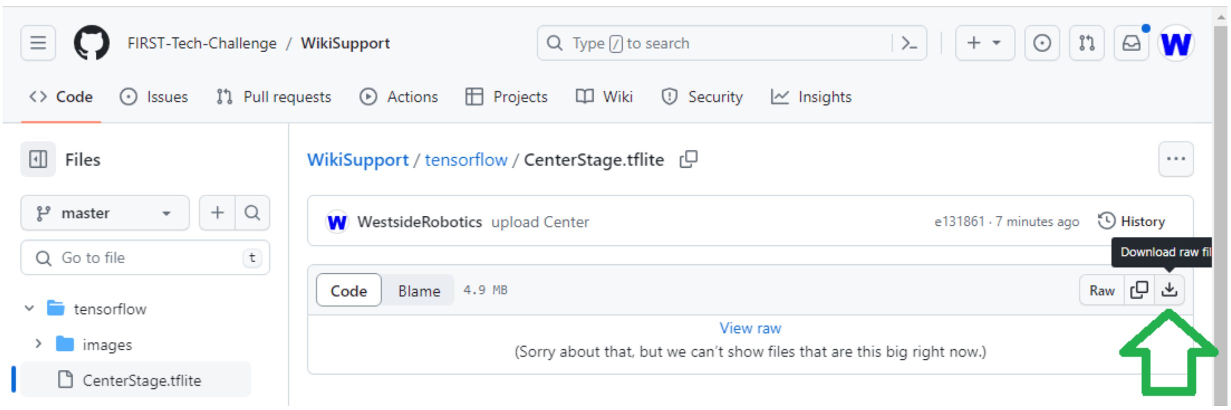


Fig. 156: Public Repo

Uploading to the Robot Controller

Next, OnBot Java users will upload the TFOD model to the Robot Controller. Connect your laptop to your Robot Controller's wireless network, open the Chrome browser, and navigate to the FTC "Manage" page:

Android Studio users should instead skip to the instructions at the bottom of this section.

Scroll down and click on "Manage TensorFlow Lite Models".

Now click the "Upload Models" button.

Click "Choose Files", and use the dialog box to find and select the downloaded `CenterStage.tflite` file.

Now the file will upload to the Robot Controller. The file will appear in the list of TensorFlow models available for use in OpModes.

Android Studio users should instead store the TFOD model in the project **assets** folder. At the left side, look under `FtcRobotController` for the folder `assets`. If it's missing, right-click `FtcRobotController`, choose `New`, `Directory` and `src\main\assets`. Right-click `assets`, choose `Open In` and `Explorer`, then copy/paste your `.tflite` file into that assets folder.

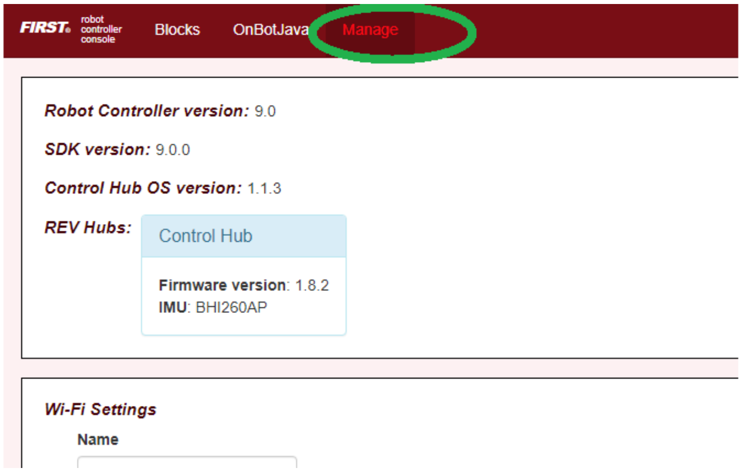


Fig. 157: Robot Controller Manage Page

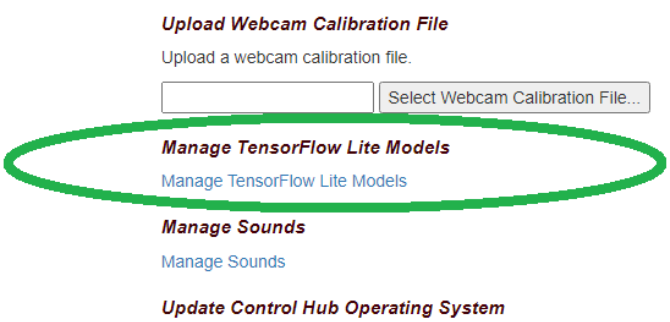


Fig. 158: TensorFlow Lite Model Management

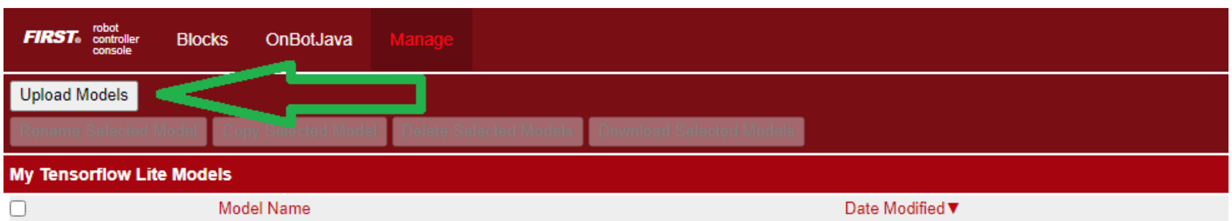


Fig. 159: Upload Models

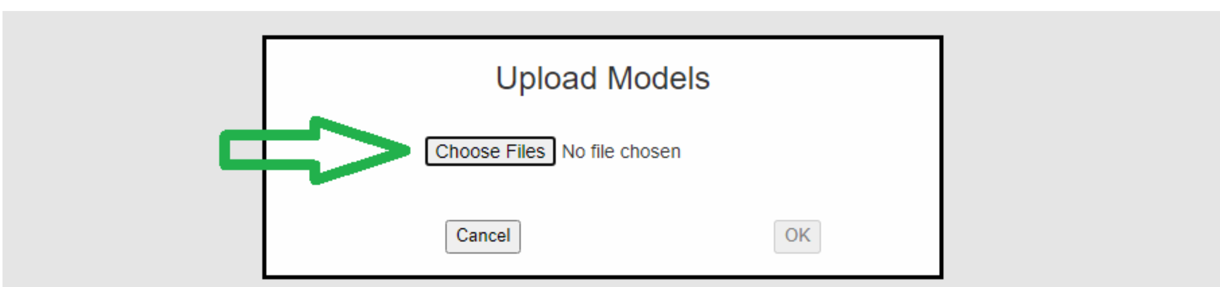


Fig. 160: Choose Files

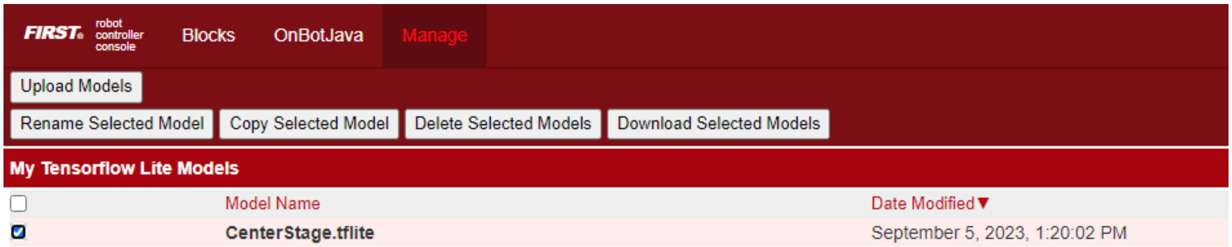


Fig. 161: CENTERSTAGE TFLITE File Uploaded

Basic OpMode Settings

This Sample OpMode can now be modified, to detect the uploaded TFOD model.

Again, this tutorial uploaded the standard TFOD model (white Pixel for CENTERSTAGE), just to demonstrate the process. Use the same steps for your custom TFOD model.

First, change the filename here:

```
private static final String TFOD_MODEL_FILE = "/sdcard/FIRST/tflitemodels/myCustomModel.tflite";
```

to this:

```
private static final String TFOD_MODEL_FILE = "/sdcard/FIRST/tflitemodels/CenterStage.tflite";
```

Later, you can change this filename back to the actual name of your custom TFOD model. Here we are using the default (white Pixel) model just downloaded.

Android Studio users should instead verify or store the TFOD model in the project **assets** folder as noted above, and use:

```
private static final String TFOD_MODEL_ASSET = "CenterStage.tflite";
```

OR (for a custom model)

```
private static final String TFOD_MODEL_ASSET = "MyModelStoredAsAsset.tflite";
```

For this example, the following line **does not** need to be changed:

```
// Define the labels recognized in the model for TFOD (must be in training order!)
private static final String[] LABELS = {
   "Pixel",
};
```

… because "Pixel" is the correct and only TFOD Label in the standard model file.

Later, you might have custom Labels like "myRedProp" and "myBlueProp" (for CENTERSTAGE). The list should be in alphabetical order and contain the labels in the dataset(s) used to make the TFOD model.

Next, scroll down to the Java method `initTfod()`.

Here is the Java **Builder pattern**, used to specify various settings for the TFOD Processor.

The **yellow ovals** indicate its distinctive features: **create** the Processor object with new `Builder()`, and **close/finalize** with the `.build()` method.

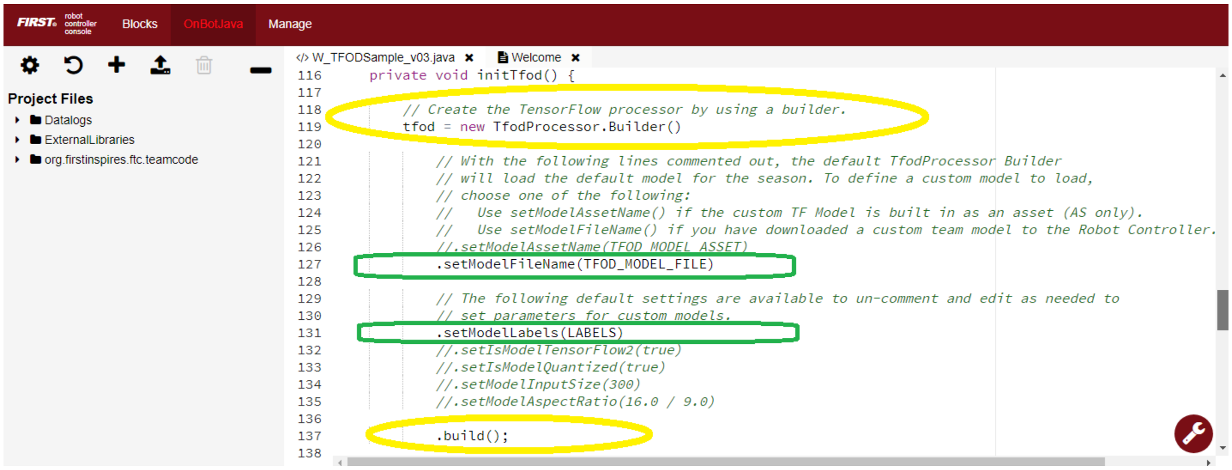


Fig. 162: Builder Pattern Settings

This is the streamlined version of the Builder pattern. Notice all the `.set` methods are "chained" to form a single Java expression, ending with a semicolon after `.build()`.

Uncomment two Builder lines, circled above in green:

```
.setModelFileName(TFOD_MODEL_FILE)
.setModelLabels(LABELS)
```

Android Studio users should instead uncomment the lines `.setModelAssetName(TFOD_MODEL_ASSET)` and `.setModelLabels(LABELS)`.

These Builder settings tell the TFOD Processor which model and labels to use for evaluating camera frames.

That's it! You are ready to test this Sample OpMode again, this time using a "custom" (uploaded) TFOD model.

Testing with Custom Model

In OnBot Java, click the "Build Everything" button (wrench icon at lower right), and wait for confirmation "BUILD SUCCESSFUL".

Now run your updated OpMode from the Driver Station. The OpMode should recognize objects within the camera's view, based on the trained TFOD model.

Test the **Camera Stream** preview during the INIT phase.

Tap to refresh the image. Expand or revert the preview size as needed. Close the preview, with 3-dots and Camera Stream again.

After the DS START button is touched, the OpMode displays Telemetry for any recognized object(s):

The above Telemetry shows the Label name, and TFOD recognition confidence level. It also gives the **center location** and **size** (in pixels) of the Bounding Box, which is the colored rectangle surrounding the recognized object.

Also test the RC's video **LiveView**, using HDMI or `scrcpy` (https://github.com/Genymobile/scrcpy):

For a large view of this standard model, right-click the image to open in a new browser tab:

When your team creates, uploads and specifies a custom model containing **red and blue Team Props**, the OpMode will recognize and process those – instead of the standard model shown here.

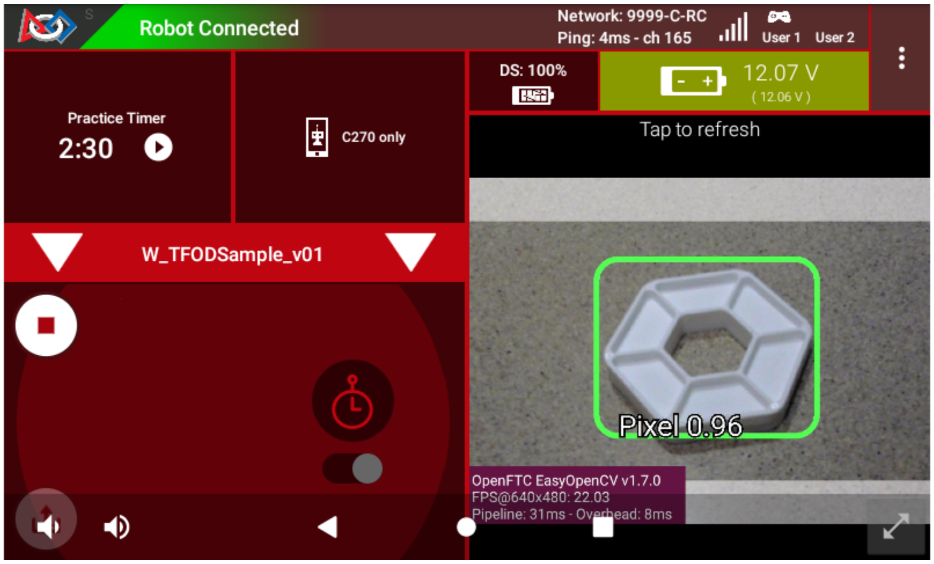


Fig. 163: Sample DS Camera Stream

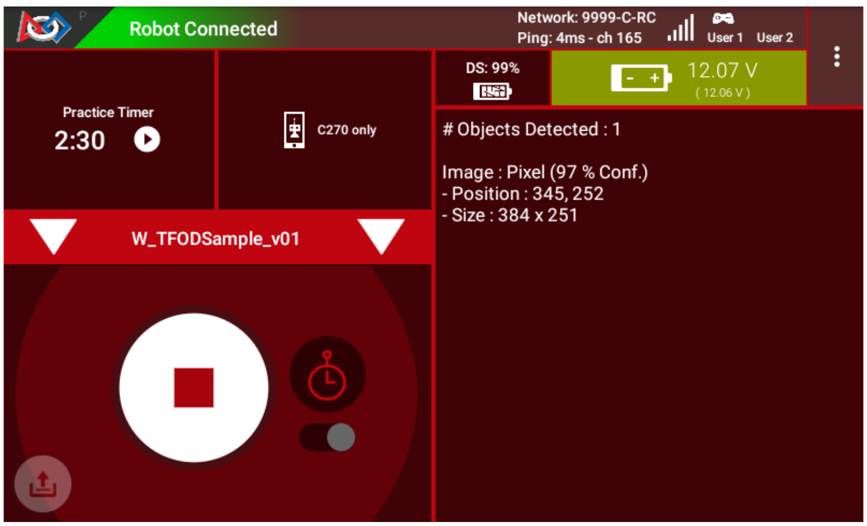


Fig. 164: Sample DS Telemetry

Fig. 165: Sample RC LiveView

Fig. 166: Sample Pixel

Program Logic and Initialization

How does this simple OpMode work?

- During the INIT stage (before DS START is touched), this OpMode calls a **method to initialize** the TFOD Processor and the FTC VisionPortal.
- After DS START is touched, the OpMode runs a continuous loop, calling a **method to display telemetry** about any TFOD recognitions.
- The OpMode also contains optional features to remind teams about **CPU resource management**, useful in vision processing.

You've already seen the first part of the method `initTfod()` which uses a streamlined, or "chained", sequence of Builder commands to create the TFOD Processor.

The second part of that method uses regular, non-chained, Builder commands to create the VisionPortal.

```
// Create the vision portal by using a builder.
VisionPortal.Builder builder = new VisionPortal.Builder();

// Set the camera (webcam vs. built-in RC phone camera).
if (USE_WEBCAM) {
   builder.setCamera(hardwareMap.get(WebcamName.class, "Webcam 1"));
} else {
   builder.setCamera(BuiltinCameraDirection.BACK);
}

// Choose a camera resolution. Not all cameras support all resolutions.
builder.setCameraResolution(new Size(640, 480));

// Enable the RC preview (LiveView).  Set "false" to omit camera monitoring.
builder.enableLiveView(true);

// Set the stream format; MJPEG uses less bandwidth than default YUY2.
builder.setStreamFormat(VisionPortal.StreamFormat.YUY2);

// Choose whether or not LiveView stops if no processors are enabled.
// If set "true", monitor shows solid orange screen if no processors enabled.
// If set "false", monitor shows camera view without annotations.
builder.setAutoStopLiveView(false);

// Set and enable the processor.
builder.addProcessor(tfod);

// Build the Vision Portal, using the above settings.
visionPortal = builder.build();
```

All settings have been uncommented here, to see them more easily.

Here the `new Builder()` creates a separate `VisionPortal.Builder` object called `builder`, allowing traditional/individual Java method calls for each setting. For the streamlined "chained" TFOD process, the `new Builder()` operated directly on the TFOD Processor called `tfod`, without creating a `TfodProcessor.Builder` object. Both approaches are valid.

Notice the process again **closes** with a call to the `.build()` method.

Telemetry Method

After DS START is touched, the OpMode continuously calls this method to display telemetry about any TFOD recognitions:

```
/**
  * Add telemetry about TensorFlow Object Detection (TFOD) recognitions.
  */
private void telemetryTfod() {

    List<Recognition> currentRecognitions = tfod.getRecognitions();
    telemetry.addData("# Objects Detected", currentRecognitions.size());

    // Step through the list of recognitions and display info for each one.
    for (Recognition recognition : currentRecognitions) {
        double x = (recognition.getLeft() + recognition.getRight()) / 2 ;
        double y = (recognition.getTop()  + recognition.getBottom()) / 2 ;

        telemetry.addData(""," ");
        telemetry.addData("Image", "%s (%.0f %% Conf.)", recognition.getLabel(), recognition.
↪getConfidence() * 100);
        telemetry.addData("- Position", "%.0f / %.0f", x, y);
        telemetry.addData("- Size", "%.0f x %.0f", recognition.getWidth(), recognition.getHeight());
    }   // end for() loop

}   // end method telemetryTfod()
```

In the first line of code, **all TFOD recognitions** are collected and stored in a List variable. The camera might "see" more than one game element in its field of view, even if not intended (i.e. for CENTERSTAGE with 1 game element).

The `for() loop` then iterates through that List, handling each item, one at a time. Here the "handling" is simply processing certain TFOD fields for DS Telemetry.

The `for() loop` calculates the pixel coordinates of the **center** of each Bounding Box (the preview's colored rectangle around a recognized object).

Telemetry is created for the Driver Station, with the object's name (Label), recognition confidence level (percentage), and the Bounding Box's location and size (in pixels).

For competition, you want to do more than display Telemetry, and you want to exit the main OpMode loop at some point. These code modifications are discussed in another section below.

Resource Management

Vision processing is "expensive", using much **CPU capacity and USB bandwidth** to process millions of pixels streaming in from the camera.

This Sample OpMode contains three optional features to remind teams about resource management. Overall, the SDK provides *over 10 tools* to manage these resources, allowing your OpMode to run effectively.

As the first example, **streaming images** from the camera can be paused and resumed. This is a very fast transition, freeing CPU resources (and potentially USB bandwidth).

```
// Save CPU resources; can resume streaming when needed.
if (gamepad1.dpad_down) {
   visionPortal.stopStreaming();
} else if (gamepad1.dpad_up) {
   visionPortal.resumeStreaming();
}
```

Pressing the Dpad buttons, you can observe the off-and-on actions in the RC preview (LiveView), described above. In your competition OpMode, these streaming actions would be programmed, not manually controlled.

The second example, commented out, similarly allows a vision processor (TFOD and/or AprilTag) to be disabled and re-enabled:

```
//Disable or re-enable the TFOD processor at any time.
visionPortal.setProcessorEnabled(tfod, true);
```

Simply set the Boolean to `false` (to disable), or `true` (to re-enable).

The third example: after exiting the main loop, the VisionPortal is closed.

```
// Save more CPU resources when camera is no longer needed.
visionPortal.close();
```

Teams may consider this at any point when the VisionPortal is no longer needed by the OpMode, freeing valuable CPU resources for other tasks.

Adjusting the Zoom Factor

If the object to be recognized will be more than roughly 2 feet (61 cm) from the camera, you might want to set the digital Zoom factor to a value greater than 1. This tells TensorFlow to use an artificially magnified portion of the image, which may offer more accurate recognitions at greater distances.

```
// Indicate that only the zoomed center area of each
// image will be passed to the TensorFlow object
// detector. For no zooming, set magnification to 1.0.
tfod.setZoom(2.0);
```

This `setZoom()` method can be placed in the INIT section of your OpMode,

- immediately after the call to the `initTfod()` method, or
- as the very last command inside the `initTfod()` method.

This method is **not** part of the TFOD Processor Builder pattern, so the Zoom factor can be set to other values during the OpMode, if desired.

The "zoomed" region can be observed in the DS preview (Camera Stream) and the RC preview (LiveView), surrounded by a greyed-out area that is **not evaluated** by the TFOD Processor.

Other Adjustments

This Sample OpMode contains another adjustment, commented out:

```
// Set confidence threshold for TFOD recognitions, at any time.
tfod.setMinResultConfidence(0.75f);
```

The SDK uses a default **minimum confidence** level of 75%. This means the TensorFlow Processor needs a confidence level of 75% or higher, to consider an object as "recognized" in its field of view.

You can see the object name and actual confidence (as a **decimal**, e.g. 0.96) near the Bounding Box, in the Driver Station preview (Camera Stream) and Robot Controller preview (Liveview).

Adjust this parameter to a higher value if you want the processor to be more selective in identifying an object.

Another option is to define, or clip, a **custom area for TFOD evaluation**, unlike `setZoom` which is always centered.

```
// Set the number of pixels to obscure on the left, top,
// right, and bottom edges of each image passed to the
// TensorFlow object detector. The size of the images are not
// changed, but the pixels in the margins are colored black.
tfod.setClippingMargins(0, 200, 0, 0);
```

Adjust the four margins as desired, in units of pixels.

These method calls can be placed in the INIT section of your OpMode,

- immediately after the call to the `initTfod()` method, or
- as the very last commands inside the `initTfod()` method.

As with `setProcessorEnabled()` and `setZoom()`, these methods are **not** part of the Processor or VisionPortal Builder patterns, so they can be set to other values during the OpMode, if desired.

Modifying the Sample

In this Sample OpMode, the main loop ends only when the DS STOP button is touched. For CENTERSTAGE competition, teams should **modify this code** in at least two ways:

- for a significant recognition, take action or store key information – inside the `for() loop`
- end the main loop based on your criteria, to continue the OpMode

As an example, you might set a Boolean variable `isPixelDetected` (or `isPropDetected`) to `true`, if a significant recognition has occurred.

You might also evaluate and store which randomized Spike Mark (red or blue tape stripe) holds the white Pixel or Team Prop.

Regarding the main loop, it could end after the camera views all three Spike Marks, or after your code provides a high-confidence result. If the camera's view includes more than one Spike Mark position, perhaps the Pixel/Prop's **Bounding Box** size and location could be useful. Teams should consider how long to seek an acceptable recognition, and what to do otherwise.

In any case, the OpMode should exit the main loop and continue running, using any stored information.

Best of luck this season!

Questions, comments and corrections to westsiderobotics@verizon.net

21.5 Vision Programming

Learning more about using vision

21.5.1 Computer Vision Overview

Introduction

The *FIRST* Tech Challenge control system software has built-in support for two computer vision technologies:

1. AprilTags - *AprilTags* are fiducial markers similar in design to a QR code that can be used for identification and localization. AprilTags are used as reference points for autonomous navigation and for assisted navigation and identification of points of interest on a game field.
    - Each season, FIRST provides 2D image tagets that can be used as navigational reference points.

- If the AprilTag system recognizes an AprilTag image, it provides very accurate pose information (assuming the camera used has calibration parameters for the working resolution) about the robot's position relative to the target.
- A robot can use this information to navigate autonomously on the field.

2. TensorFlow Lite - TensorFlow Lite is a lightweight version of Google's TensorFlow *machine learning* technology that is designed to run on mobile devices such as an Android smartphone.
   - Each season FIRST creates a TensorFlow *inference model* that can be used to "look" for specific game elements.
   - If TensorFlow recognizes an object, it returns location info about the identified object.
   - A robot can use this location information to navigate to the recognized object.

TensorFlow vs AprilTags

AprilTag Advantages

- Very efficient with a fast detection rate (estimated 15 to 20 detections per second, depending on decimation and target size).
- Provides accurate, relative pose information of camera to target in field coordinates.
- Is less prone to fluctuating or varied lighting conditions on the field.

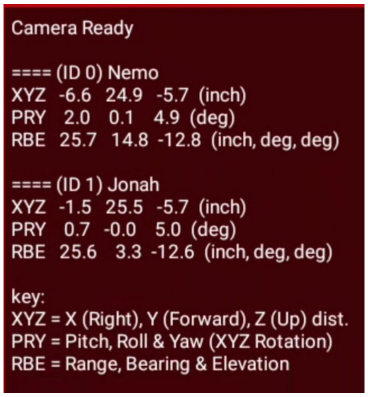


Fig. 167: AprilTag can provide accurate pose information to target

AprilTag Disadvantages

- The entire AprilTag must be in the camera view in order to be recognized, any occlusions render the object unprocessable.
- AprilTags must be included in the tag library in order to process pose information for the tag (tag size and value must be known to the AprilTag system in advance).
- Cameras require calibration data for every resolution used in order to process correct pose information.

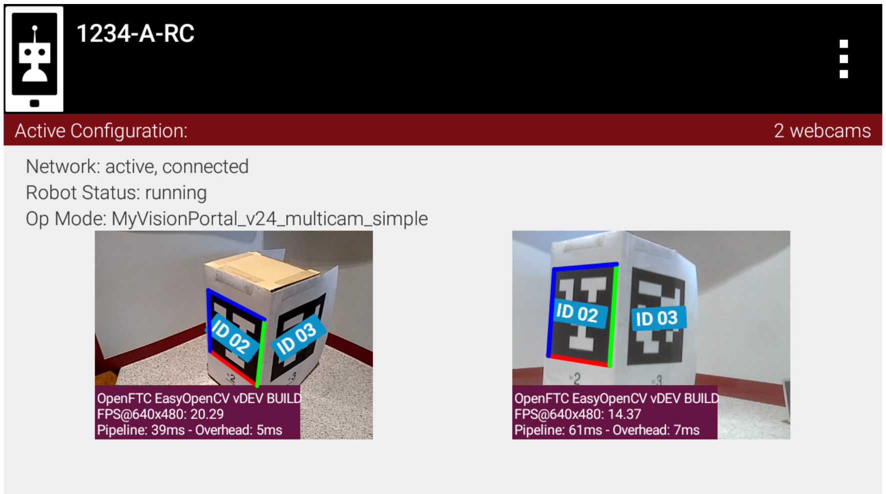


Fig. 168: AprilTags not in Tag Library detected, but no pose data available

TensorFlow Advantages

- TensorFlow learns how to recognize target objects, not just specific images.
  - Recognizes objects in spite of different backgrounds.
  - Recognizes objects in varied lighting conditions.
  - Recognizes objects even when objects are oriented in different positions.
- TensorFlow can be taught how to distinguish between similar looking (but still distinct) objects, such as a Stone and a Skystone from the 2019-2020 challenge.

TensorFlow Disadvantages

- Training a TensorFlow model can be daunting at first. It requires a lot of understanding of the TensorFlow training metrics and behaviors.
- TensorFlow is computationally intensive and has a low detection rate (an estimated 1 to 2 detections per second).
- If TensorFlow recognizes an object in its field of view, it only returns location information on where the target object is within its field of view.

Which Should I Use?

The choice of whether to use TensorFlow Lite or AprilTags will be influenced by factors such as distance-to-target, lighting, accuracy required, camera placement and etc..

If the object and tag can always be guaranteed to be in a specific orientation and the tag fully visible, AprilTags are likely the best solution. However, if the object does not belong to you or a tag is not able to be physically placed on the object, TensorFlow can be a good solution.

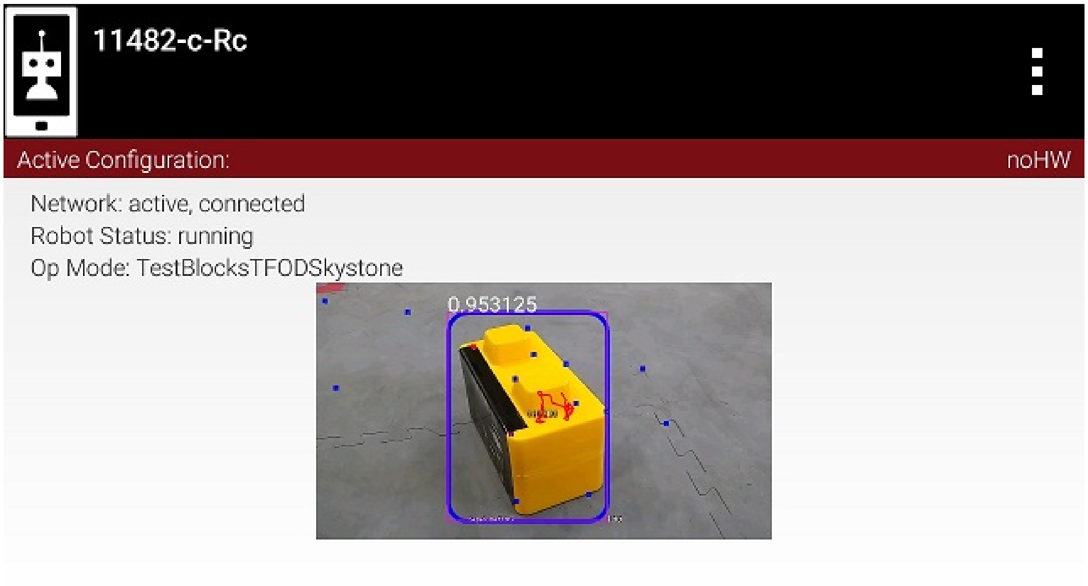


Fig. 169: TensorFlow can recognize actual objects (and not just 2D image targets).

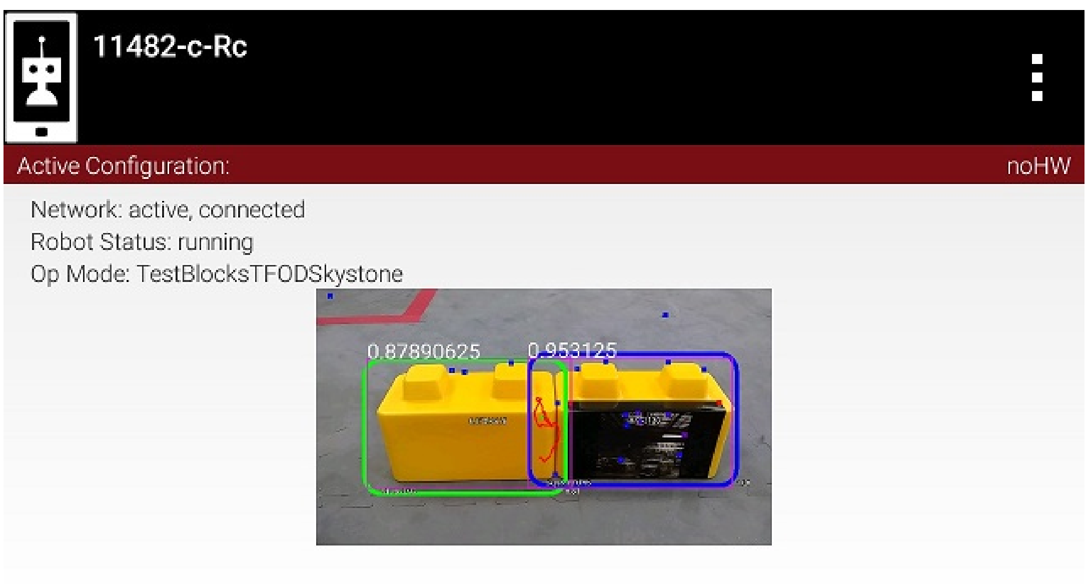


Fig. 170: TensorFlow can be taught to distinguish between similar looking objects.

21.5.2 Webcam Controls

This basic tutorial describes 8 webcam controls available in the SDK. It includes an example, using 2 of these controls to potentially improve TensorFlow recognition in Freight Frenzy.

Hats off to rgatkinson and Windwoes who developed these webcam controls.

Software Overview

The SDK contains a superinterface called CameraControl, which contains 5 interfaces:

- *ExposureControl*
- *GainControl*
- *WhiteBalanceControl* (new for SDK 7.1)
- *FocusControl*
- *PtzControl*

Similar to Java classes, Java interfaces provide methods. A webcam can be controlled using methods of these 5 interfaces.

PtzControl allows control of 3 related features: virtual pan, tilt and zoom. ExposureControl also contains a feature called auto-exposure priority, or AE Priority. Together there are **8 webcam controls** discussed in this tutorial.

The official documentation is found in the Javadocs. Click the link for **RobotCore**, then click the **CameraControl** link in the left column.

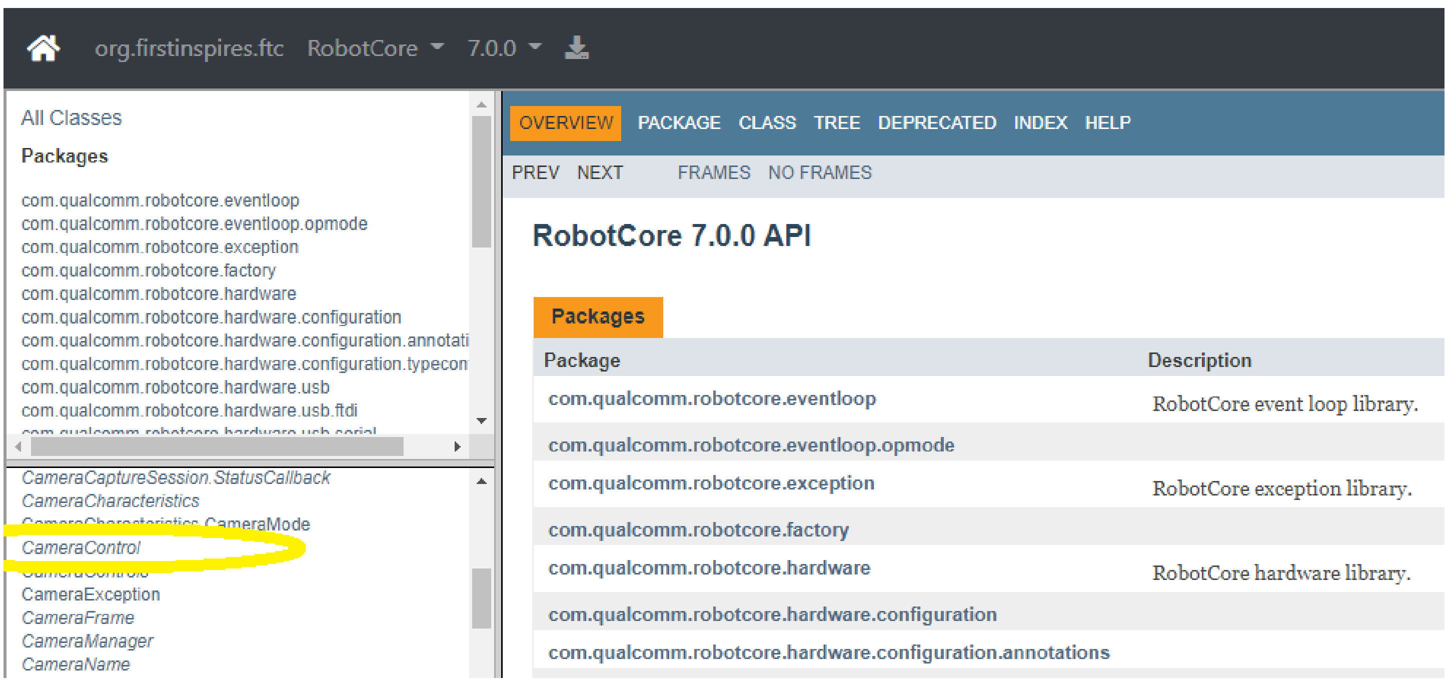


Fig. 171: RobotCore Javadoc API

That page provides links to the 5 interfaces listed above.

The methods described here can be used in Android Studio or OnBot Java. They can also be provided to Blocks programmers by creating myBlocks, covered in a separate *Blocks programming Tutorial*.

You will see Vuforia mentioned here, and in the *sample OpModes* below. **Why Vuforia?** The *FIRST* Tech Challenge implementation of Google's TensorFlow Lite receives camera images from a Vuforia video stream. The SDK already includes and uses Vuforia for navigation, so it's a convenient tool for passing camera streams to TFOD.

These CameraControl interfaces allow some control of the webcam, within requirements or settings of Vuforia for its own performance. Such settings include resolution and frame rate, not covered here.

Exposure - Webcam Controls

Exposure Control

Exposure is the amount of light that reaches the webcam sensor. It is an important part of how bright or dark your image appears.

Exposure varies directly with the amount of time that the shutter is open, allowing light to enter and reach the sensor. So, the interface ExposureControl uses a single value of **duration**, in units of time that you specify, typically `TimeUnit.MILLISECONDS`.

For example, at a frame rate of 60 frames per second (fps), exposure duration is 1/60 of a second, or 1/60 x 1000 = 16 milliseconds. This basic tutorial does not address frame rate.

Here are the methods to manage exposure:

- setExposure() has two parameters: duration and time unit
- getExposure() has one parameter: time unit

The webcam may support minimum and maximum allowed values of exposure. These can be retrieved with:

- getMinExposure(TimeUnit.MILLISECONDS)
- getMaxExposure(TimeUnit.MILLISECONDS)

There are no `set()` methods for min and max exposure; these are hard-coded in the webcam's firmware. Note that firmware settings may vary among different versions of the same webcam model.

These and other exposure methods are called on an ExposureControl object; sample code is shown below, after Exposure Control Mode.

Exposure Control Mode

org.firstinspires.ftc.robotcore.external.hardware.camera.controls

A webcam may operate in one of various exposure modes.

Many common webcams offer only some of these modes. To directly control the exposure, set the webcam to Manual mode.

The SDK supports these values of ExposureControl.Mode:

- *AperturePriority*
- *Auto*
- *ContinuousAuto*
- *Manual*
- *ShutterPriority*
- *Unknown*

Mode is managed with these ExposureControl methods:

- setMode(ExposureControl.Mode._mode_)
- getMode()

The Logitech C920 and C270 models offer two exposure modes: AperturePriority and Manual.

Exposure Control Code Samples

1. Import the interface. This line is automatically added by OnBot Java when the interface is used (coded).
   - `import org.firstinspires.ftc.robotcore.external.hardware.camera.controls.ExposureControl;`
2. Declare the ExposureControl object, before `runOpMode()`.
   - `ExposureControl myExposureControl;`
3. Assign the Vuforia/TFOD video stream control to your control object, in `runOpMode()`.
   - `myExposureControl = vuforia.getCamera().getControl(ExposureControl.class);`
4. Set the mode to Manual, for direct control.
   - `myExposureControl.setMode(ExposureControl.Mode.Manual);`
5. Set the exposure duration, in this case to 30 milliseconds.
   - `myExposureControl.setExposure(30, TimeUnit.MILLISECONDS);`

See far below for these and other exposure controls illustrated in *Sample OpModes*.

AE Priority

Auto-Exposure Priority is a setting within the ExposureControl interface. It's listed here at the end, not likely to be needed in since it it operates in very low lighting.

What does it do? Imagine that the webcam is operating at its default frame rate, for example 30 frames per second (fps). *Note that frame rate is not covered in this basic tutorial.*

If the webcam's built-in auto-exposure detects that the image is very dark, AE Priority **allows the frame rate to decrease**. This slowdown, or 'undershoot', allows more light per frame, which can 'brighten' the image.

Its methods are:

- setAePriority(boolean priority)
- getAePriority()

These AE Priority methods are called on an ExposureControl object, as described above.

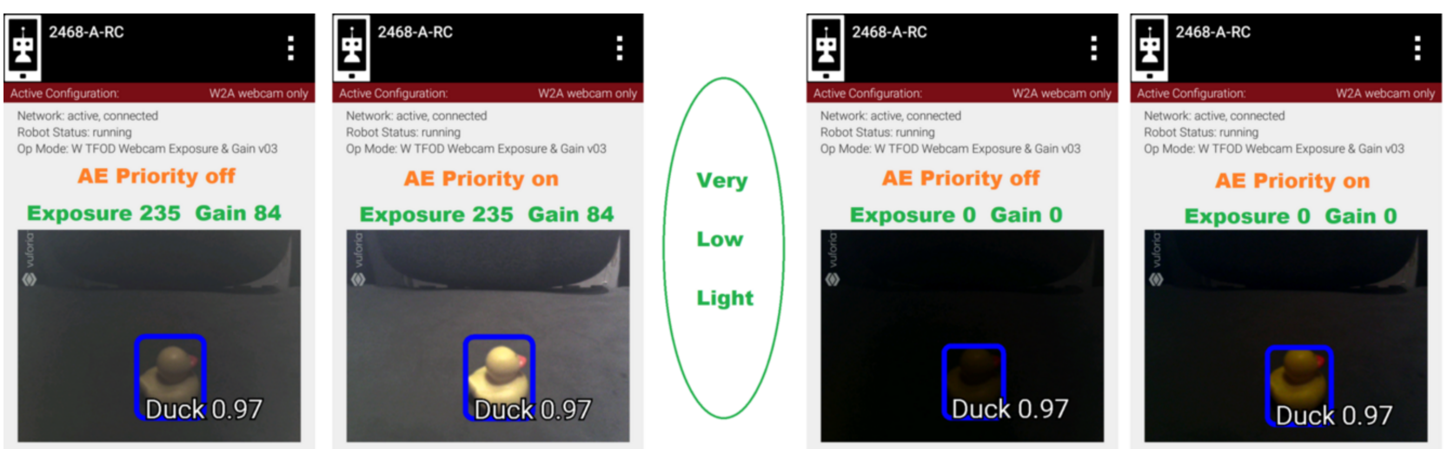


Fig. 172: Two examples of AE Priority

Here are two pairs of previews, each with AE Priority off and on. In both pairs, the ambient light level is very low. These results are from a Logitech C270 webcam.

The Exposure=0 recognition here was made before reducing exposure and gain. When testing 'instant' results, AE Priority could improve the chance of recognition.

Again, this effect is triggered only in very low lighting, not expected in competition. If the building loses all power, Duck recognition becomes… less essential.

Gain - Webcam Controls

Gain Control

org.firstinspires.ftc.robotcore.external.hardware.camera.controls

Gain is a digital camera setting that controls the amplification of the signal from the webcam sensor. This amplifies the whole signal, including any associated background noise.

Gain can be managed in coordination with exposure. Raising exposure and keeping gain low, can provide a bright image and low noise. On the other hand, longer exposure can cause motion blur, which may affect target tracking performance. In some cases, reducing exposure duration and increasing gain may provide a sharper image, although with more noise.

The interface GainControl uses a single value to control gain. It's used for amplification, and thus has no units – it's just a number of type integer. Its methods are:

- setGain(int gain)
- getGain()

As with exposure, the webcam may support minimum and maximum allowed values of gain. These can be retrieved with:

- getMinGain()
- getMaxGain()

There are no `set()` methods for min and max gain; these are hard-coded in the webcam's firmware. Note that firmware settings may vary among different versions of the same webcam model.

These and other gain methods are called on a GainControl object, as described above for exposure.

Example 1: Exposure's effect on TFOD

We interrupt this tutorial to demonstrate the two webcam interfaces described so far: ExposureControl and GainControl.

These 2 examples assume you are already using TensorFlow Object Detection (TFOD) in the Freight Frenzy game. Namely you have a TFOD model and OpMode that are working reasonably well. Here we will discuss only the Duck game element. **Can the exposure and/or gain controls improve the chance of a fast, accurate TFOD detection?**

Another way to frame this effort is: can these controls simulate the lighting conditions used for TFOD model training? Namely, if the competition field has different lighting that affects recognition, can you achieve close to **your original (trained) TFOD performance**?

We first try exposure alone. Setting gain to zero, we apply TFOD to webcam images at various exposure values.

Five fresh readings were taken at each exposure setting. Namely the test OpMode was opened (INIT) each time for a new TFOD initialization and webcam image processing.

This chart shows TFOD confidence levels; 'instant' is defined here as recognition within 1 second.

Higher exposure does improve recognition, then performance suddenly drops. Then at higher levels, this TFOD model begins to "see" a Cube, not a Duck. Not good!

So, there does seem to be a range of exposure values that gives better results. Note the sharp drop-off at both ends of the range: below 25 and above 40. In engineering, a **robust** solution can withstand variation. Using a value in the middle of the improved range, can reduce the effects of unforeseen variation. But this range varies with ambient lighting conditions, which may be quite different at the tournament venue.

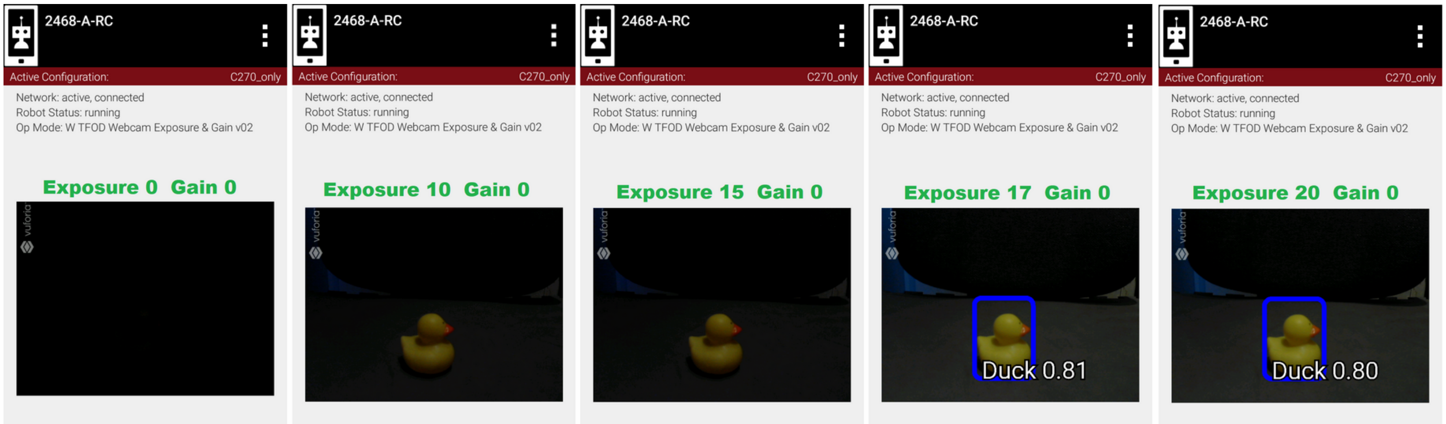


Fig. 173: Gain 0, Exp 0 -> 20

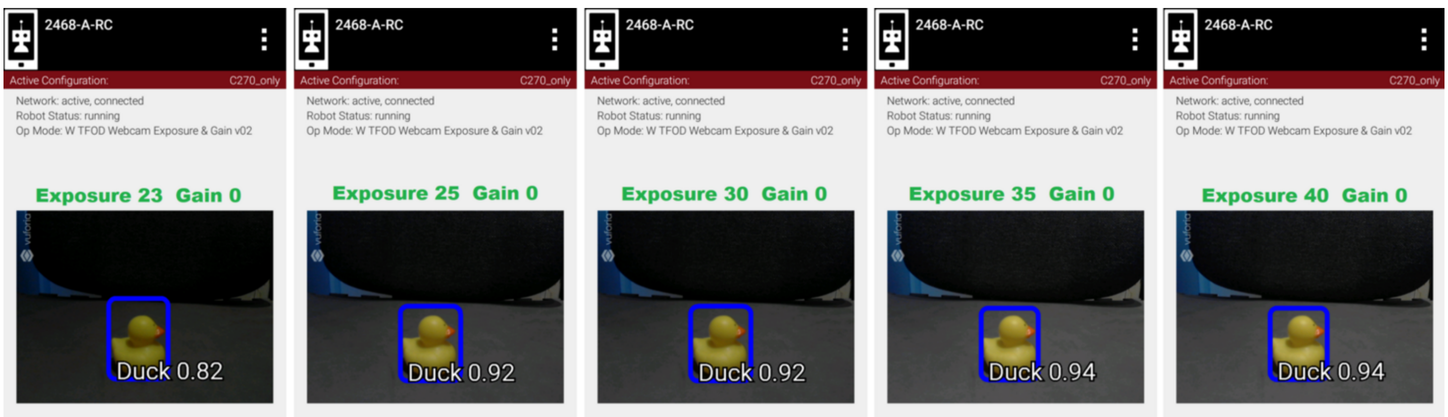


Fig. 174: Gain 0, Exp 23 - > 40

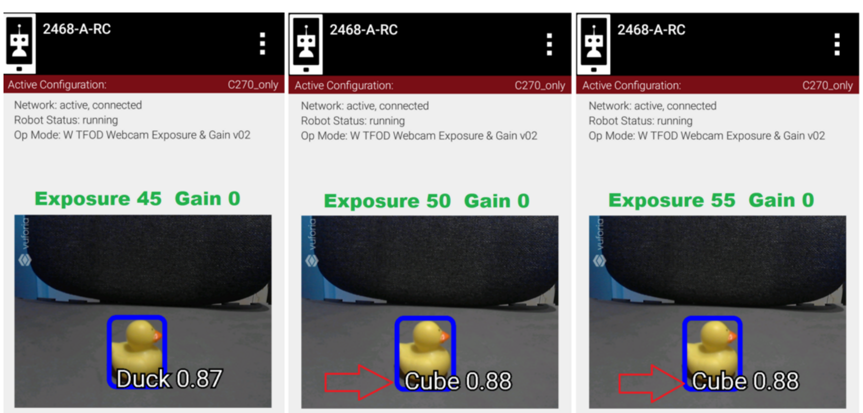


Fig. 175: Gain 0, Exp 45 -> 55

TFOD Duck Confidence, instant (< 1 sec.)					Gain = 0
	Test Round				
Exposure	1	2	3	4	5
0	none	none	none	none	none
10	none	none	none	none	none
15	none	none	none	none	none
17	0.81	none	0.81	0.80	none
20	0.81	0.80	0.80	0.81	0.81
23	0.82	none	0.81	0.80	none
25	0.92	0.92	0.91	0.91	0.91
30	0.92	0.91	0.91	0.92	0.91
35	0.94	0.93	0.92	0.93	0.93
40	0.94	0.93	0.93	0.93	0.92
45	0.87	none	none	none	none
50	Cube .88	Cube .87	Cube .82	Cube .85	Cube .81
55	Cube .88	Cube .87	Cube .87	Cube .89	Cube .88

Fig. 176: Five readings at each exposure level

This data is the result of a very particular combination of: webcam model (Logitech C270), distance (12 inches), lookdown angle (30 degrees), TFOD model (SDK 7.0 default), ambient lighting, background, etc. **Your results will vary, perhaps significantly.**

Example 2: Gain's effect on TFOD

Now we adjust only gain. We set Exposure to a fixed value of 15, selected because it was a poor performer in Example 1. **Can gain help?**

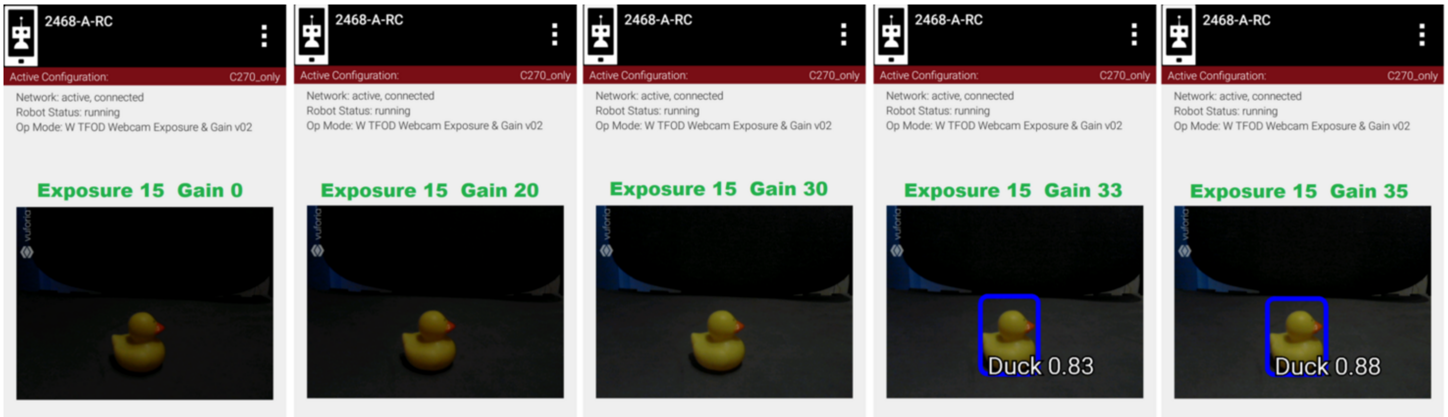


Fig. 177: Exp 15, Gain 000 -> 035

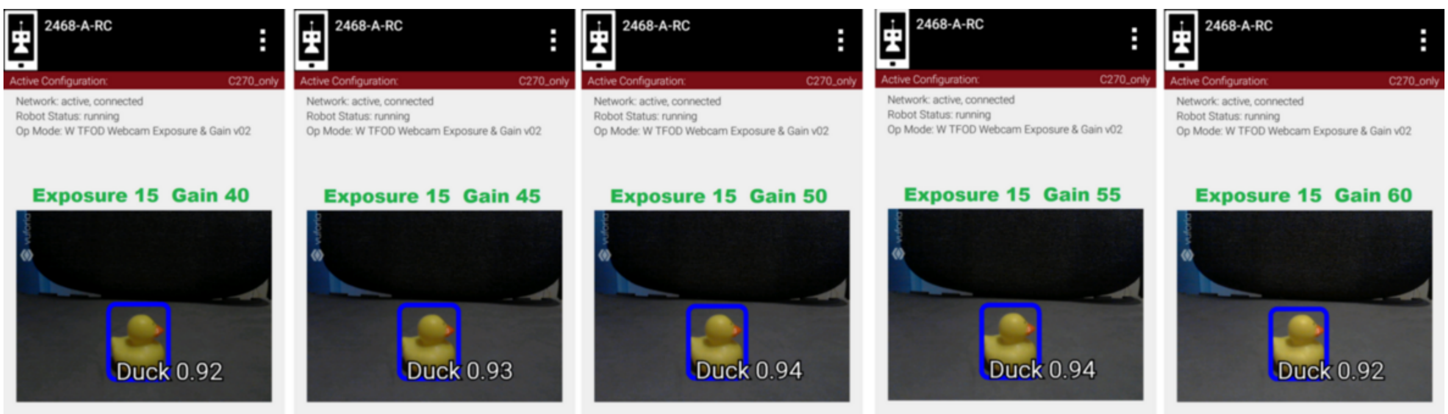


Fig. 178: Exp 15, Gain 040 -> 060

Five fresh readings were taken at each gain setting.

Higher gain does improve recognition, then performance declines. Then at higher levels, this TFOD model begins to "see" a Cube, not a Duck. The gain effect was similar to the exposure effect.

These two charts suggest that TFOD results are affected by, and can perhaps be optimized by, setting specific values for exposure and gain. A team should compare this with the default or automatic performance of their robot and webcam, in the full range of expected match conditions.

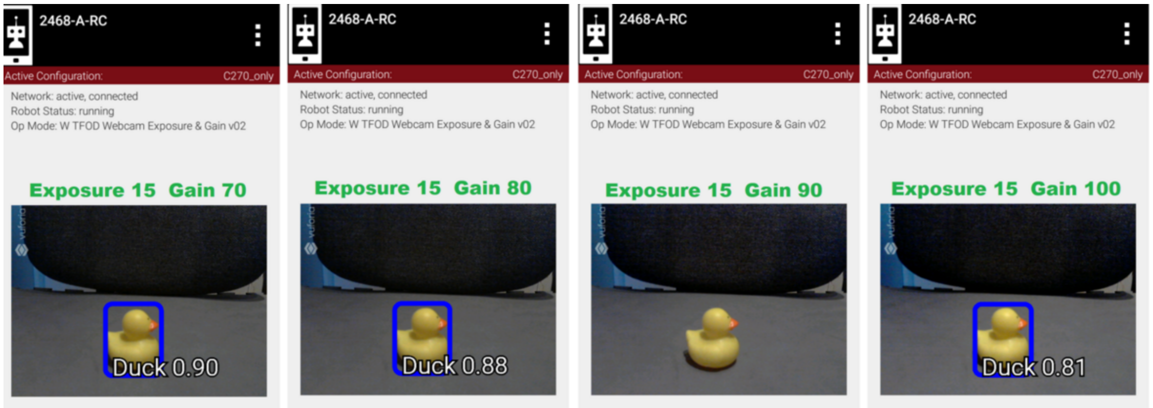


Fig. 179: Exp 15, Gain 070 -> 100

TFOD Duck Confidence, instant (< 1 sec.)					**Exposure = 15**
	Test Round				
Gain	**1**	**2**	**3**	**4**	**5**
0	none	none	none	none	none
20	none	none	none	none	none
30	none	none	none	none	none
33	0.84	0.83	0.84	0.82	0.82
35	0.88	0.87	0.86	0.88	0.87
40	0.91	0.91	0.91	0.91	0.91
45	0.93	0.92	0.92	0.92	0.92
50	0.93	0.93	0.93	0.93	0.92
55	0.94	0.94	0.93	0.94	0.94
60	0.94	0.93	0.94	0.94	0.94
70	0.90	0.90	0.91	0.90	0.91
80	0.88	0.85	none	0.88	0.83
90	none	none	none	none	none
100	0.81	Cube .85	Cube .83	Cube .83	Cube .88

Fig. 180: Five readings at each gain level

Example 3: An odd preview

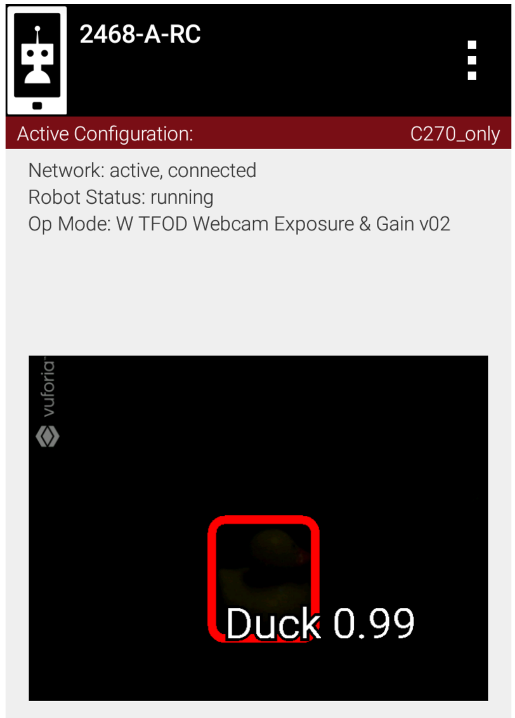


Fig. 181: Did TFOD make this recognition?

How can this be? Answer: this image was not an 'instant' result. Exposure was reduced very low, **after** TFOD had recognized the Duck.

The implementations of TensorFlow Lite (and Vuforia) are good at **tracking** a currently-identified object (or image) through translation, rotation, partial blockage, and even extreme changes in exposure.

White Balance - Webcam Controls

White Balance Control

org.firstinspires.ftc.robotcore.external.hardware.camera.controls.WhiteBalanceControl

Continuing with other interfaces, the SDK (new for version 7.1) provides methods for white balance control.

White balance is a digital camera setting that balances the **color temperature** in the image. Color temperature is measured in units of degrees Kelvin (K) and is a physical property of light.

For example, sunlight at noon measures between 5200-6000 K. An incandescent light bulb (warm/orange) has a color temperature of around 3000 K, while shade (cool/blue) measures around 8000 K.

When performed automatically, white balance adds the opposite color to the image in an attempt to bring the color temperature back to neutral. This interface WhiteBalanceControl allows the color temperature to be directly programmed by a user.

A single value is used here to control white balance temperature, in units of degrees Kelvin, of Java type integer. Here are the methods:

- setWhiteBalanceTemperature(int temperature)
- getWhiteBalanceTemperature()

As with exposure and gain, the webcam may support minimum and maximum allowed values of white balance temperature. These can be retrieved with:

- getMinWhiteBalanceTemperature()
- getMaxWhiteBalanceTemperature()

There are no `set()` methods for min and max temperature values; these are hard-coded in the webcam's firmware. Note that firmware settings may vary among different versions of the same webcam model.

The Logitech C920 webcam has a min value of 2000 and a max value of 6500.

White Balance Control Mode

org.firstinspires.ftc.robotcore.external.hardware.camera.controls.WhiteBalanceControl.Mode

This interface supports 3 values of WhiteBalanceControl.Mode:

- AUTO
- MANUAL
- UNKNOWN

To directly control the color balance temperature, set the webcam to Manual mode. Mode is managed with these WhiteBalanceControl methods:

- setMode(WhiteBalanceControl.Mode.MODE)
- getMode()

The Logitech C920 defaults to Auto mode for white balance control, and even reverts to Auto in a fresh session, after being set to Manual in a previous session. For other CameraControl settings, some webcams revert to a default value and some preserve their last commanded value.

Focus - Webcam Controls

Focus Control

org.firstinspires.ftc.robotcore.external.hardware.camera.controls.FocusControl

At a distance called "focus length", a subject's image (light rays) converge from the lens to form a clear image on the webcam sensor.

If supported by the webcam, focus can be managed with these FocusControl methods:

- setFocusLength(double focusLength)
- getFocusLength()

Distance units are not specified here; they may be undimensioned values within an allowed range. For example, the Logitech C920 allows values from 0 to 250, with **higher** values focusing on **closer** objects.

The webcam may support minimum and maximum allowed values of focus length. These can be retrieved with:

- getMinFocusLength()
- getMaxFocusLength()

There are no `set()` methods for min and max focus length; these are hard-coded in the webcam's firmware. Note that firmware settings may vary among different versions of the same webcam model.

These and other focus methods are called on a FocusControl object, as described above for exposure.

Focus Control Mode

org.firstinspires.ftc.robotcore.external.hardware.camera.controls.FocusControl.Mode

A webcam may operate in one of various focus modes. To directly control the focus length, set the webcam to Fixed mode.

The SDK supports these values of FocusControl.Mode:

- *Auto*
- *ContinuousAuto*
- *Fixed*
- *Infinity*
- *Macro*
- *Unknown*

Mode is managed with these FocusControl methods:

- setMode(ExposureControl.Mode._mode_)
- getMode()

The Logitech C920 webcam offers two modes: ContinuousAuto and Fixed, which does respond to FocusControl methods. The Logitech C270 (older model) offers only Fixed mode, but does not allow programmed control.

Full details are described in the FocusControl Javadoc.

Pan-Tilt-Zoom Control

org.firstinspires.ftc.robotcore.external.hardware.camera.controls.PtzControl

The SDK provides methods for virtual pan (horizontal motion), tilt (vertical motion), and zoom (enlargement and reduction of image size). This is **virtual** PTZ since the actions are digitally simulated, within the full original image captured by the webcam. Pan and tilt are possible only to the extent that zoom has provided extra image space to move in that direction.

Pan and Tilt

A webcam does not typically express pan and tilt values in *pixels*, the smallest unit of image capture by the webcam sensor. For example, the Logitech C920 and the Microsoft LifeCam VX-5000 have a range of +/-36,000 units, far greater than the pixel count in each axis.

The webcam accepts pan and tilt as a pair of (x, y) values. Thus the SDK pan and tilt methods handle these values **only as a pair**, in a special class named PanTiltHolder. This class has two fields, named pan and tilt, of type integer.

Here's an example to illustrate using the basic methods:

```
myHolder.pan = 5;                   // assign the pan field
myHolder.tilt = 10;                 // assign the tilt field
myPtzControl.setPanTilt(myHolder);          // command the webcam with (x, y) pair
```

To retrieve values from the webcam:

```
newHolder = myPtzControl.getPanTilt();      // retrieve (x, y) pair from webcam
int currentPanValue = newHolder.pan;        // access the pan value
int currentTiltValue = newHolder.tilt;      // access the tilt value
```

The above examples assume these objects already exist:

```
PtzControl myPtzControl = vuforia.getCamera().getControl(PtzControl.class); // create PTZ webcam␣
↪control object
PtzControl.PanTiltHolder myHolder = new PtzControl.PanTiltHolder();         // instantiate input␣
↪holder object
PtzControl.PanTiltHolder newHolder;                                      // declare output holder object
```

The webcam may support minimum and maximum allowed pan/tilt paired values. Subject to the control object guidelines shown above, these can be retrieved as follows:

- `minPanTiltHolder = getMinPanTilt();`
- `maxPanTiltHolder = getMaxPanTilt();`

There are no `set()` methods for min and max pan/tilt values; these are hard-coded in the webcam's firmware. Note that firmware settings may vary among different versions of the same webcam model.

These pan and tilt methods are called on a PtzControl object, as described above for exposure.

Zoom

Virtual zoom is described with a single dimensionless value of type integer. Similar to the interfaces described above, virtual zoom can be managed with these methods:

- setZoom(int zoom)
- getZoom()
- getMinZoom()
- getMaxZoom()

The Logitech C920 allows zoom values ranging from 100 to 500, although values higher than 250-280 have no further effect on the preview image (influenced by Vuforia).

These zoom methods are called on a PtzControl object, as described above for exposure.

Evaluating Your Webcam

The firmware of a specific webcam may or may not support certain features described here. The SDK provides some methods to query the webcam and/or return values that indicate whether a valid response was available.

Exposure Support

Here are two methods to query exposure and a specific exposure mode:

- isExposureSupported()
- isModeSupported(ExposureControl.Mode._mode_)
  - for *mode*, enter the specific mode name you are testing

For the following methods, a field called `unknownExposure` of type long is returned if exposure unavailable:

- getExposure(TimeUnit.MILLISECONDS)
- getMinExposure(TimeUnit.MILLISECONDS)
- getMaxExposure(TimeUnit.MILLISECONDS)

The methods that set the exposure and mode can also return a Boolean, presumably indicating whether the operation was successful or not. As optional examples:

- `wasExposureSet = setExposure(25);`
- `wasExposureModeSet = setMode(ExposureControl.Mode.Manual)`

Likewise the AE Priority feature can return a Boolean. For example:

- `wasAEPrioritySet = setAePriority(true);`

Gain Support

The method that sets the gain can also return a Boolean indicating whether the operation was successful or not. As an optional example:

- `wasGainSet = setGain(25);`

White Balance Support

The methods that set temperature and mode can also return a Boolean, indicating whether the operation was successful or not. As optional examples:

- `wasTemperatureSet = setWhiteBalanceTemperature(3000);`
- `wasWhiteBalanceModeSet = setMode(WhiteBalanceControl.Mode.MANUAL);`

Focus Support

Here are two methods to query focus and and a specific focus mode:

- isFocusLengthSupported()
- isModeSupported(FocusControl.Mode._mode_)

The following methods return a **negative value** if the requested focus value is unavailable. For example, -1 is returned by the Logitech C270 and the Microsoft LifeCam VX-5000. The Javadoc also mentions a field `unknownFocusLength` of type double.

- getFocusLength()
- getMinFocusLength()
- getMaxFocusLength()

The methods that set the focus length and mode can also return a Boolean, presumably indicating whether the operation was successful or not. As optional examples:

- `wasFocusSet = setFocusLength(25);`
- `wasFocusModeSet = setMode(FocusControl.Mode.Fixed)`

PTZ Support

The methods that set the pan/tilt pair and zoom value can also return a Boolean, presumably indicating whether the operation was successful or not. As optional examples:

- `wasPanTiltSet = setPanTilt(myHolder);`
- `wasZoomSet = setZoom(3)`

For PTZ get() methods, some webcams simply **return zero** for unsupported values.

Some Caveats

- the SDK supports webcams conforming to the UVC standard
  - many non-UVC webcams work well in competition, despite lacking UVC certification
  - some non-UVC webcams can be listed in Configure Robot, but crash the RC app at runtime
- webcams may retain an assigned Exposure Mode or Focus Mode, even if unplugged
  - always verify the current mode
- for a given exposure value, one mode's preview may look very different than another mode's preview
- some webcams **accept** / `set()` and **confirm** / `get()` a **non-supported mode**
- Logitech C270 preview becomes **lighter** up to exposure 655, then rolls over to **dark** at 656
  - this webcam's Min is 0, Max is 1000.
- Logitech V-UAX16 preview looks normal at exposure = 0, becomes **darker** up to 30-40
- Logitech C920 **gain** value (0-255) greatly influences preview quality, comparable to **exposure** (0-204)
- restarting the RC app is sometimes needed after a webcam OpMode crashes
- firmware versions may vary among webcams of the same model number

Lastly, some features here may be implemented or enhanced with the help of an external library such as OpenCV or Easy-OpenCV. That potential is not covered in this basic tutorial. A separate tutorial covers the general use of External Libraries in Blocks and OnBot Java.

Sample OpModes

The intent of this tutorial is to describe the available webcam controls, allowing programmers to **develop their own solutions** guided by the SDK API (Javadoc).

The following sample OpModes are linked here for reference only. These rudimentary OpModes may not apply to your webcam and may not meet your needs in general.

Adjust exposure, gain and AE Priority

`W_WebcamControls_Exp_Gain.java`

```
/*

This example OpMode allows direct gamepad control of webcam exposure and
gain.  It's a companion to the FTC wiki tutorial on Webcam Controls.

Add your own Vuforia key, where shown below.
```

(continues on next page)

(continued from previous page)

```
Questions, comments and corrections to westsiderobotics@verizon.net

from v06 11/10/21
*/

package org.firstinspires.ftc.teamcode;

import org.firstinspires.ftc.robotcore.external.hardware.camera.controls.ExposureControl;
import org.firstinspires.ftc.robotcore.external.hardware.camera.controls.GainControl;
import java.util.concurrent.TimeUnit;

import com.qualcomm.robotcore.eventloop.opmode.Disabled;
import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.eventloop.opmode.TeleOp;

import org.firstinspires.ftc.robotcore.external.ClassFactory;
import org.firstinspires.ftc.robotcore.external.hardware.camera.WebcamName;
import org.firstinspires.ftc.robotcore.external.navigation.VuforiaLocalizer;

@TeleOp(name="Webcam Controls - Exp & Gain v06", group ="Webcam Controls")

public class W_WebcamControls_Exp_Gain_v06 extends LinearOpMode {

    private static final String VUFORIA_KEY =
            "  INSERT YOUR VUFORIA KEY HERE  ";

    // Declare class members
    private VuforiaLocalizer vuforia    = null;
    private WebcamName webcamName       = null;

    ExposureControl myExposureControl;  // declare exposure control object
    long minExp;
    long maxExp;
    long curExp;            // exposure is duration, in time units specified

    GainControl myGainControl;      // declare gain control object
    int minGain;
    int maxGain;
    int curGain;
    boolean wasSetGainSuccessful;   // returned from setGain()

    @Override public void runOpMode() {

        telemetry.setMsTransmissionInterval(50);

        // Connect to the webcam, using exact name per robot Configuration.
        webcamName = hardwareMap.get(WebcamName.class, "Webcam 1");

        /*
         * Configure Vuforia by creating a Parameter object, and passing it to the Vuforia engine.
         * We pass Vuforia the handle to a camera preview resource (on the RC screen).
         */

        int cameraMonitorViewId = hardwareMap.appContext.getResources().getIdentifier(
↪"cameraMonitorViewId", "id", hardwareMap.appContext.getPackageName());
        VuforiaLocalizer.Parameters parameters = new VuforiaLocalizer.
↪Parameters(cameraMonitorViewId);
        // VuforiaLocalizer.Parameters parameters = new VuforiaLocalizer.Parameters();
```

(continues on next page)

(continued from previous page)

```
        parameters.vuforiaLicenseKey = VUFORIA_KEY;

        // We also indicate which camera we wish to use.
        parameters.cameraName = webcamName;

        // Assign the Vuforia engine object.
        vuforia = ClassFactory.getInstance().createVuforia(parameters);

        // Assign the exposure and gain control objects, to use their methods.
        myExposureControl = vuforia.getCamera().getControl(ExposureControl.class);
        myGainControl = vuforia.getCamera().getControl(GainControl.class);

        // Display exposure features and settings of this webcam.
        checkExposureFeatures();

        // Retrieve from webcam its current exposure and gain values.
        curExp = myExposureControl.getExposure(TimeUnit.MILLISECONDS);
        curGain = myGainControl.getGain();

        // Display mode and starting values to user.
        telemetry.addLine("\nTouch Start arrow to control webcam Exposure and Gain");
        telemetry.addData("\nCurrent exposure mode", myExposureControl.getMode());
        telemetry.addData("Current exposure value", curExp);
        telemetry.addData("Current gain value", curGain);
        telemetry.update();


        waitForStart();

        // Get webcam exposure limits.
            minExp = myExposureControl.getMinExposure(TimeUnit.MILLISECONDS);
            maxExp = myExposureControl.getMaxExposure(TimeUnit.MILLISECONDS);

        // Get webcam gain limits.
            minGain = myGainControl.getMinGain();
            maxGain = myGainControl.getMaxGain();

        // Change mode to Manual, in order to control directly.
        // A non-default setting may persist in the camera, until changed again.
        myExposureControl.setMode(ExposureControl.Mode.Manual);

        // Set initial exposure and gain, same as current.
        myExposureControl.setExposure(curExp, TimeUnit.MILLISECONDS);
        myGainControl.setGain(curGain);

        // This loop allows manual adjustment of exposure and gain,
        // while observing the effect on the preview image.
        while (opModeIsActive()) {

            // Manually adjust the webcam exposure and gain variables.
            float changeExp = -gamepad1.left_stick_y;
            float changeGain = -gamepad1.right_stick_y;

            int changeExpInt = (int) (changeExp*5);
            int changeGainInt = (int) (changeGain*5);

            curExp += changeExpInt;
            curGain += changeGainInt;
```

(continues on next page)

(continued from previous page)

```
            // Ensure inputs are within webcam limits, if provided.
            curExp = Math.max(curExp, minExp);
            curExp = Math.min(curExp, maxExp);
            curGain = Math.max(curGain, minGain);
            curGain = Math.min(curGain, maxGain);

            // Update the webcam's settings.
            myExposureControl.setExposure(curExp, TimeUnit.MILLISECONDS);
            wasSetGainSuccessful = myGainControl.setGain(curGain);

            // Manually set Auto-Exposure Priority.
            if (gamepad1.a) {                           // turn on with green A
                myExposureControl.setAePriority(true);
            } else if (gamepad1.b) {                    // turn off with red B
                myExposureControl.setAePriority(false);
            }

            telemetry.addLine("\nExposure: left stick Y; Gain: right stick Y");
            telemetry.addData("Exposure", "Min:%d, Max:%d, Current:%d", minExp, maxExp, curExp);
            telemetry.addData("Gain", "Min:%d, Max:%d, Current:%d", minGain, maxGain, curGain);
            telemetry.addData("Gain change successful?", wasSetGainSuccessful);
            telemetry.addData("Current exposure mode", myExposureControl.getMode());

            telemetry.addLine("\nAutoExposure Priority: green A ON; red B OFF");
            telemetry.addData("AutoExposure Priority?", myExposureControl.getAePriority());
            telemetry.update();

            sleep(100);

        }   // end main while() loop

    }    // end OpMode


    // Display the exposure features and modes supported by this webcam.
    private void checkExposureFeatures() {

        while (!gamepad1.y && !isStopRequested()) {
            telemetry.addLine("**Exposure settings of this webcam:");
            telemetry.addData("Exposure control supported?", myExposureControl.
↪isExposureSupported());
            telemetry.addData("Autoexposure priority?", myExposureControl.getAePriority());

            telemetry.addLine("\n**Exposure Modes supported by this webcam:");
            telemetry.addData("AperturePriority", myExposureControl.isModeSupported(ExposureControl.
↪Mode.AperturePriority));
            telemetry.addData("Auto", myExposureControl.isModeSupported(ExposureControl.Mode.Auto));
            telemetry.addData("ContinuousAuto", myExposureControl.isModeSupported(ExposureControl.
↪Mode.ContinuousAuto));
            telemetry.addData("Manual", myExposureControl.isModeSupported(ExposureControl.Mode.
↪Manual));
            telemetry.addData("ShutterPriority", myExposureControl.isModeSupported(ExposureControl.
↪Mode.ShutterPriority));
            telemetry.addData("Unknown", myExposureControl.isModeSupported(ExposureControl.Mode.
↪Unknown));
            telemetry.addLine("*** PRESS Y TO CONTINUE ***");
            telemetry.update();
        }
```

(continues on next page)

(continued from previous page)

```
    }   // end method checkExposureFeatures()


}   // end OpMode class
```

Adjust exposure and gain with TFOD (test OpMode for Examples 1, 2, 3)

W_TFOD_WebcamExpGain.java

```
/*
 * This example OpMode shows how existing webcam controls can affect
 * TensorFlow Object Detection (TFOD) of FTC Freight Frenzy game elements.
 * It's a companion to the FTC wiki tutorial on Webcam Controls.
 *
 * Put the Driver Station in Landscape Mode for this telemetry.
 *
 * The FTC SDK 7.0 includes up to 7 ways of controlling the preview image,
 * depending on webcam capability.  This OpMode uses 2 of those controls,
 * Exposure and Gain, available on most webcams and offering good
 * potential for affecting TFOD recognition.
 *
 * This OpMode simply adds ExposureControl and GainControl methods to the FTC
 * sample called "ConceptTensorFlowObjectDetectionWebcam.java".  Here, you can
 * use a gamepad to directly change the preview image and observe TFOD results.
 *
 * Teams can use this to seek better recognition results from their existing
 * TFOD model -- whether the basic version in the 7.0 release, or their own
 * custom model created with the FTC Machine Learning toolchain.
 *
 * Exposure, gain and other CameraControl values could be pre-programmed in
 * team autonomous OpModes.  It's also possible to manually enter such values
 * before a match begins, based on anticipated lighting, starting position and
 * other game-time factors.
 *
 * Add your own Vuforia key, where shown below.
 *
 * Questions, comments and corrections to westsiderobotics@verizon.net
 *
 * from v04 11/11/21
 */

package org.firstinspires.ftc.teamcode;

import org.firstinspires.ftc.robotcore.external.hardware.camera.controls.ExposureControl;
import org.firstinspires.ftc.robotcore.external.hardware.camera.controls.GainControl;
import java.util.concurrent.TimeUnit;

import com.qualcomm.robotcore.eventloop.opmode.TeleOp;
import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import org.firstinspires.ftc.robotcore.external.navigation.VuforiaLocalizer;
import org.firstinspires.ftc.robotcore.external.tfod.TFObjectDetector;
import org.firstinspires.ftc.robotcore.external.tfod.Recognition;
import org.firstinspires.ftc.robotcore.external.hardware.camera.WebcamName;
import org.firstinspires.ftc.robotcore.external.ClassFactory;
import java.util.List;
```

(continues on next page)

(continued from previous page)

```
@TeleOp(name = "W TFOD Webcam Exposure & Gain v04", group = "Webcam Controls")

public class W_TFOD_WebcamExpGain_v04 extends LinearOpMode {
  /* Note: This sample uses the all-objects Tensor Flow model (FreightFrenzy_BCDM.tflite), which␣
↪contains
   * the following 4 detectable objects
   *  0: Ball,
   *  1: Cube,
   *  2: Duck,
   *  3: Marker (duck location tape marker)
   *
   *  Two additional model assets are available which only contain a subset of the objects:
   *  FreightFrenzy_BC.tflite  0: Ball,  1: Cube
   *  FreightFrenzy_DM.tflite  0: Duck,  1: Marker
   */
    private static final String TFOD_MODEL_ASSET = "FreightFrenzy_BCDM.tflite";
    private static final String[] LABELS = {
      "Ball",
      "Cube",
      "Duck",
      "Marker"
    };

    /*
     * IMPORTANT: You need to obtain your own license key to use Vuforia. The string below with␣
↪which
     * 'parameters.vuforiaLicenseKey' is initialized is for illustration only, and will not␣
↪function.
     * A Vuforia 'Development' license key, can be obtained free of charge from the Vuforia␣
↪developer
     * web site at https://developer.vuforia.com/license-manager.
     *
     * Vuforia license keys are always 380 characters long, and look as if they contain mostly
     * random data. As an example, here is a example of a fragment of a valid key:
     *      ... yIgIzTqZ4mWjk9wd3cZO9T1axEqzuhxoGlfOOI2dRzKS4T0hQ8kT ...
     * Once you've obtained a license key, copy the string from the Vuforia web site
     * and paste it in to your code on the next line, between the double quotes.
     */
    private static final String VUFORIA_KEY =
            //"   INSERT YOUR VUFORIA KEY HERE  ";

    /**
     * {@link #vuforia} is the variable we will use to store our instance of the Vuforia
     * localization engine.
     */
    private VuforiaLocalizer vuforia;

    /**
     * {@link #tfod} is the variable we will use to store our instance of the TensorFlow Object
     * Detection engine.
     */
    private TFObjectDetector tfod;

    // *** ADD WEBCAM CONTROLS -- SECTION START ***
    ExposureControl myExposureControl;  // declare exposure control object
    long minExp;
    long maxExp;
    long curExp;            // exposure is duration, in time units specified
```

(continues on next page)

(continued from previous page)

```
    GainControl myGainControl;      // declare gain control object
    int minGain;
    int maxGain;
    int curGain;
    boolean wasSetGainSuccessful;   // returned from setGain()

    boolean isAEPriorityOn = false;
    // *** ADD WEBCAM CONTROLS -- SECTION END ***

    @Override
    public void runOpMode() {
        // The TFObjectDetector uses the camera frames from the VuforiaLocalizer, so we create that
        // first.
        initVuforia();
        initTfod();

        /**
         * Activate TensorFlow Object Detection before we wait for the start command.
         * Do it here so that the Camera Stream window will have the TensorFlow annotations visible.
         **/
        if (tfod != null) {
            tfod.activate();

            // The TensorFlow software will scale the input images from the camera to a lower␣
↪resolution.
            // This can result in lower detection accuracy at longer distances (> 55cm or 22").
            // If your target is at distance greater than 50 cm (20") you can adjust the␣
↪magnification value
            // to artificially zoom in to the center of image.  For best results, the "aspectRatio"␣
↪argument
            // should be set to the value of the images used to create the TensorFlow Object␣
↪Detection model
            // (typically 16/9).
            tfod.setZoom(1.0, 16.0/9.0);    // modified for testing Exposure & Gain
            // tfod.setZoom(2.5, 16.0/9.0); // original settings in Concept OpMode
        }

        // *** ADD WEBCAM CONTROLS -- SECTION START ***

        // Assign the exposure and gain control objects, to use their methods.
        myExposureControl = vuforia.getCamera().getControl(ExposureControl.class);
        myGainControl = vuforia.getCamera().getControl(GainControl.class);

        // get webcam exposure limits
        minExp = myExposureControl.getMinExposure(TimeUnit.MILLISECONDS);
        maxExp = myExposureControl.getMaxExposure(TimeUnit.MILLISECONDS);

        // get webcam gain limits
        minGain = myGainControl.getMinGain();
        maxGain = myGainControl.getMaxGain();

        // Change mode to Manual, in order to control directly.
        // A non-default setting may persist in the camera, until changed again.
        myExposureControl.setMode(ExposureControl.Mode.Manual);

        // Retrieve from webcam its current exposure and gain values
        curExp = myExposureControl.getExposure(TimeUnit.MILLISECONDS);
        curGain = myGainControl.getGain();
```

(continues on next page)

(continued from previous page)

```
        // display exposure mode and starting values to user
        telemetry.addLine("\nTouch Start arrow to control webcam Exposure and Gain");
        telemetry.addData("\nCurrent exposure mode", myExposureControl.getMode());
        telemetry.addData("Current exposure value", curExp);
        telemetry.addData("Current gain value", curGain);
        telemetry.update();
        // *** ADD WEBCAM CONTROLS -- SECTION END ***


        waitForStart();

        if (opModeIsActive()) {
            while (opModeIsActive()) {

                // *** ADD WEBCAM CONTROLS -- SECTION START ***
                // Driver Station in Landscape Mode for this telemetry.
                telemetry.addLine("Exposure: left stick Y; Gain: right stick Y");
                telemetry.addData("Exposure", "Min:%d, Max:%d, Current:%d", minExp, maxExp, curExp);
                telemetry.addData("Gain", "Min:%d, Max:%d, Current:%d", minGain, maxGain, curGain);
                telemetry.addLine("\nAutoExposure Priority: green A ON; red B OFF");
                telemetry.addData("AE Priority on?", isAEPriorityOn);
                // *** ADD WEBCAM CONTROLS -- SECTION END ***

                if (tfod != null) {
                    // getUpdatedRecognitions() will return null if no new information is available␣
↪since
                    // the last time that call was made.
                    List<Recognition> updatedRecognitions = tfod.getUpdatedRecognitions();
                    if (updatedRecognitions != null) {
                      telemetry.addData("\n# Object Detected", updatedRecognitions.size());
                      // step through the list of recognitions and display boundary info.
                      int i = 0;
                      for (Recognition recognition : updatedRecognitions) {
                        telemetry.addData(String.format("label (%d)", i), recognition.getLabel());
                        telemetry.addData(String.format("  left,top (%d)", i), "%.03f , %.03f",
                                recognition.getLeft(), recognition.getTop());
                        telemetry.addData(String.format("  right,bottom (%d)", i), "%.03f , %.03f",
                                recognition.getRight(), recognition.getBottom());
                        i++;
                      }  // end for() loop

                    }   // end if (updatedRecognitions)

                }   // end if (tfod)

                telemetry.update();

                // *** ADD WEBCAM CONTROLS -- SECTION START ***

                // manually adjust the webcam exposure & gain variables
                float changeExp = -gamepad1.left_stick_y;
                float changeGain = -gamepad1.right_stick_y;

                int changeExpInt = (int) (changeExp*2);     // was *5
                int changeGainInt = (int) (changeGain*2);   // was *5

                curExp += changeExpInt;
                curGain += changeGainInt;
```

(continues on next page)

(continued from previous page)

```
            if (gamepad1.a) {           // AE Priority ON with green A
                myExposureControl.setAePriority(true);
                isAEPriorityOn = true;
            } else if (gamepad1.b) {    // AE Priority OFF with red B
                myExposureControl.setAePriority(false);
                isAEPriorityOn = false;
            }

            // ensure inputs are within webcam limits, if provided
            curExp = Math.max(curExp, minExp);
            curExp = Math.min(curExp, maxExp);
            curGain = Math.max(curGain, minGain);
            curGain = Math.min(curGain, maxGain);

            // update the webcam's settings
            myExposureControl.setExposure(curExp, TimeUnit.MILLISECONDS);
            wasSetGainSuccessful = myGainControl.setGain(curGain);

            sleep(50);             // slow down the main while() loop

            // *** ADD WEBCAM CONTROLS -- SECTION END ***

        }   // end main while() loop

    } // end if opModeIsActive()

}  // end runOpMode()


/**
 * Initialize the Vuforia localization engine.
 */
private void initVuforia() {
    /*
     * Configure Vuforia by creating a Parameter object, and passing it to the Vuforia engine.
     */
    VuforiaLocalizer.Parameters parameters = new VuforiaLocalizer.Parameters();

    parameters.vuforiaLicenseKey = VUFORIA_KEY;
    parameters.cameraName = hardwareMap.get(WebcamName.class, "Webcam 1");

    //  Instantiate the Vuforia engine
    vuforia = ClassFactory.getInstance().createVuforia(parameters);

    // Loading trackables is not necessary for the TensorFlow Object Detection engine.

}   // end method initVuforia()

/**
 * Initialize the TensorFlow Object Detection engine.
 */
private void initTfod() {
    int tfodMonitorViewId = hardwareMap.appContext.getResources().getIdentifier(
        "tfodMonitorViewId", "id", hardwareMap.appContext.getPackageName());
    TFObjectDetector.Parameters tfodParameters = new TFObjectDetector.
→Parameters(tfodMonitorViewId);
   tfodParameters.minResultConfidence = 0.8f;
   tfodParameters.isModelTensorFlow2 = true;
   tfodParameters.inputSize = 320;
```

(continues on next page)

(continued from previous page)

```
        tfod = ClassFactory.getInstance().createTFObjectDetector(tfodParameters, vuforia);
        tfod.loadModelFromAsset(TFOD_MODEL_ASSET, LABELS);
    }  // end method initTfod()

}  //end class
```

Adjust white balance temperature, if supported

W_WebcamControls_WhiteBalance.java

```
/*

This example OpMode allows direct gamepad control of white balance temperature,
if supported.  It's a companion to the FTC wiki tutorial on Webcam Controls.

Put the Driver Station Layout in Landscape mode for this telemetry.

Add your own Vuforia key, where shown below.

Questions, comments and corrections to westsiderobotics@verizon.net

from v01 11/12/21
 */

package org.firstinspires.ftc.teamcode;

import org.firstinspires.ftc.robotcore.external.hardware.camera.controls.WhiteBalanceControl;

import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.eventloop.opmode.TeleOp;

import org.firstinspires.ftc.robotcore.external.ClassFactory;
import org.firstinspires.ftc.robotcore.external.hardware.camera.WebcamName;
import org.firstinspires.ftc.robotcore.external.navigation.VuforiaLocalizer;

@TeleOp(name="Webcam Controls - White Balance v01", group ="Webcam Controls")

public class W_WebcamControls_WhiteBalance_v01 extends LinearOpMode {

    private static final String VUFORIA_KEY =
            //"   INSERT YOUR VUFORIA KEY HERE  ";

    // Class Members
    private VuforiaLocalizer vuforia    = null;
    private WebcamName webcamName       = null;

    WhiteBalanceControl myWBControl;  // declare White Balance Control object
    int minWhiteBalanceTemp;          // temperature in degrees Kelvin (K)
    int maxWhiteBalanceTemp;
    int curWhiteBalanceTemp;

    int tempIncrement = 100;          // for manual gamepad adjustment
    boolean wasTemperatureSet;        // did the set() operation succeed?
    boolean wasWhiteBalanceModeSet;   // did the setMode() operation succeed?
    boolean useTempLimits = true;

    @Override public void runOpMode() {
```

(continues on next page)

(continued from previous page)

```
        telemetry.setMsTransmissionInterval(50);

        // Connect to the webcam, using exact name per robot Configuration.
        webcamName = hardwareMap.get(WebcamName.class, "Webcam 1");

        /*
         * Configure Vuforia by creating a Parameter object, and passing it to the Vuforia engine.
         * We pass Vuforia the handle to a camera preview resource (on the RC screen).
         */

        int cameraMonitorViewId = hardwareMap.appContext.getResources().getIdentifier(
↪"cameraMonitorViewId", "id", hardwareMap.appContext.getPackageName());
        VuforiaLocalizer.Parameters parameters = new VuforiaLocalizer.
↪Parameters(cameraMonitorViewId);

        parameters.vuforiaLicenseKey = VUFORIA_KEY;

        // We also indicate which camera we wish to use.
        parameters.cameraName = webcamName;

        //  Set up the Vuforia engine
        vuforia = ClassFactory.getInstance().createVuforia(parameters);

        // Assign the white balance control object, to use its methods.
        myWBControl = vuforia.getCamera().getControl(WhiteBalanceControl.class);

        // display current white balance mode
        telemetry.addLine("\nTouch Start arrow to control white balance temperature.");
        telemetry.addLine("\nRecommended: put Driver Station Layout in Landscape.");
        telemetry.addData("\nCurrent white balance mode", myWBControl.getMode());
        telemetry.update();


        waitForStart();

        // set variable to current actual temperature, if supported
        curWhiteBalanceTemp = myWBControl.getWhiteBalanceTemperature();

        // get webcam temperature limits, if provided
        minWhiteBalanceTemp = myWBControl.getMinWhiteBalanceTemperature();
        maxWhiteBalanceTemp = myWBControl.getMaxWhiteBalanceTemperature();

        // Set white balance mode to Manual, for direct control.
        // A non-default setting may persist in the camera, until changed again.
        wasWhiteBalanceModeSet = myWBControl.setMode(WhiteBalanceControl.Mode.MANUAL);

        while (opModeIsActive()) {

            // manually adjust the color temperature variable
            if (gamepad1.x) {                   // increase with blue X (cooler)
                curWhiteBalanceTemp += tempIncrement;
            }  else if (gamepad1.b) {           // decrease with red B (warmer)
                curWhiteBalanceTemp -= tempIncrement;
            }

            // ensure inputs are within webcam limits, if provided
            if (useTempLimits) {
                curWhiteBalanceTemp = Math.max(curWhiteBalanceTemp, minWhiteBalanceTemp);
```

(continues on next page)

(continued from previous page)

```
                curWhiteBalanceTemp = Math.min(curWhiteBalanceTemp, maxWhiteBalanceTemp);
            }

            // update the color temperature setting
            wasTemperatureSet = myWBControl.setWhiteBalanceTemperature(curWhiteBalanceTemp);

            // display live feedback while user observes preview image
            telemetry.addLine("Adjust temperature with blue X (cooler) & red B (warmer)");

            telemetry.addData("\nWhite Balance Temperature",
                "Min: %d, Max: %d, Actual: %d",
                minWhiteBalanceTemp, maxWhiteBalanceTemp,
                myWBControl.getWhiteBalanceTemperature());

            telemetry.addData("\nProgrammed temperature", "%d", curWhiteBalanceTemp);
            telemetry.addData("Temperature set OK?", wasTemperatureSet);

            telemetry.addData("\nCurrent white balance mode", myWBControl.getMode());
            telemetry.addData("White balance mode set OK?", wasWhiteBalanceModeSet);
            telemetry.update();

            sleep(100);

        }   // end main while() loop

    }    // end OpMode

}   // end class
```

Adjust focus, if supported

W_WebcamControls_Focus.java

```
/*

This example OpMode allows direct gamepad control of webcam focus,
if supported.  It's a companion to the FTC wiki tutorial on Webcam Controls.

Add your own Vuforia key, where shown below.

Questions, comments and corrections to westsiderobotics@verizon.net

from v03 11/10/21
 */

package org.firstinspires.ftc.teamcode;

import org.firstinspires.ftc.robotcore.external.hardware.camera.controls.FocusControl;

import com.qualcomm.robotcore.eventloop.opmode.OpMode;
import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.eventloop.opmode.TeleOp;
import org.firstinspires.ftc.robotcore.external.Telemetry;

import org.firstinspires.ftc.robotcore.external.ClassFactory;
import org.firstinspires.ftc.robotcore.external.hardware.camera.WebcamName;
import org.firstinspires.ftc.robotcore.external.navigation.VuforiaLocalizer;
```

(continues on next page)

(continued from previous page)

```
@TeleOp(name="Webcam Controls - Focus v03", group ="Webcam Controls")

public class W_WebcamControls_Focus_v03 extends LinearOpMode {

    private static final String VUFORIA_KEY =
            "   INSERT YOUR VUFORIA KEY HERE  ";

    // Class Members
    private VuforiaLocalizer vuforia    = null;
    private WebcamName webcamName       = null;

    FocusControl myFocusControl;  // declare Focus Control object
    double minFocus;              // focus length
    double maxFocus;
    double curFocus;
    double focusIncrement = 10;   // for manual gamepad adjustment
    boolean isFocusSupported;     // does this webcam support getFocusLength()?
    boolean isMinFocusSupported;  // does this webcam support getMinFocusLength()?
    boolean isMaxFocusSupported;  // does this webcam support getMaxFocusLength()?

    @Override public void runOpMode() {

        telemetry.setMsTransmissionInterval(50);

        // Connect to the webcam, using exact name per robot Configuration.
        webcamName = hardwareMap.get(WebcamName.class, "Webcam 1");

        /*
         * Configure Vuforia by creating a Parameter object, and passing it to the Vuforia engine.
         * We pass Vuforia the handle to a camera preview resource (on the RC screen).
         */

        int cameraMonitorViewId = hardwareMap.appContext.getResources().getIdentifier(
↪"cameraMonitorViewId", "id", hardwareMap.appContext.getPackageName());
        VuforiaLocalizer.Parameters parameters = new VuforiaLocalizer.
↪Parameters(cameraMonitorViewId);

        parameters.vuforiaLicenseKey = VUFORIA_KEY;

        // We also indicate which camera we wish to use.
        parameters.cameraName = webcamName;

        //  Set up the Vuforia engine
        vuforia = ClassFactory.getInstance().createVuforia(parameters);

        // Assign the focus control object, to use its methods.
        myFocusControl = vuforia.getCamera().getControl(FocusControl.class);

        // display current Focus Control Mode
        telemetry.addLine("\nTouch Start arrow to control webcam Focus");
        telemetry.addData("\nDefault focus mode", myFocusControl.getMode());
        telemetry.update();


        waitForStart();

        // set variable to current actual focal length of webcam, if supported
        curFocus = myFocusControl.getFocusLength();
```

(continues on next page)

(continued from previous page)

```
        isFocusSupported = (curFocus >= 0.0);           // false if negative

        //isFocusSupported = true;  // can activate this line for testing

        // get webcam focal length limits, if provided
        minFocus = myFocusControl.getMinFocusLength();
        isMinFocusSupported = (minFocus >= 0.0);        // false if negative

        maxFocus = myFocusControl.getMaxFocusLength();
        isMaxFocusSupported = (maxFocus >= 0.0);        // false if negative

        // A non-default setting may persist in the camera, until changed again.
        myFocusControl.setMode(FocusControl.Mode.Fixed);

        // set initial focus length, if supported
        myFocusControl.setFocusLength(curFocus);

        checkFocusModes();     // display Focus Modes supported by this webcam

        while (opModeIsActive()) {

            // manually adjust the webcam focus variable
            if (gamepad1.right_bumper) {
                curFocus += focusIncrement;
            }  else if (gamepad1.left_bumper) {
                curFocus -= focusIncrement;
            }

            // ensure inputs are within webcam limits, if provided
            if (isMinFocusSupported) {
                curFocus = Math.max(curFocus, minFocus);
            } else {
                telemetry.addLine("minFocus not available on this webcam");
            }

            if (isMaxFocusSupported) {
                curFocus = Math.min(curFocus, maxFocus);
            } else {
                telemetry.addLine("maxFocus not available on this webcam");
            }

            // update the webcam's focus length setting
            myFocusControl.setFocusLength(curFocus);

            // display live feedback while user observes preview image
            if (isFocusSupported) {
                telemetry.addLine("Adjust focus length with Left & Right Bumpers");

                telemetry.addLine("\nWebcam properties (negative means not supported)");
                telemetry.addData("Focus Length", "Min: %.1f, Max: %.1f, Actual: %.1f",
                    minFocus, maxFocus, myFocusControl.getFocusLength());

                telemetry.addData("\nProgrammed Focus Length", "%.1f", curFocus);
            } else {
                telemetry.addLine("\nThis webcam does not support adustable focus length.");
            }

            telemetry.update();
```

(continues on next page)

(continued from previous page)

```
            sleep(100);

        }   // end main while() loop

    }    // end OpMode


    // display Focus Modes supported by this webcam
    private void checkFocusModes() {

        while (!gamepad1.y && opModeIsActive()) {
            telemetry.addLine("Focus Modes supported by this webcam:");
            telemetry.addData("Auto", myFocusControl.isModeSupported(FocusControl.Mode.Auto));
            telemetry.addData("ContinuousAuto", myFocusControl.isModeSupported(FocusControl.Mode.
↪ContinuousAuto));
            telemetry.addData("Fixed", myFocusControl.isModeSupported(FocusControl.Mode.Fixed));
            telemetry.addData("Infinity", myFocusControl.isModeSupported(FocusControl.Mode.
↪Infinity));
            telemetry.addData("Macro", myFocusControl.isModeSupported(FocusControl.Mode.Macro));
            telemetry.addData("Unknown", myFocusControl.isModeSupported(FocusControl.Mode.Unknown));
            telemetry.addLine("*** PRESS Y TO CONTINUE ***");
            telemetry.update();
        }

    }   // end method checkFocusModes()

}   // end class
```

Adjust virtual pan, tilt and zoom, if supported

W_WebcamControls_PTZ.java

```
/*

This example OpMode allows direct gamepad control of webcam virtual pan/tilt/zoom,
if supported.  It's a companion to the FTC wiki tutorial on Webcam Controls.

Add your own Vuforia key, where shown below.

Some tested webcams:
Logitech C920 responds to all pan/tilt/zoom (PTZ) methods
Microsoft LifeCam VX-5000 does support PTZ, with 10 positions each.
Logitech C270 (old firmware) does not support PTZ.

Questions, comments and corrections to westsiderobotics@verizon.net

from v03 11/11/21
 */

package org.firstinspires.ftc.teamcode;

import org.firstinspires.ftc.robotcore.external.hardware.camera.controls.PtzControl;

import com.qualcomm.robotcore.eventloop.opmode.Disabled;
import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.eventloop.opmode.TeleOp;
```

(continues on next page)

(continued from previous page)

```
import org.firstinspires.ftc.robotcore.external.ClassFactory;
import org.firstinspires.ftc.robotcore.external.hardware.camera.WebcamName;
import org.firstinspires.ftc.robotcore.external.navigation.VuforiaLocalizer;

import org.firstinspires.ftc.robotcore.external.Telemetry;
import org.firstinspires.ftc.robotcore.external.Telemetry.DisplayFormat;


@TeleOp(name="Webcam Controls - PTZ v03", group ="Webcam Controls")

public class W_WebcamControls_PTZ_v03 extends LinearOpMode {

    private static final String VUFORIA_KEY =
            // "  INSERT YOUR VUFORIA KEY HERE   ";

    // Class Members
    private VuforiaLocalizer vuforia    = null;
    private WebcamName webcamName       = null;

    PtzControl myPtzControl;               // declare PTZ Control object

    PtzControl.PanTiltHolder minPanTilt;    // declare Holder for min
    int minPan;
    int minTilt;

    PtzControl.PanTiltHolder maxPanTilt;    // declare Holder for max
    int maxPan;
    int maxTilt;

    // declare Holder for current; must instantiate to set values
    PtzControl.PanTiltHolder curPanTilt = new PtzControl.PanTiltHolder();
    int curPan;
    int curTilt;

    int minZoom;
    int maxZoom;
    int curZoom;

    int panIncrement = 7200;        // for manual gamepad control
    int tiltIncrement = 7200;
    int zoomIncrement = 1;
    // pan/tilt increment 7200 is for Microsoft LifeCam VX-5000
    // can use smaller increment for Logitech C920

    boolean useLimits = true;       // use webcam-provided limits


    @Override public void runOpMode() {

        telemetry.setMsTransmissionInterval(50);

        // Connect to the webcam, using exact name per robot Configuration.
        webcamName = hardwareMap.get(WebcamName.class, "Webcam 1");

        /*
         * Configure Vuforia by creating a Parameter object, and passing it to the Vuforia engine.
         * We pass Vuforia the handle to a camera preview resource (on the RC screen).
         */
```

(continues on next page)

(continued from previous page)

```
        int cameraMonitorViewId = hardwareMap.appContext.getResources().getIdentifier(
↪"cameraMonitorViewId", "id", hardwareMap.appContext.getPackageName());
        VuforiaLocalizer.Parameters parameters = new VuforiaLocalizer.
↪Parameters(cameraMonitorViewId);

        parameters.vuforiaLicenseKey = VUFORIA_KEY;

        // We also indicate which camera we wish to use.
        parameters.cameraName = webcamName;

        // Assign the Vuforia engine object
        vuforia = ClassFactory.getInstance().createVuforia(parameters);

        // Assign the PTZ control object, to use its methods.
        myPtzControl = vuforia.getCamera().getControl(PtzControl.class);

        // display current PTZ values to user
        telemetry.addLine("\nTouch Start arrow to control webcam Pan, Tilt & Zoom (PTZ)");

        // Get the current properties from the webcam.  May be dummy zeroes.
        curPanTilt = myPtzControl.getPanTilt();
        curPan = curPanTilt.pan;
        curTilt = curPanTilt.tilt;
        curZoom = myPtzControl.getZoom();

        telemetry.addData("\nInitial pan value", curPan);
        telemetry.addData("Initial tilt value", curTilt);
        telemetry.addData("Initial zoom value", curZoom);
        telemetry.update();


        waitForStart();

        // Get webcam PTZ limits; may be dummy zeroes.
        minPanTilt = myPtzControl.getMinPanTilt();
        minPan = minPanTilt.pan;
        minTilt = minPanTilt.tilt;

        maxPanTilt = myPtzControl.getMaxPanTilt();
        maxPan = maxPanTilt.pan;
        maxTilt = maxPanTilt.tilt;

        minZoom = myPtzControl.getMinZoom();
        maxZoom = myPtzControl.getMaxZoom();

        while (opModeIsActive()) {

            // manually adjust the webcam PTZ variables
            if (gamepad1.dpad_right) {
                curPan += panIncrement;
            }  else if (gamepad1.dpad_left) {
                curPan -= panIncrement;
            }

            if (gamepad1.dpad_up) {
                curTilt += tiltIncrement;
            }  else if (gamepad1.dpad_down) {
                curTilt -= tiltIncrement;
            }  //reverse tilt direction for Microsoft LifeCam VX-5000
```

(continues on next page)

(continued from previous page)

```
        if (gamepad1.y) {
            curZoom += zoomIncrement;
        }  else if (gamepad1.a) {
            curZoom -= zoomIncrement;
        }

        // ensure inputs are within webcam limits, if provided
        if (useLimits) {
            curPan = Math.max(curPan, minPan);
            curPan = Math.min(curPan, maxPan);

            curTilt = Math.max(curTilt, minTilt);
            curTilt = Math.min(curTilt, maxTilt);

            curZoom = Math.max(curZoom, minZoom);
            curZoom = Math.min(curZoom, maxZoom);
        }


        // update the webcam's settings
        curPanTilt.pan = curPan;
        curPanTilt.tilt = curTilt;
        myPtzControl.setPanTilt(curPanTilt);
        myPtzControl.setZoom(curZoom);

        // display live feedback while user observes preview image
        telemetry.addLine("\nPAN: Dpad up/dn; TILT: Dpad L/R; ZOOM: Y/A");

        telemetry.addLine("\nWebcam properties (zero may mean not supported)");

        telemetry.addData("Pan", "Min: %d, Max: %d, Actual: %d",
            minPan, maxPan, myPtzControl.getPanTilt().pan);
        telemetry.addData("Programmed Pan", curPan);

        telemetry.addData("\nTilt", "Min: %d, Max: %d, Actual: %d",
            minTilt, maxTilt, myPtzControl.getPanTilt().tilt);

        telemetry.addData("Programmed Tilt", curTilt);

        telemetry.addData("\nZoom", "Min: %d, Max: %d, Actual: %d",
            minZoom, maxZoom, myPtzControl.getZoom());
        telemetry.addData("Programmed Zoom", curZoom);

        telemetry.update();

        sleep(100);

    }   // end main while() loop

}     // end OpMode

}   // end class
```

Summary

Some webcam controls in the SDK could potentially improve TFOD recognitions. Exposure, gain and other values could be pre-programmed in team autonomous OpModes. It's also possible to manually enter such values before a match begins, based on anticipated lighting, starting position and other game-time factors.

You are encouraged to submit other webcam reports and examples that worked for you.

Questions, comments and corrections to westsiderobotics@verizon.net

21.5.3 Camera Calibration for *FIRST* Tech Challenge

What is a camera calibration and why is it needed?

Cameras are composed of many different components that can introduce variability in the actual image that a camera ultimately "sees". Camera calibration is a process that mathematically models how a camera & lens combination ultimately sees the world, for example how wide the field of view is. Calibrating your camera is a must if you desire to use it for high-precision tasks, such as performing precision measurements using the camera or obtaining accurate 6DOF pose data from fiducial marker systems like AprilTags. It's important to note that calibrations are not only specific to the camera and lens, but also specific to the resolution used on a particular camera as well!

Warning: Due to the differences in refractive index, calibrations performed in air and in liquids (for example, in water) are not transferrable. Calibrations must be performed within the medium that the camera will be operating in.

Camera Calibration Methods

There are many methods to calibrate cameras, including OpenCV, MATLAB, MRCAL etc.

- For advanced teams, using MRCAL is likely the best option - it is a tool developed by NASA JPL that provides extensive data on how good your calibration is and what goes into the numerical optimization to arrive at the optimal parameters.
- For the rest of us, here we explain how to calibrate your camera using 3DF Zephyr, which is extremely easy to use and can provide reasonable results.

Warning: 3DF Zephyr is a Microsoft Windows 64-bit application. It is not supported on 32-bit versions of Windows, nor is it supported on Mac or on Linux platforms.

Calibrating with 3DF Zephyr

1. Download and install 3DF Zephyr Free Edition.
2. Copy the sample `UtilityCameraFrameCapture` OpMode to your teamcode folder, and modify the parameters at the top according to your needs. It's important to note that this Sample is only written in Java.
3. In 3DF Zephyr, go to:
   - Utilities –> Images –> Camera Calibration

   and follow the instructions. Use the frame capture OpMode to take the pictures.
4. Connect your Robot Controller device to your computer with a USB cable and copy the captured frames to your computer. They will be located in the root of the USB storage, with names prefixed by `VisionPortal-`.

5. Press the *Add Images* button in 3DF Zephyr and point it to the images you just copied to your computer.
6. Run the calibration target analysis in 3DF Zephyr; when it is complete, it will provide you with `fx, fy, cx, cy` which are the needed calibration parameters to be applied to your `AprilTagProcessor`.

21.6 Camera Color Processing

Learn more about using a simple webcam or smartphone camera to perform Color Processing

21.6.1 Color Processing Introduction

Overview

The *FIRST* Tech Challenge SDK v10.1 software now includes some **Color Processing** features from OpenCV, a popular and powerful open-source library for vision processing.

Introduced with INTO THE DEEP, these new features will help *FIRST* Tech Challenge teams **identify colors** and **process color blobs**, useful in any game requiring color and shape recognition.

Here's the outline of this tutorial's main pages:

Color Processing Color Sensor

Overview

A simple way to use FTC's new OpenCV vision tools is to operate a "Color Sensor". Namely, it can determine **the color seen by the robot's camera**, in a specified zone.

Below, the small central rectangle is the region being evaluated:

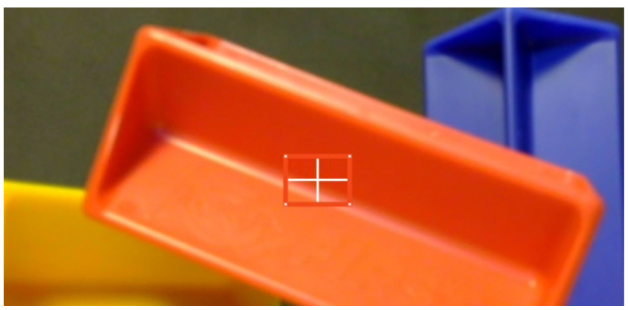


Fig. 182: Color sensor detection zone

A key benefit is that the camera can be much further away from the object than, for example, a REV Color Sensor or others like it.

It's still important to accurately point the camera and carefully select the image zone to inspect.

For the above example, OpenCV can provide results like this:

The following sections describe how to do this, with a Sample OpMode.

Fig. 183: RED Detection using Color Sensor

Configuration

Skip this section if …

- *the active robot configuration already contains "Webcam 1"*, or
- *using the built-in camera of an Android phone as Robot Controller.*

Before starting the programming, REV Control Hub users should make a robot configuration that includes the USB webcam to be used as a color sensor.

For now, use the default webcam name, "Webcam 1". If a different name is preferred, edit the Sample OpMode to agree with the exact webcam name in the robot configuration.

Save and activate that configuration; its name should appear on the paired Driver Station screen.

Sample OpMode

Opening the Sample OpMode

To learn about opening the Sample OpMode, click the tab for Blocks or Java:

Blocks

1. On a laptop or desktop computer connected via Wi-Fi to the Robot Controller, open the Chrome browser. Go to the REV Control Hub's address `http://192.168.43.1:8080` (or `http://192.168.49.1:8080` for Android RC phone) and click the *Blocks* tab.
2. Click `Create New OpMode`, enter a new name such as "ColorSensor_Maria_v01", and select the Sample OpMode `ConceptVisionColorSensor`.
3. At the top of the Blocks screen, you can change the type from "TeleOp" to "Autonomous", since this Sample OpMode does not use gamepads.
4. If using the built-in camera of an RC phone, drag out the relevant Block from the left-side `VisionPortal.Builder` toolbox.
5. Save the OpMode, time to try it!

Java

1. Open your choice of OnBot Java or Android Studio.
2. In the `teamcode` folder, add/create a new OpMode with a name such as "ColorSensor_Bobby_v01.java", and select the Sample OpMode `ConceptVisionColorSensor.java`.
3. At about Line 58, you can change `@TeleOp` to `@Autonomous`, since this Sample OpMode does not use gamepads.
4. If using the built-in camera of an RC phone, follow the OpMode comments to specify that camera.
5. Click "Build", time to try it!

Running the Sample OpMode

On the Driver Station:

1. Select the Autonomous OpMode that you just saved or built.
2. Turn off the automatic 30-second match timer (green slider).
3. Touch INIT only.

The OpMode should give Telemetry, stating the main "matched" color inside the Region of Interest.

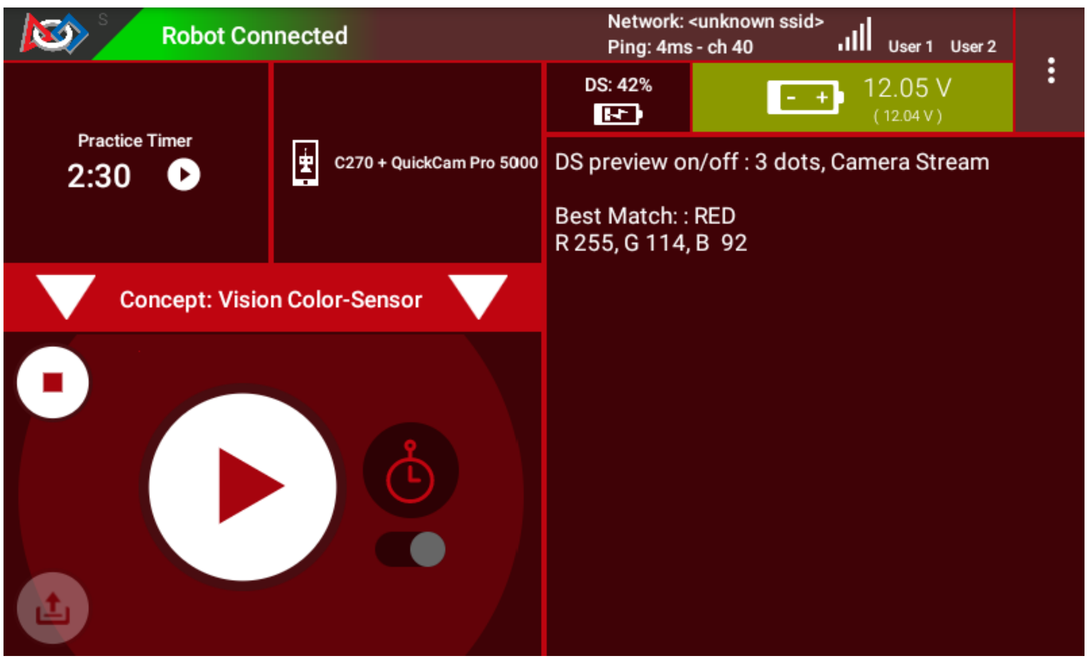


Fig. 184: Driver Station Telemetry

Move the camera around, and watch the Telemetry area on the Driver Station screen. It should state "BLUE" when pointing at a blue object, and likewise should identify other common colors.

It's working! You have a color sensor in your robot camera. Think about how to use this in the FTC Robot Game.

Skip the next two sections if you already know how to use FTC previews.

DS Preview

Before describing how to modify the OpMode, this page offers two sections showing how to view the OpenCV results with **previews**. Previewing is essential for working with vision code.

Opening the DS Preview

1. On the Driver Station (DS), remain in INIT – don't touch the Start button.
2. At the top right corner, touch the 3-dots menu, then `Camera Stream`. This shows the camera's view; tap the image to refresh it.

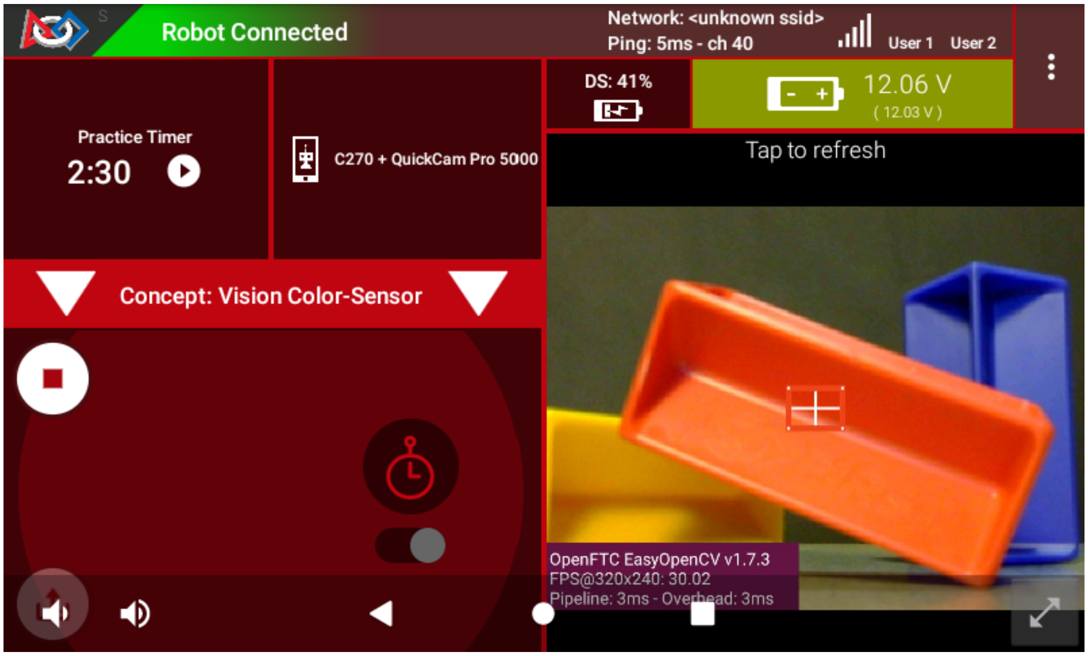


Fig. 185: Camera Stream Preview

Drawn on the image is the rectangle being evaluated, called the **Region of Interest** (ROI). The ROI border color is the rectangle's predominant color, reported to DS Telemetry.

If that border "disappears" against a solid-color background, the thin white cross-hairs and 4 small white dots can still identify the ROI.

For a BIG preview, touch the arrows at the bottom right corner.

Or, select Camera Stream again, to return to the previous screen and its Telemetry.

RC Preview

The Robot Controller (RC) device also makes a preview, called `LiveView`. This is full video, and is shown automatically on the screen of an RC phone.

The above preview is from a REV Control Hub.

It has no physical screen, so you must plug in an HDMI monitor **or** use open-source scrcpy (called "screen copy") to see the preview on a laptop or computer that's connected via Wi-Fi to the Control Hub.

Fig. 186: Control Hub LiveView

Modify the Sample

This Sample OpMode is designed for the user to select/edit **two inputs**:

- define the Region of Interest (ROI)
- list the colors that might be found

For the **first input**, there are 3 ways to define the ROI:

- entire frame
- sub-region, defined with standard image coordinates
- sub-region, defined with a normalized +/- 1.0 coordinate system

For the **second input**, you must list the candidate colors from which a result will be selected as a "Match".

Simply choose from the 10 "Swatches": RED, ORANGE, YELLOW, GREEN, CYAN, BLUE, PURPLE, MAGENTA, BLACK, WHITE. For efficiency, add only those Swatches for which you reasonably expect to get a match.

The Blocks and Java OpModes contain detailed comments to guide you through these edits. They are not repeated in this tutorial.

Building the VisionPortal

The Sample OpMode first creates a "Predominant Color" **Processor** using the **Builder** pattern. This is the same Builder pattern used to create an AprilTag Processor, and previously a TensorFlow Processor.

The Sample OpMode then creates a **VisionPortal**, again using a Builder pattern. This includes adding the "Predominant Color" Processor to the VisionPortal.

How does OpenCV determine the "predominant color" of the ROI? An algorithm called "k-means" determines clusters of similar colors. The color of the cluster with the most pixels is called "predominant" here. *(This will NOT be on the final.)*

Testing the Result

After trying and learning how the commands work, you can incorporate this Color Sensor into your Autonomous and/or TeleOp OpModes.

As seen in the OpMode's Telemetry section, the result is called `closestSwatch` and appears as a word (RED, BLUE, etc.). But this is not plain text!

Testing, or comparing, for a particular color-match must be done as follows. Select and read the Blocks **or** Java section below:

Blocks

At the left side, pull out the following multi-Block from `Vision/PredominantColor/Processor`:

Fig. 187: Closest Swatch Comparison

You must use this special Block to determine if the result is (for example) RED.

Why? The result, called `closestSwatch` is not **text** (yes it seems like text!). It's a type called `Swatch` and can be compared only to another `Swatch`.

Java

In the sample OpMode, here's the Telemetry that gives the result:

```
telemetry.addData("Best Match:", result.closestSwatch);
```

This displays as text, but this is **not** Java type `String`!

Here's how to determine if the result is (for example) RED:

```
if (result.closestSwatch == Swatch.RED)   {  }
```

Why? The result, called `closestSwatch` is of type `Swatch` and can be compared only to another `Swatch`.

OpMode Programming

The Color Sensor part of your team's Autonomous OpMode might include these goals:

1. Seek a color, using the code from this Sample OpMode
2. Take a robot action, based on finding that color

If so, select and read the Blocks **or** Java section below:

Blocks

Beginners often try this first:

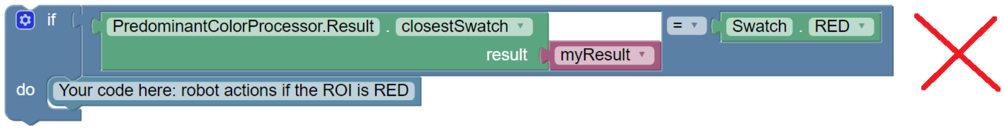


Fig. 188: Wrong way to act upon match result

The problem is, after the robot does the action for RED, the OpMode is still inside the vision loop. Very messy and unpredictable.

A better approach is to save the result (as text!), exit the loop, then retrieve the stored result to take the desired RED action.

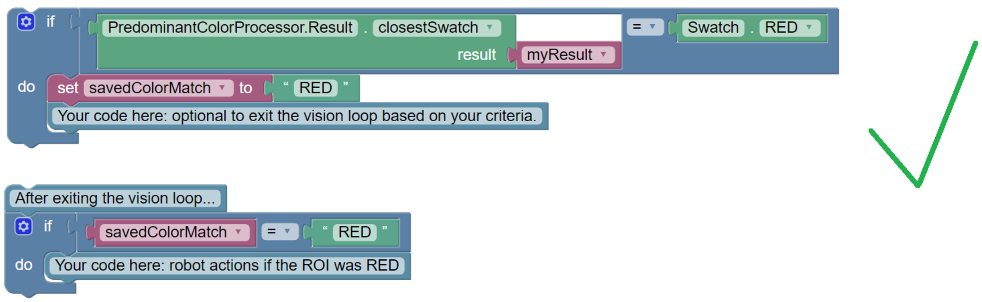


Fig. 189: Right way to act upon match result

How to exit the vision loop? It could be based on time , or finding a particular color, or finding a particular color 10 times in a row, or finding only a particular color for 1 full second, or any other desired criteria.

Java

The color result is generated inside a vision loop. Save the result (as text!), exit the loop, then retrieve the stored result to take the desired RED action.

```
String savedColorMatch = "NULL";
.
.
if (result.closestSwatch == Swatch.RED)     {
     savedColorMatch = "RED";
     // your code here: optional to exit the vision loop based on your criteria
     }
.
.
// After exiting the vision loop...
if (savedColorMatch == "RED")     {
     // your code here: robot actions if the ROI was RED
     }
```

How to exit the vision loop? It could be based on time, or finding a particular color, or finding a particular color 10 times in a row, or finding only a particular color for 1 full second, or any other desired criteria.

Advanced Use

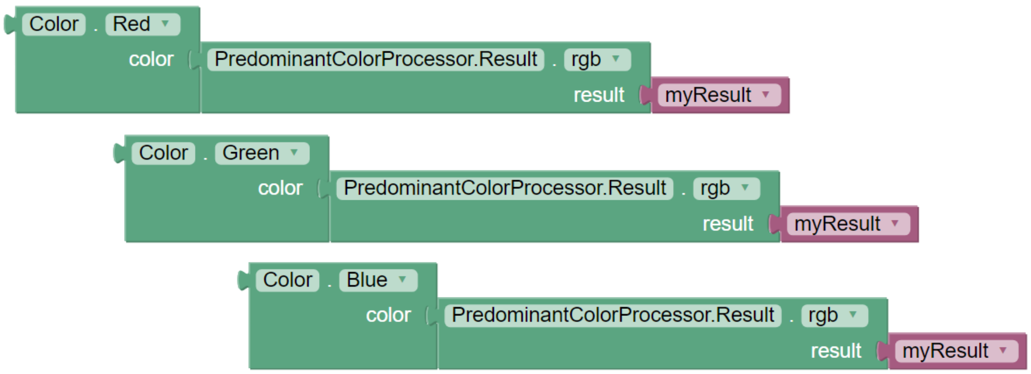
Some teams may prefer to read and evaluate the **actual RGB color values**, rather than rely on a generic Swatch result.

RGB is a **Color Space** that uses three numerical components of Red, Green and Blue. Values range from 0 to 255. For more info, see this tutorial's *Color Spaces* page.

Extracting the RGB components can be seen in the Telemetry portion of the Sample OpMode. Click the Blocks or Java tab:

Blocks

Here are the RGB components of the ROI's predominant color:



Note: the `Color` Block has a drop-down list that includes Hue, Saturation and Value. Those settings will **not work** here, to produce components in the HSV Color Space, because the source Block provides only RGB color (its method name is `.rgb`).

Java

Here are the RGB components of the ROI's predominant color:

- `Color.red(result.rgb)`
- `Color.green(result.rgb)`
- `Color.blue(result.rgb)`

For Blocks or Java, those component values can be assigned to numeric variables, with names like `ROIRedValue`, `ROIGreenValue`, and `ROIBlueValue`.

Now your code can process those RGB variables as desired.

Next Sections

Soon you can move ahead to try the **Color Locator** processor.

But first, learn a few basic concepts at this tutorial's *Color Blob Concepts* page.

Questions, comments and corrections to westsiderobotics@verizon.net

Color Blob Concepts

Color Blobs

An image can be evaluated by its **groupings of similar colors**.

The smallest unit of any digital image is a **pixel**: a tiny square of one particular color.

Each grouping or cluster of similar-colored pixels is called a **Blob**, which can be irregular in size and shape.

Forming a Blob is done automatically by the software. It seeks pixels of similar color that are **contiguous** – touching each other along an edge, not just at a corner.

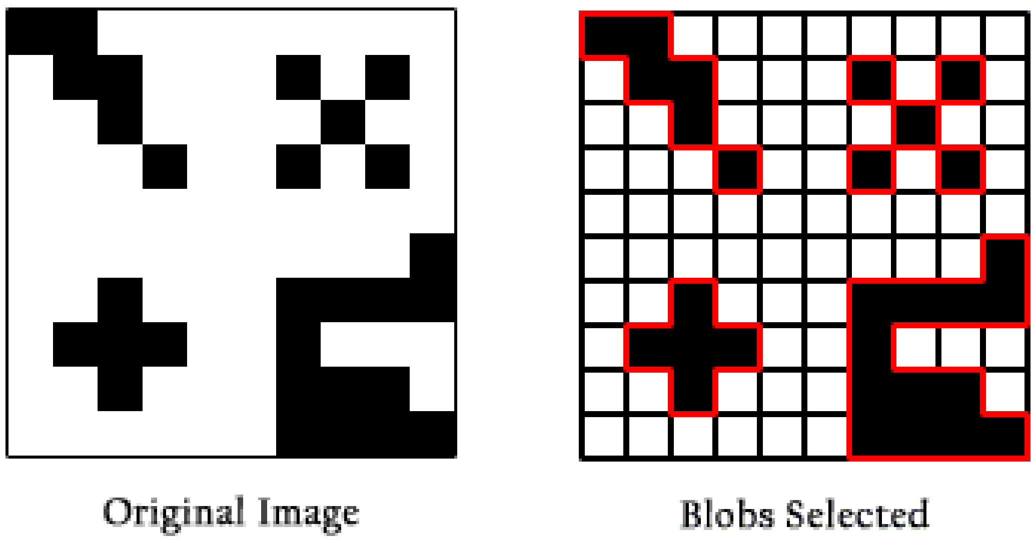


Fig. 190: Blob Formation Visualization

There are 9 Blobs here, not 4. Some are very small, just one pixel each.

The 5 pixels at top right, for example, are not contiguous (edges joined), so they are not joined to form a larger Blob.

The above simple example has only 2 colors: black and white. For FTC, the definition of "similar" colors is a range specified by you.

In the above example, the chair surfaces are not **exactly** the same shade of red. But with a **target** definition "close to red" or "mostly red", the software can form reasonable Blobs for further processing.

Color Processing

Now let's point the camera at an INTO THE DEEP game element called a **Sample**.

Here the software was told to seek shades of blue. The orange rectangle encloses a Blob of blue color.

But why doesn't the rectangle enclose the entire game piece? The software is processing only a certain **Region of Interest** or ROI. That's the white rectangle; its size and location are specified by you.

Anything outside the ROI will not be considered part of any Blob that is detected. This can help you avoid detecting (unwanted) background objects.

In the example above, the Blob was actually outlined in teal (blue-green color), very hard to see. Let's try another image:

Now the teal outline of the blue Blob can be seen. Its shape is irregular, which can be difficult for your OpMode to evaluate.

Fig. 191: Blobs from a Red Chair image

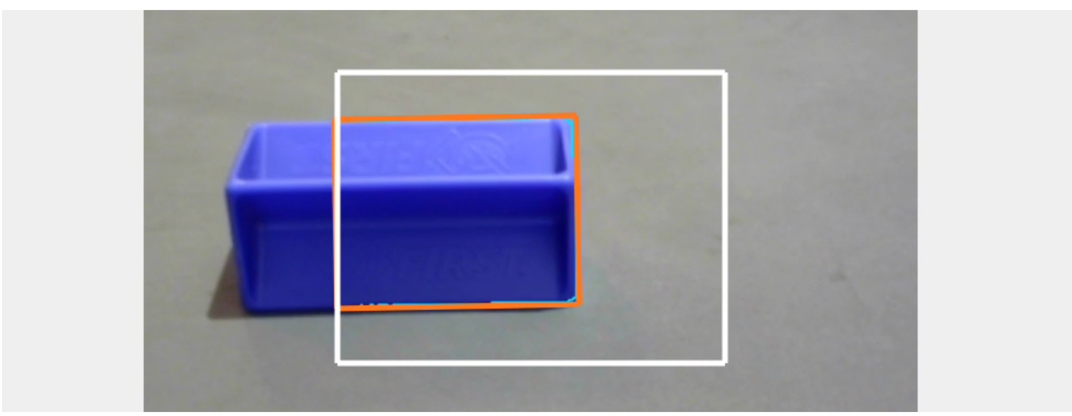


Fig. 192: Blob from a Blue SAMPLE

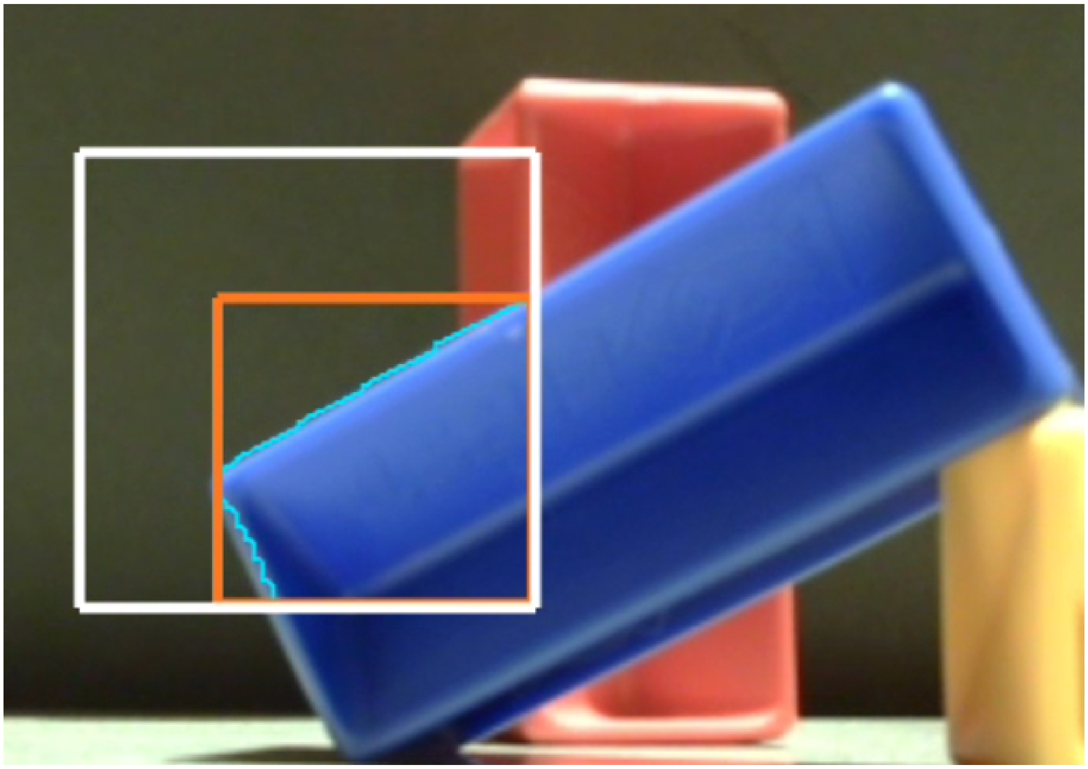


Fig. 193: Teal Outline of Blue Blob

boxFit Rectangles

The orange rectangle is drawn automatically by OpenCV, to give your OpMode a simpler geometric shape that represents the Blob. It's not **exactly** like the actual Blob, but hopefully still useful.

The orange rectangle, called the **boxFit**, fits tightly around the extreme edges of the Blob. The boxFit is **not** required to stay inside the Region of Interest. In the above case, the best-fitting rectangle happens to stay inside the ROI.

But here's another case:

Look very closely for the teal outline of the Blob, with its very rough lower edge.

Here, the best-fitting rectangle (boxFit) is **tilted**, and is not contained inside the ROI.

OpenCV provides all data for the boxFit, including its corner points, size, and tilt angle. It can even provide a fitted horizontal version of the boxFit rectangle, if you prefer not to handle a tilted boxFit.

Now things get a bit more complicated:

OpenCV detected **two Blobs**, each with a teal outline and each with a boxFit.

Your OpMode will need to "decide" which boxFit is important and which to ignore. Fortunately, OpenCV provides tools to **filter out** certain unwanted results. After filtering, your OpMode can **sort** the remaining results, to focus on the highest priority.

With these tools, your OpMode could handle even a "busy" result like this one:

Your programming tasks will include:

- determine which boxFit is most relevant,
- evaluate its data, and
- take robot action accordingly.

Now try the Sample OpMode for the *Color Locator* processor.

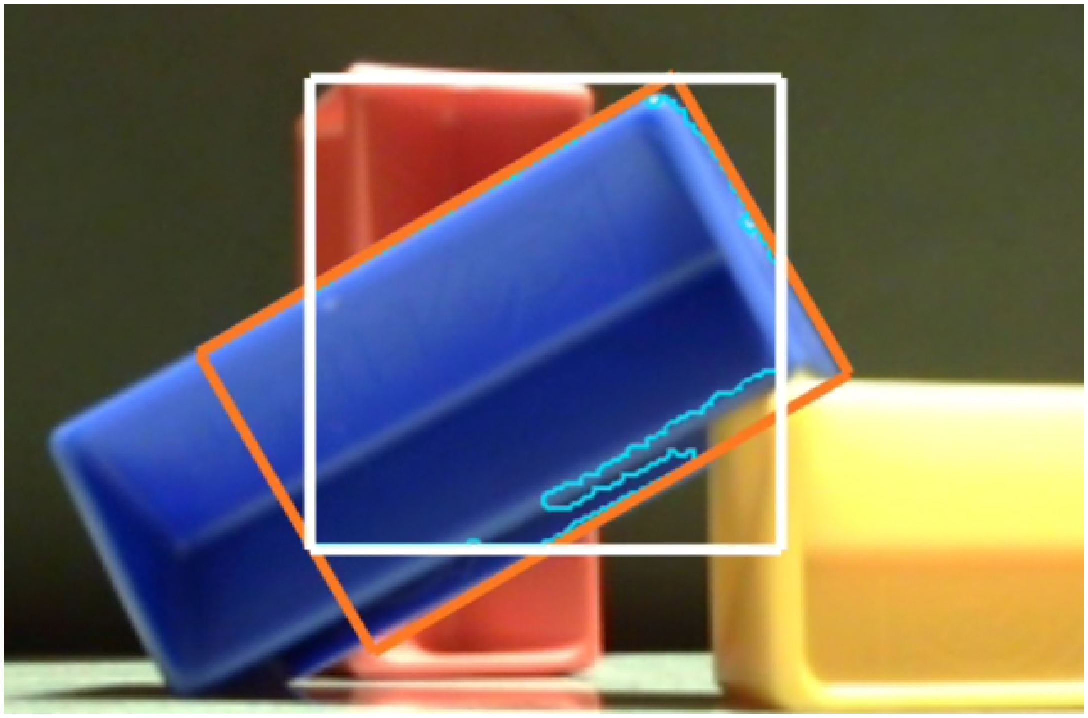


Fig. 194: New boxFit position

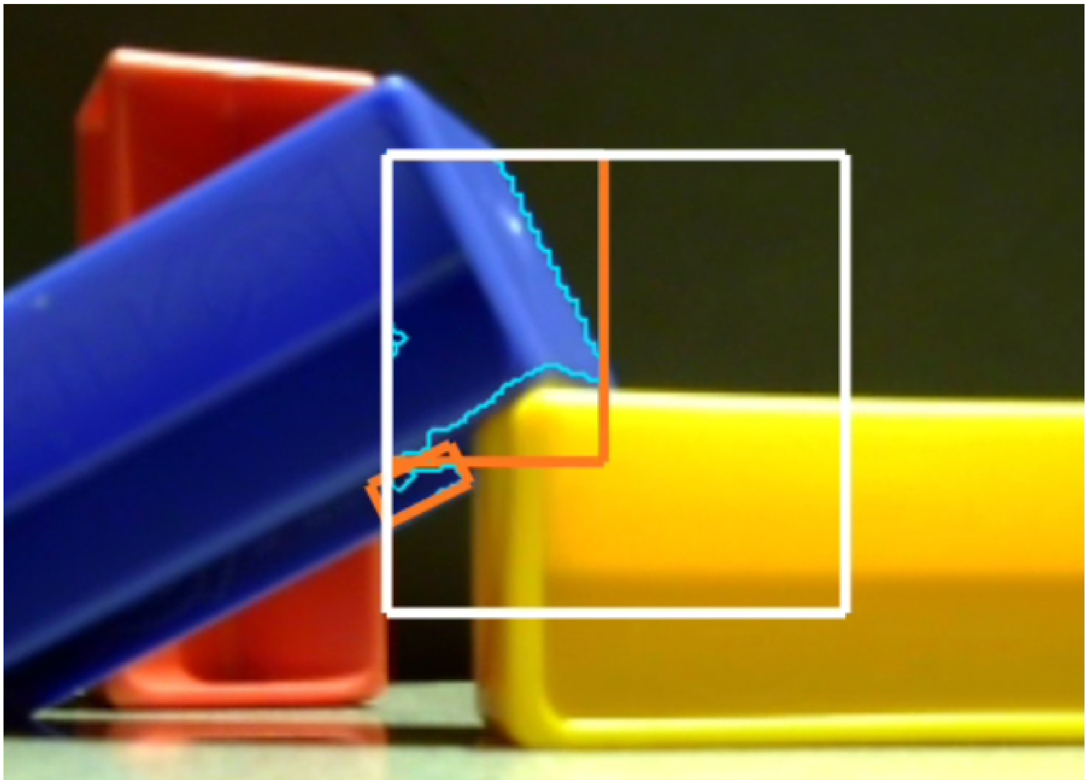


Fig. 195: Detecting two blobs

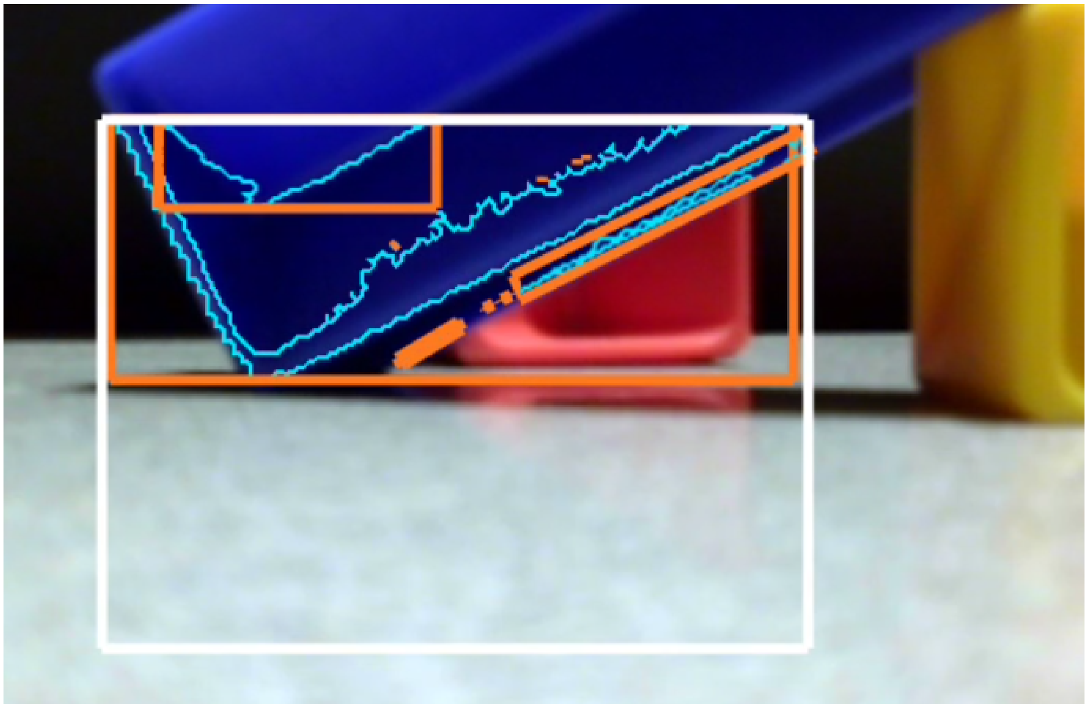


Fig. 196: Many Blob Detections

Questions, comments and corrections to westsiderobotics@verizon.net

Color Locator (Discover)

Overview

Another way to use FTC's new OpenCV vision tools is to operate a "Color Locator". Namely, it can **find a target color that you specify**.

As with the Color Sensor tool, you can specify a **Region of Interest** (ROI). Only that zone of the camera's view will be searched for the target color.

The "target color" is actually a **range** of numerical color values, for a better chance of finding the desired color.

OpenCV will form **"Blobs"** of that color. As described in the Concepts page here, a Blob is a contiguous cluster of similar-colored pixels.

Blobs often have a complex, irregular shape or **contour**, so they are represented here by a best-fit rectangle called **"boxFit"**.

The target color here is BLUE. The white rectangle is the Region of Interest (ROI), the teal jagged line is the Blob's contour (fully inside the ROI), and the purple rectangle is the boxFit.

The software reports the size, position and orientation of each "boxFit". That data can be evaluated by your OpMode for **robot navigation** and other actions.

The following sections describe how to do this, with a Sample OpMode.

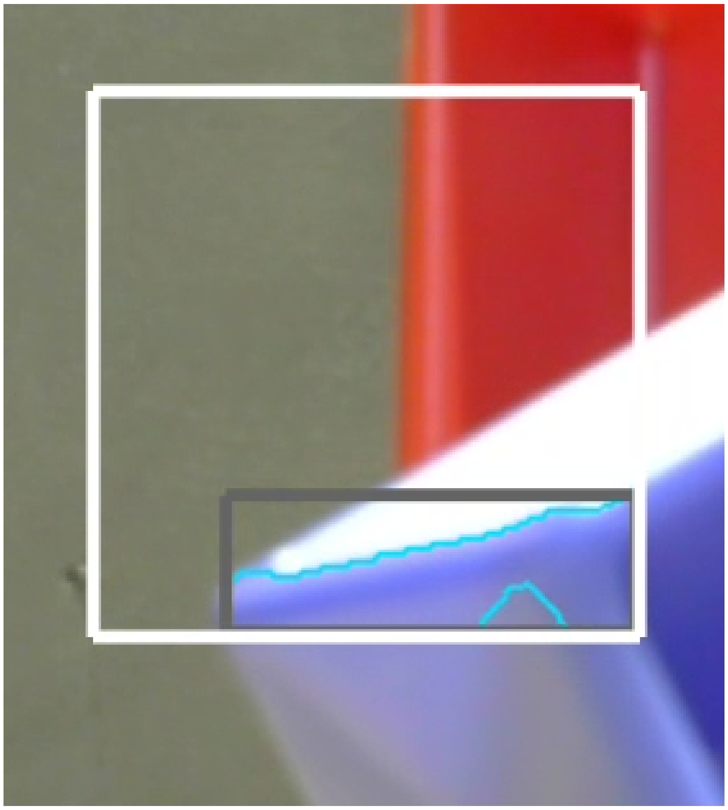


Fig. 197: Zoomed Color Blob detection

Configuration

Skip this section if …

- *the active robot configuration already contains "Webcam 1"*, or
- *using the built-in camera of an Android phone as Robot Controller.*

Before starting the programming, REV Control Hub users should make a robot configuration that includes the USB webcam to be used as a color locator.

For now, use the default webcam name, "Webcam 1". If a different name is preferred, edit the Sample OpMode to agree with the exact webcam name in the robot configuration.

Save and activate that configuration; its name should appear on the paired Driver Station screen.

Sample OpMode

Opening the Sample OpMode

To learn about opening the Sample OpMode, select and read the Blocks **or** Java section below:

Blocks

1. On a laptop or desktop computer connected via Wi-Fi to the Robot Controller, open the Chrome browser. Go to the REV Control Hub's address http://192.168.43.1:8080 (or http://192.168.49.1:8080 for Android RC phone) and click the Blocks tab.
2. Click `Create New OpMode`, enter a new name such as "ColorLocator_Monica_v01", and select the Sample OpMode `ConceptVisionColorLocator`.
3. At the top of the Blocks screen, you can change the type from "TeleOp" to "Autonomous", since this Sample OpMode does not use gamepads.
4. If using the built-in camera of an RC phone, drag out the relevant Block from the left-side `VisionPortal.Builder` toolbox.
5. Save the OpMode, time to try it!

Java

1. Open your choice of OnBot Java or Android Studio.
2. In the `teamcode` folder, add/create a new OpMode with a name such as "ColorLocator_Javier_v01.java", and select the Sample OpMode `ConceptVisionColorLocator.java`.
3. At about Line 63, you can change `@TeleOp` to `@Autonomous`, since this Sample OpMode does not use gamepads.
4. If using the built-in camera of an RC phone, follow the OpMode comments to specify that camera.
5. Click "Build", time to try it!

Running the Sample OpMode

1. On the Driver Station, select the Autonomous OpMode that you just saved or built.
2. Turn off the automatic 30-second match timer (green slider). Aim the camera at a **blue object**.
3. Touch INIT only. The OpMode should give Telemetry showing the results of one or more Blobs:

In this example, the Region of Interest (ROI) contains only one Blob of the default target color BLUE.

Move the camera around, especially at BLUE objects, and watch the Telemetry area on the Driver Station screen. It may sometimes show more lines of Blob data, and sometimes show no Blob data at all.

It's working! Your camera is working as a **color locator**. Think about how to use this in the FTC Robot Game.

Skip the next two sections, if you already know how to use FTC previews.

DS Preview

Before describing the telemetry data, this page offers two sections showing how to view the OpenCV results with **previews**. Previewing is essential for working with vision code.

On the Driver Station (DS), remain in INIT – don't touch the Start button.

At the top right corner, touch the 3-dots menu, then `Camera Stream`. This shows the camera's view; tap the image to refresh it.

The default target color here is BLUE. The white rectangle is the Region of Interest (ROI), the teal (light blue) jagged line is the Blob's contour (fully inside the ROI), and the orange rectangle is the boxFit.

For a BIG preview, touch the arrows at the bottom right corner.

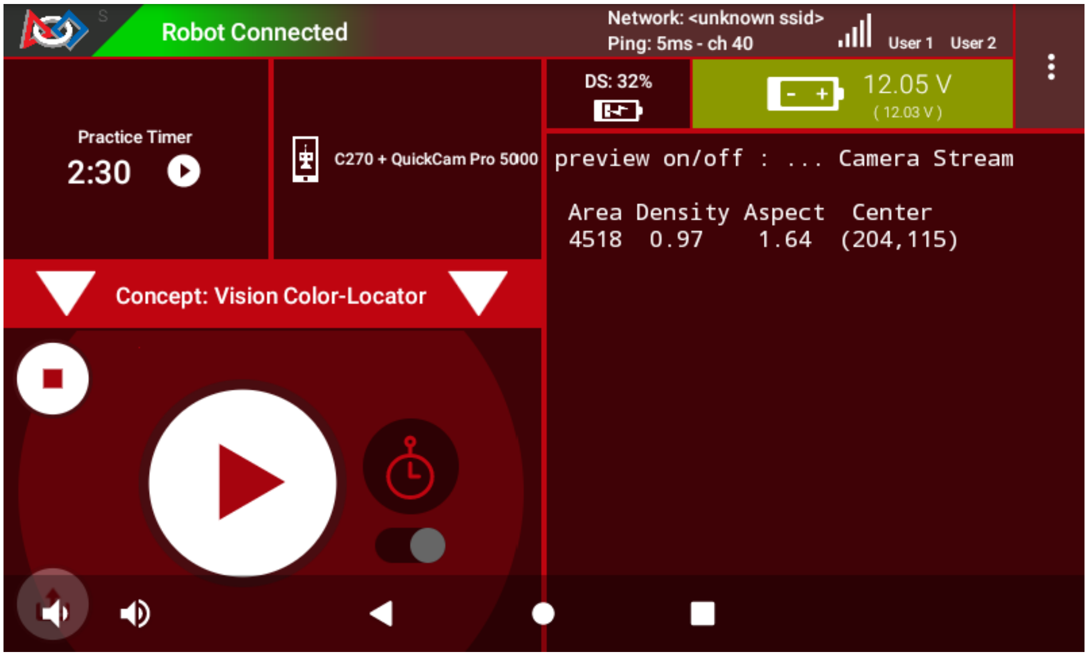


Fig. 198: Basic Telemetry

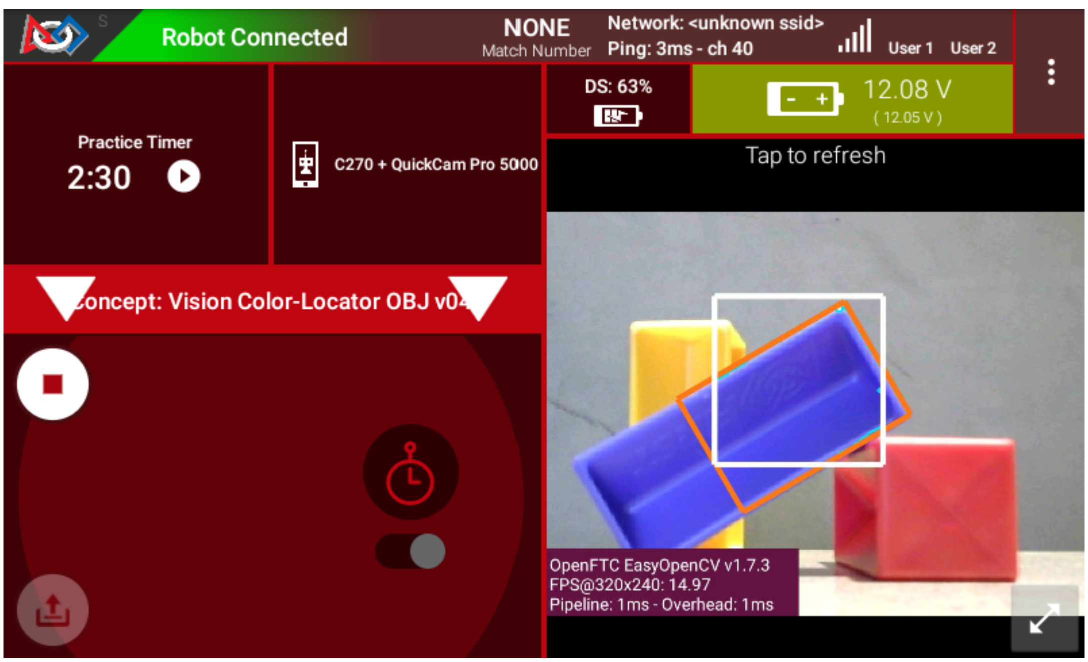


Fig. 199: DS Camera Stream Preview

Or, select Camera Stream again, to return to the previous screen and its Telemetry.

RC Preview

The Robot Controller (RC) device also makes a preview, called `LiveView`. This is full video, and is shown automatically on the screen of an RC phone.

Fig. 200: LiveView stream

The above preview is from a REV Control Hub.

It has no physical screen, so you must plug in an HDMI monitor **or** use open-source scrcpy (called "screen copy") to see the preview on a laptop or computer that's connected via Wi-Fi to the Control Hub.

Basic Telemetry Data

Let's look closer at the DS telemetry:

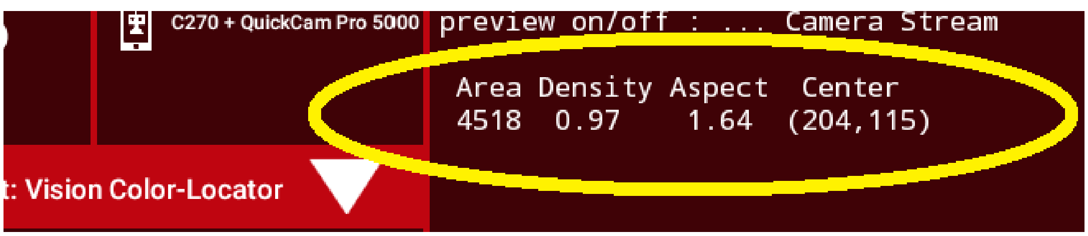


Fig. 201: Locator Telemetry

In this example, the Region of Interest (ROI) contains only one Blob of the default target color BLUE. You could probably move your camera to achieve the same result - with the help of previews.

The **first column** shows the **Area**, in pixels, of the Blob (contour, not boxFit). By default, the Sample OpMode uses a **filter** to show Blobs between 50 and 20,000 pixels. Also by default, the Sample uses a **sort** tool to display multiple Blobs in descending order of Area (largest is first).

The **second column** shows the **Density** of the Blob contour. From the Sample comments:

> *A blob's density is an indication of how "full" the contour is. If you put a rubber band around the contour you would get the "Convex Hull" of the contour. The density is the ratio of Contour-area to Convex Hull-area.*

The **third column** shows the **Aspect Ratio of the boxFit**, the best-fit rectangle around the contour:

> *A blob's Aspect Ratio is the ratio of boxFit long side to short side. A perfect Square has an Aspect Ratio of 1. All others are > 1.*

Tip: The boxFit is not required to stay inside the ROI. Also the boxFit may be **tilted** at some angle, namely not horizontal. This will be discussed more in a later page.

The **fourth column** shows the (X, Y) position of the **Center** of the boxFit rectangle. With the origin at the full image's top left corner, X increases to the right and Y increases downward.

Blob Formation

So far these examples have shown a **single Blob** formed by OpenCV:

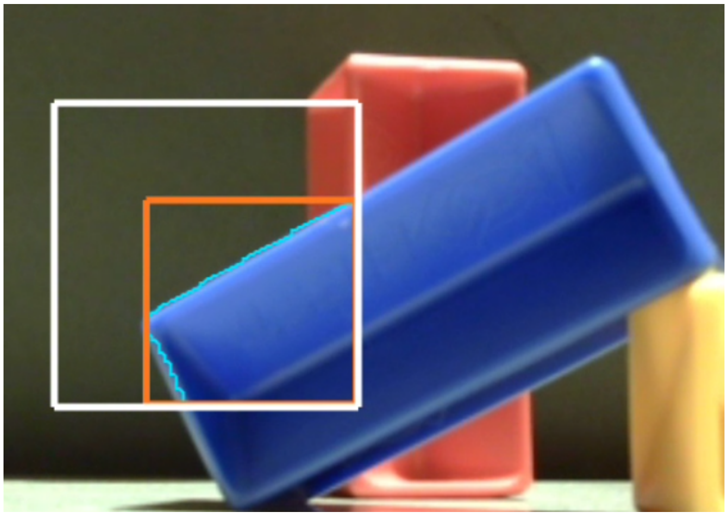


Fig. 202: Single blob discovery

But OpenCV can form and return **multiple Blobs** in a single set of results:

Without controls, OpenCV can easily form a **high number of Blobs** (at least 12 here):

And as mentioned above, some of those Blobs might have a **boxFit tilted** at some angle:

This tutorial's **next two pages** show how to manage these scenarios by **editing the OpMode's default settings**, and **accessing more OpenCV features** not covered in the Sample OpMode.

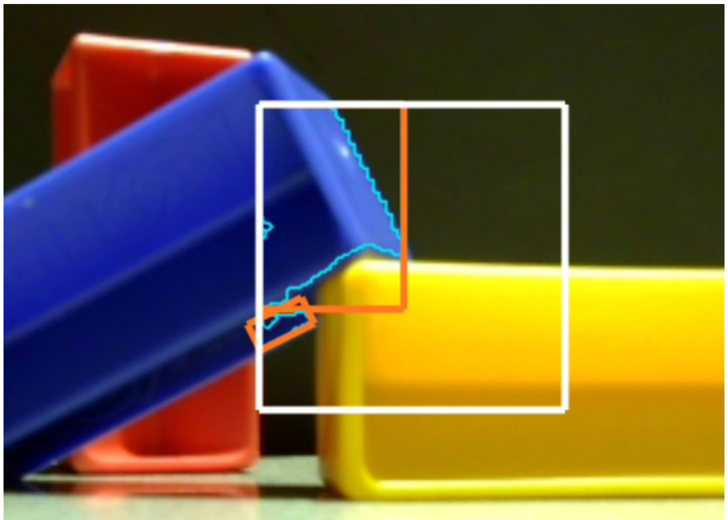


Fig. 203: Two blob discovery

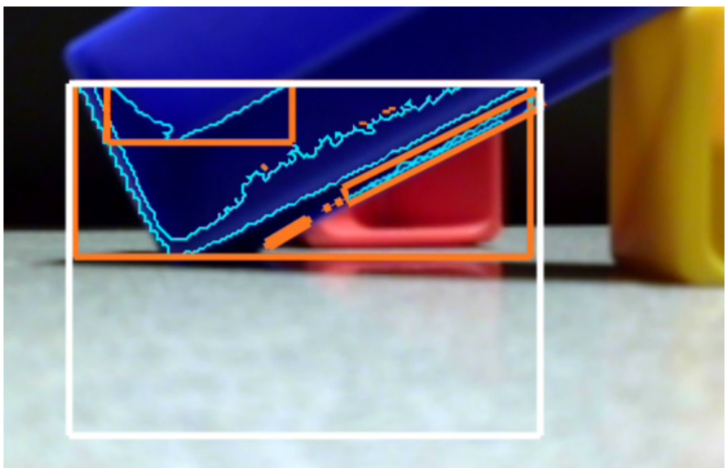


Fig. 204: Multiple blob discovery

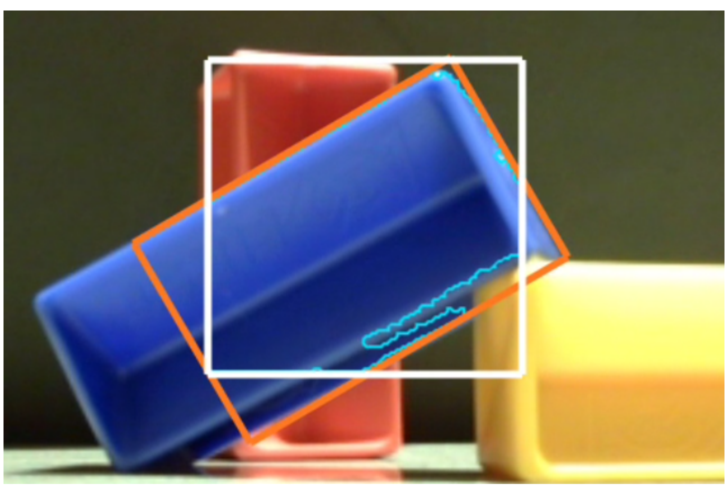


Fig. 205: Tilted Boxfit

Using boxFit Data for Position

A team's Autonomous code can evaluate boxFit data to navigate or guide the robot on the field.

Imagine your camera is on the robot, looking forward. **Underneath the camera** is your **intake mechanism**, perhaps a top grabber, sideways claw or spinner.

Fig. 206: Game piece targeting

OpenCV will report the data for this orange boxFit. Could your code use this data to **position the robot** directly in front of the game piece, for a better chance to collect it?

How would you do it?

Using boxFit Data for Manipulation

For advanced teams: imagine your webcam is on a grabber arm, looking down into the Submersible (from INTO THE DEEP).

Fig. 207: Targeting in clutter

Could the data from this boxFit (orange rectangle) help you **grab only the Blue Sample**?

Could this help in Autonomous **and** TeleOp?

More Documentation

This tutorial's next page called *Explore* covers **editing the OpMode's existing default settings**.

After that, the following page called *Challenge* shows how to **access more OpenCV features** not covered in the Sample OpMode.

Questions, comments and corrections to westsiderobotics@verizon.net

Color Locator (Explore)

Overview

This **Explore** page shows how to modify the default settings of the **ColorLocator** Sample OpMode. It assumes you have already followed this tutorial's previous *Discover* page, to open and test this OpMode.

ColorLocator has only two required **inputs**:

- target color range
- Region of Interest (ROI)

Both of these can be specified in multiple ways.

This Explore page covers **settings** that are already in the Sample:

- which contour types to process
- whether to draw contours in the preview
- pre-processing of the image
- camera resolution
- post-filter the Blob results
- post-sort the Blob results

Building the VisionPortal

The Sample OpMode first creates a "Color Blob Locator" **Processor** using the Java **Builder** pattern. This is the same Builder pattern used to create an AprilTag Processor, and previously a TensorFlow Processor.

The Sample OpMode then creates a **VisionPortal**, again using a Builder pattern. This includes adding the "Color Blob Locator" Processor to the VisionPortal.

The FTC VisionPortal was introduced in 2023. More information is available on the *ftc-docs VisionPortal Page*.

Target Color Range

The "target color" is actually a **range** of numerical color values, for a better chance of finding the desired color.

Each **Swatch** name (BLUE, RED, YELLOW, GREEN) has been pre-programmed with a range of color values to detect most shades of that color, in most lighting conditions.

The values for Red, Blue and Yellow were tuned for the plastic game pieces (called Samples) from INTO THE DEEP.

Select and read the Blocks **or** Java section below:

Blocks

The sample OpMode uses this Builder Block to specify the target color Swatch:

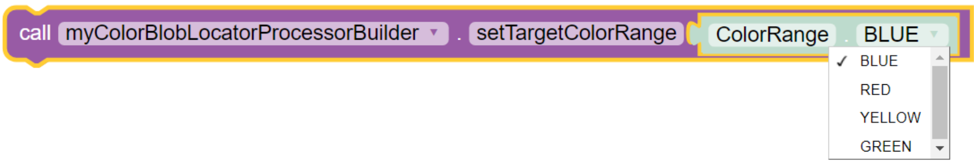


Fig. 208: Setting the Color Range Swatch

Use the drop-down list to select the Swatch of the desired target color range.

As an alternate, this Block can be replaced with the following Block in the Vision/ColorBlobLocator/Processor.Builder toolbox:

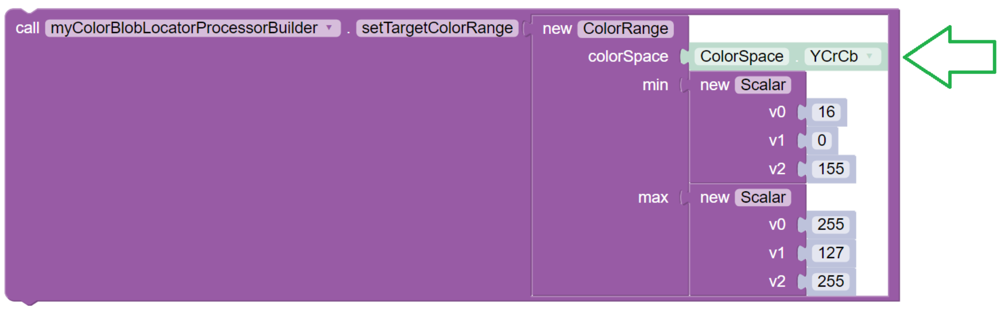


Fig. 209: Setting a custom color range

First use the drop-down list (green arrow) to choose the **Color Space** : YCrCb, HSV or RGB. Learn more about Color Spaces at the separate page in this tutorial.

Then select the numerical values in that Color Space to define the range of the target color.

The `min` and `max` fields relate to a corresponding pair of values, namely (v0, v0), (v1, v1) or (v2, v2).

Java

In the Sample OpMode, follow the instructions at about lines 70-75 to set the desired target color range at about line 110.

Use a predefined Swatch, or set a custom range in a specified Color Space (YCrCb, HSV or RGB). Learn more about Color Spaces at the separate page in this tutorial.

```
import org.opencv.core.Scalar;
.
.
// use a predefined color match
.setTargetColorRange(ColorRange.BLUE)
// Available predefined colors are: RED, BLUE, YELLOW, GREEN
.
// or define your own color match
.setTargetColorRange(new ColorRange(ColorSpace.YCrCb,
                                    new Scalar( 32, 176,  0),
                                    new Scalar(255, 255, 132)))
```

Region of Interest (ROI)

The Blocks and Java Sample OpModes give this description:

> *Focus the color locator by defining a RegionOfInterest (ROI) which you want to search. This can be the entire frame, or a sub-region defined using standard image coordinates or a normalized +/- 1.0 coordinate system. Use one form of the ImageRegion class to define the ROI.*

Caution: changing the ROI size and/or changing the camera resolution may require an adjustment to filtering by Area. Post-filtering is covered here at this tutorial's **Explore** page, and pre-filtering is covered at the following *Challenge* page.

Select and read the Blocks **or** Java section below:

Blocks

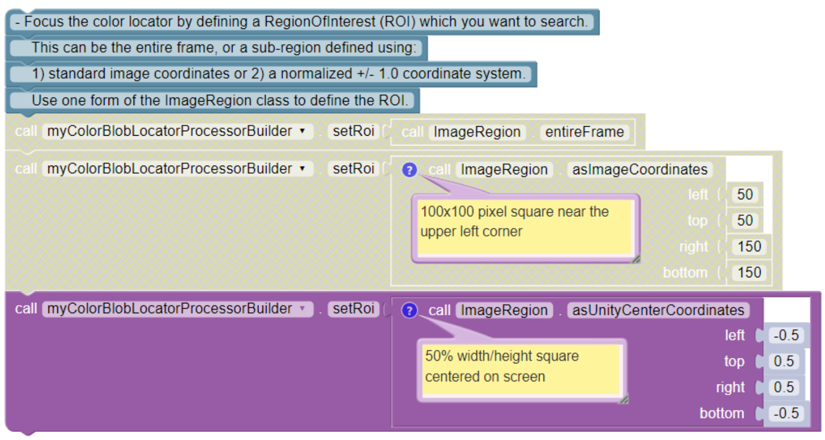


Fig. 210: Setting the ROI

Java

In the Sample OpMode, follow the instructions at about lines 77-83 to set the desired ROI at about line 112.

```
.setRoi(ImageRegion.entireFrame())
.
// 100x100 pixel square near the upper left corner
.setRoi(ImageRegion.asImageCoordinates(50, 50,  150, 150))
.
// 50% width/height square centered on screen
.setRoi(ImageRegion.asUnityCenterCoordinates(-0.5, 0.5, 0.5, -0.5))
```

Choice of Contours

The Blocks and Java Sample OpModes give this description:

> *Define which contours are included. You can get ALL the contours, or you can skip any contours that are completely inside another contour. note: EXTERNAL_ONLY helps to avoid bright reflection spots from breaking up areas of solid color.*

Also, the display of contours (in the previews) can be turned ON or OFF:

> Turning this on helps debugging but takes up valuable CPU time.

Select and read the Blocks **or** Java section below:

Blocks

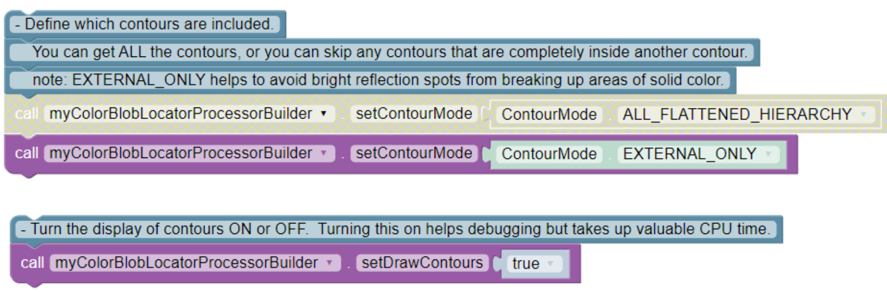


Fig. 211: Contour Choices

Java

In the Sample OpMode, follow the instructions at about lines 85-92 to set the desired contour mode and drawing setting at about lines 111 and 113, respectively.

```
// return all contours
.setContourMode(ColorBlobLocatorProcessor.ContourMode.ALL_FLATTENED_HIERARCHY)
.
// exclude contours inside other contours
.setContourMode(ColorBlobLocatorProcessor.ContourMode.EXTERNAL_ONLY)
.
// show contours in the DS and RC previews
.setDrawContours(true)
```

Image Pre-Processing

The default Sample OpMode purposely **blurs** the camera's image. This "pre-processing" happens **before** OpenCV performs Blob formation, thus affecting the contours seen in DS and RC previews.

The effect is very small (default kernel size of 5x5 pixels), but can significantly improve Blob formation, giving more useful results.

Blurring is one of three available image adjustments to improve processing results. You can experiment with these advanced tools, after studying their usage. See links at the section below called **More Documentation**.

The Blocks and Java Sample OpModes give this description:

Include any pre-processing of the image or mask before looking for Blobs.

There is some extra processing you can include to improve the formation of blobs. Using these features requires an understanding of how they may affect the final blobs. The “pixels” argument sets the NxN kernel size.

Blurring an image helps to provide a smooth color transition between objects, and smoother contours. The higher the number of pixels, the more blurred the image becomes. Note: Even “pixels” values will be incremented to satisfy the “odd number” requirement. Blurring too much may hide smaller features. A “pixels” size of 5 is good for a 320x240 image.

Erosion removes floating pixels and thin lines so that only substantive objects remain. Erosion can grow holes inside regions, and also shrink objects. A “pixels” value in the range of 2-4 is suitable for low res images.

Dilation makes objects more visible by filling in small holes, making lines appear thicker, and making filled shapes appear larger. Dilation is useful for joining broken parts of an object, such as when removing noise from an image. A “pixels” value in the range of 2-4 is suitable for low res images.

Select and read the Blocks **or** Java section below:

Blocks

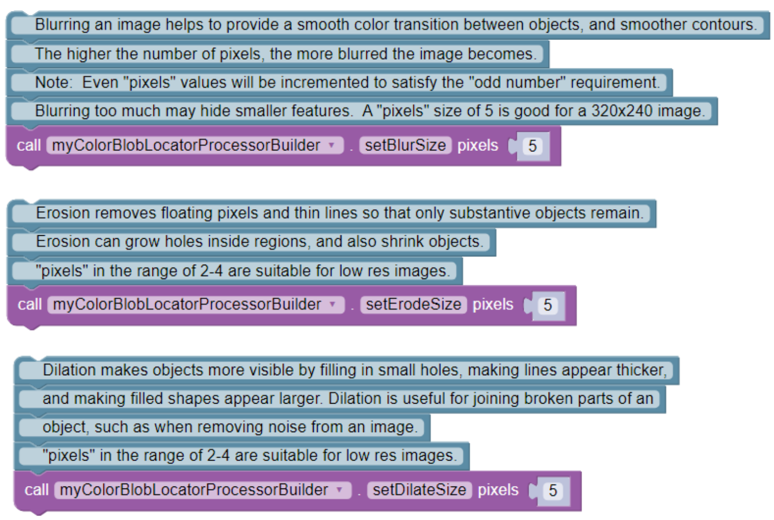


Fig. 212: Pre-processor Options

Java

In the Sample OpMode, follow the instructions at about lines 94-107 to set the desired pre-processing at about line 114.

```
.setBlurSize(int pixels)
.setErodeSize(int pixels)
.setDilateSize(int pixels)
```

Any of these pre-processing settings can be **disabled** by setting their pixel value to zero, or by removing the command.

In the FTC processor, any specified erosion is performed **before** dilation. This removes specular noise, then returns the remaining blobs to a size similar to their original size. (This also will **not** be on the final.)

Camera Resolution

The Sample OpMode uses a default camera resolution of 320 x 240 pixels, supported by most webcams and Android phone cameras. You may edit this resolution, subject to a trade-off between:

- computing performance, and
- image detail, possibly needed beyond ColorLocator.

Caution: changing the camera resolution and/or changing the ROI size may require an adjustment to filtering by Area. Post-filtering is covered here at this tutorial's **Explore** page, and pre-filtering is covered at the following *Challenge* page.

The Blocks and Java Sample OpModes give this description:

> Set the desired video resolution. Since a high resolution will not improve this process, choose a lower resolution that is supported by your camera. This will improve overall performance and reduce latency.

Select and read the Blocks **or** Java section below:

Blocks

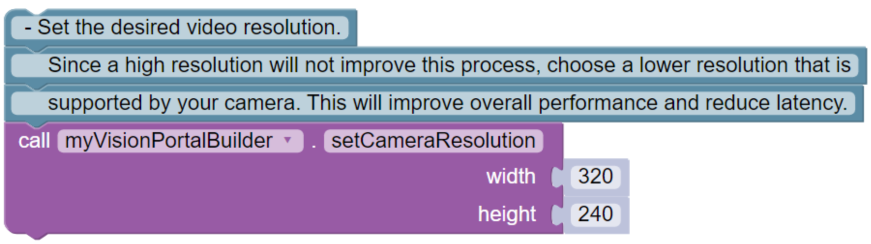


Fig. 213: Camera Resolution

Java

In the Sample OpMode, follow the instructions at about lines 121-123 to set the desired camera resolution at about line 131. This setting is made in the VisionPortal Builder, not the Processor Builder.

```
.setCameraResolution(new Size(320, 240))
```

Post-filter the Blob Results

After OpenCV has formed Blobs and provided results with the `getBlobs()` command (in Blocks and Java), your OpMode can **post-filter** or reduce the list.

Here the term "post-" means after Blob formation and **after the DS and RC previews**. So, you will still see contours and boxFits for **all Blobs**.

By default, the Sample OpMode uses a **Contour Area** filter of 50 pixels (minimum) to 20,000 pixels (maximum). The lower limit eliminates very small Blobs, while the upper limit is approximately the size of the default Region of Interest (ROI).

Caution: changing the ROI size and/or changing the camera resolution may require an adjustment to filtering by Area.

Tip: Remember that a Blob contour never extends beyond the ROI, although a boxFit may do so.

Why filter? A smaller list means faster processing, with fewer boxFits for your OpMode to evaluate.

You can experiment with increasing the lower limit, and observing the effect on Telemetry. Also experiment with the other filters for **Density** and **Aspect Ratio**.

The Blocks and Java Sample OpModes give this description:

> The list of Blobs can be filtered to remove unwanted Blobs. Note: All contours will be still displayed on the Stream Preview, but only those that satisfy the filter conditions will remain in the current list of "blobs". Multiple filters may be used. Use any of the following filters.
>
> **Util.filterByArea()** A Blob's area is the number of pixels contained within the contour. Filter out any that are too big or small. Start with a large range and then refine the range based on the likely size of the desired object in the viewfinder.
>
> **Util.filterByDensity()** A blob's density is an indication of how "full" the contour is. If you put a rubber band around the contour you would get the "Convex Hull" of the contour. The density is the ratio of Contour-area to Convex Hull-area.
>
> **Util.filterByAspectRatio()** A blob's aspect ratio is the ratio of **boxFit** long side to short side. A perfect square has an aspect ratio of 1. All others are > 1

Select and read the Blocks **or** Java section below:

Blocks

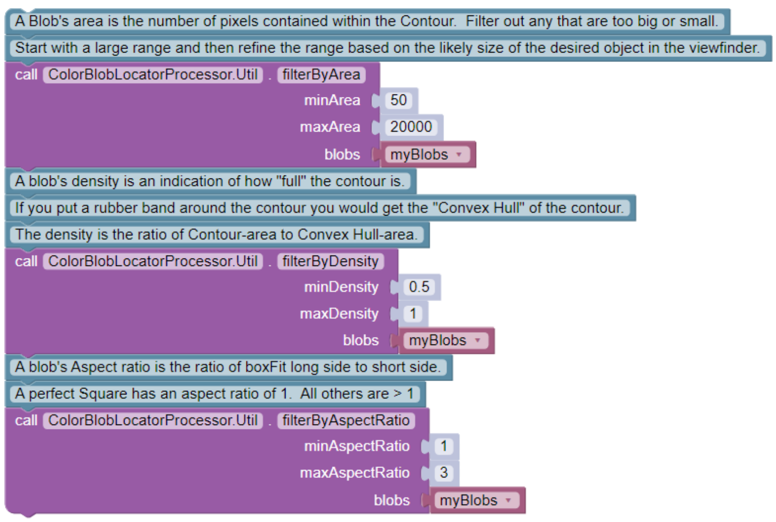


Fig. 214: Post Filter

Java

In the Sample OpMode, follow the instructions at about lines 147-164 to set the desired post-filtering at about line 166.

```
ColorBlobLocatorProcessor.Util.filterByArea(minArea, maxArea, blobs);
ColorBlobLocatorProcessor.Util.filterByDensity(minDensity, maxDensity, blobs);
ColorBlobLocatorProcessor.Util.filterByAspectRatio(minAspect, maxAspect, blobs);
```

Post-filtering commands should be placed **after** calling `getBlobs()` and **before** your OpMode's handling (or Telemetry) of the `getBlobs()` results. Remember this as you incorporate these tools into your team's larger OpModes.

Post-sort the Blob Results

After OpenCV has formed Blobs and provided results with the `getBlobs()` command (in Blocks and Java), your OpMode can **post-sort** the list.

By default, the Sample OpMode sorts by **Contour Area** in descending order (largest is first). This is an internally programmed sort, not appearing in the Sample OpMode. This default is overridden or replaced by any sort specified in the OpMode.

Why sort? A sorted list means your OpMode can process Blobs in a known order, perhaps allowing your code to quickly reach a "conclusion". Namely some logic condition (probably about boxFits) could be satisfied sooner, to exit the vision processing loop and move on to robot action.

The Blocks and Java Sample OpModes give this description:

> *The list of Blobs can be sorted using the same Blob attributes as listed above. No more than one sort call should be made. Sorting can use ascending or descending order.*

Select and read the Blocks **or** Java section below:

Blocks

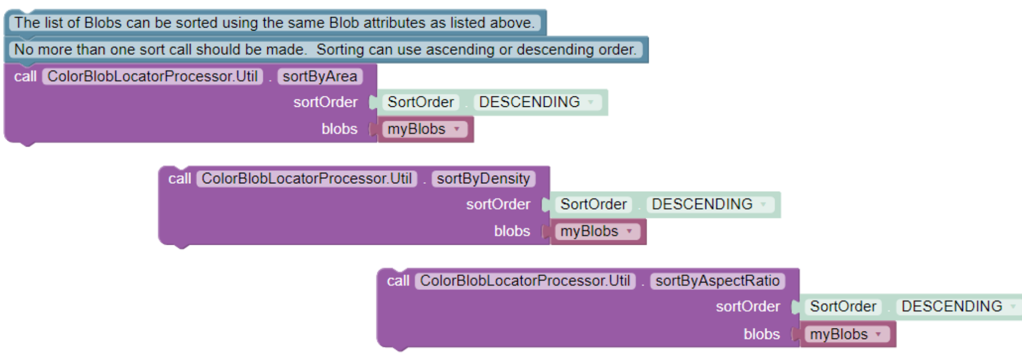


Fig. 215: Post Sort

Java

In the Sample OpMode, follow the instructions at about lines 169-173 to set the desired post-sorting immediately after those instructions.

```
ColorBlobLocatorProcessor.Util.sortByArea(SortOrder.DESCENDING, blobs);  // Default
ColorBlobLocatorProcessor.Util.sortByDensity(SortOrder.DESCENDING, blobs);
ColorBlobLocatorProcessor.Util.sortByAspectRatio(SortOrder.DESCENDING, blobs);
```

A post-sorting command should be placed **after** calling `getBlobs()` and any post-filtering, and **before** your OpMode's handling (or Telemetry) of the `getBlobs()` results. Remember this as you incorporate these tools into your team's larger OpModes.

More Documentation

How does OpenCV match colors here? The upper and lower values of the target color range are used to **threshold** the image's pixels and find those within the range. Technical information on thresholding is available at the OpenCV website for thresholding.

Technical information on Blur, Erosion and Dilation can be found here and at the OpenCV website for morphology.

Here's a conceptual note from co-developer @Windwoes:

> The command `getBlobs()` does not initiate or perform the processing (Blob formation). The processing is **happening continuously**; `getBlobs()` just obtains a reference to the latest results.

Next, this tutorial's *Challenge* page shows how to **access more OpenCV features** not covered in the Sample OpMode.

Questions, comments and corrections to westsiderobotics@verizon.net

Color Locator (Challenge)

Overview

This **Challenge** page introduces Color Locator settings that were **not mentioned in the Sample OpMode**. It assumes you have already followed this tutorial's previous pages:

- *Discover* page, to open and test the Sample OpMode
- *Explore* page, to edit settings mentioned in the Sample OpMode

Here are the additional ColorLocator settings covered in this page:

- pre-filtering, affecting Blob results and previews
- pre-sorting
- custom outline colors in the previews: ROI, contour, boxFit
- access a boxFit's corner points
- access all vertices of a contour
- access a boxFit's size and tilt angle
- create a horizonal rectangle around a tilted boxFit
- access the horizontal rectangle's size and location

Pre-Filter Intro

Here the term "pre-" means the filter criteria are implemented **before** the Blob formation results are passed further. Thus the DS and RC previews will **not display** any filtered-out contour or its boxFit. This can save CPU resources used to draw the outlines.

Likewise the resulting list of results will not include any filtered-out Blobs. A shorter list can help OpMode cycle time.

This contrasts with the post-filtering present in the Sample OpMode and discussed at this tutorial's Explore page. Pre-filtering is a setting that persists, while post-filtering is a one-time action performed on a single set of processing results.

Teams may wish to use both. Use a pre-filter to "clean up" the preview, which can appear chaotic with potentially dozens of Blobs. Then use a post-filter to focus on the particular boxFit results of interest.

Caution: changing the ROI size and/or changing the camera resolution may require an adjustment to filtering by Area.

Pre-Filter Programming

To apply a filter setting, use two steps:

- **set** the filter name and criteria
- **add** that filter to the existing Processor

The "Color Blob Locator" Processor must already be created; adding a filter is **not** part of the Builder pattern here. A pre-filter can be added before or after the VisionPortal is built.

In general, a pre-filter setting remains in place and cannot be edited. To "change" a pre-filter, it must be **removed** from the Processor, then **added** again.

Recall from this tutorial's *Explore* page, to edit settings mentioned in the Sample OpMode that filtering can be done by Contour Area, Blob Density and boxFit Aspect Ratio.

Multiple pre-filters can operate at the same time. A single common filter name **could** be used, if its criteria are defined, then added – then redefined, and added again, etc.

You might find it more versatile and convenient to create unique filter names, each separately managed (i.e. set criteria, add, remove, add again).

Select and read the Blocks **or** Java section below:

Blocks

These pre-filter Blocks are in the Vision/ColorBlobLocator/Pre-processing toolbox:

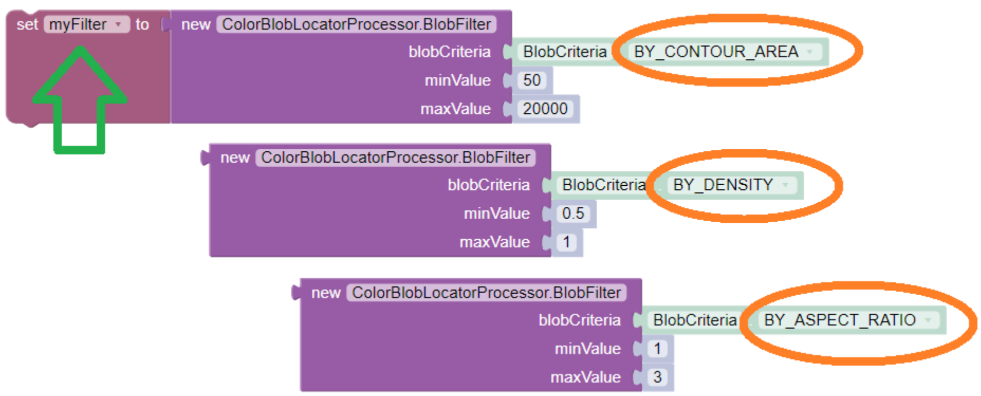


Fig. 216: Setting a pre-filter

The Blocks in the toolbox use the same variable name `myFilter` (green arrow) for the three available criteria (orange ovals). As noted above, be careful about using a single name for different pre-filters.

For multiple pre-filters, you might prefer unique names; see the orange arrows:

These Blocks for adding and removing filters are in the Vision/ColorBlobLocator/Processor toolbox:

Be careful to designate the correct filter to be added or removed; see the green arrows.

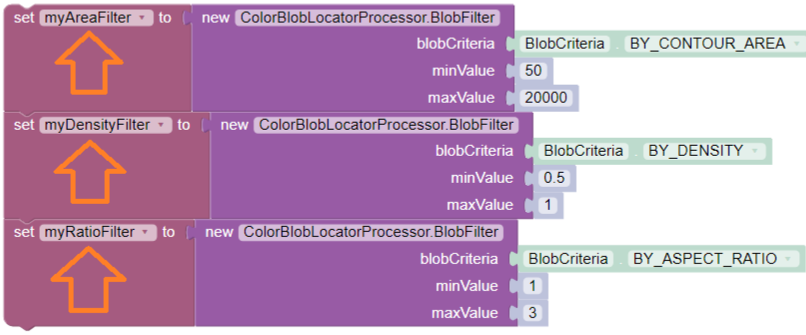


Fig. 217: Setting 3 filters

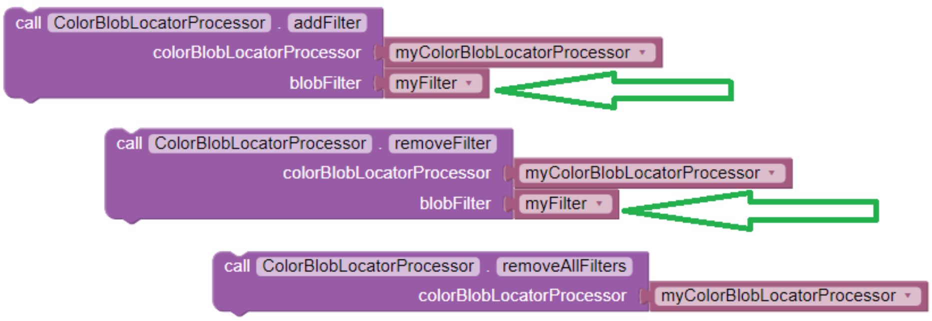


Fig. 218: Adding and Removing a filter

Java

For multiple pre-filters, you might prefer unique names:

```
myAreaFilter = new BlobFilter(BlobCriteria.BY_CONTOUR_AREA, 100, 20000);
myDensityFilter = new BlobFilter(BlobCriteria.BY_DENSITY, 0.5, 1.0);
myRatioFilter = new BlobFilter(BlobCriteria.BY_ASPECT_RATIO, 1.0, 10.0);
```

After defining a filter's criteria, add the filter to an already-existing Processor:

```
colorLocator.addFilter(myAreaFilter);
colorLocator.addFilter(myDensityFilter);
colorLocator.addFilter(myRatioFilter);
```

These methods can remove one or all filters from a Processor:

```
colorLocator.removeFilter(myAreaFilter);
colorLocator.removeFilter(myDensityFilter);
colorLocator.removeFilter(myRatioFilter);
colorLocator.removeAllFilters();
```

After removal, a filter can be re-added to the Processor.

Pre-sort

Here the term "pre-" also means the sort criteria are already established, and applied to the results of the Blob formation process.

This works the same as the post-sorting mentioned in the Sample OpMode and discussed at this tutorial's Explore page. Pre-sorting is a setting that persists, while post-sorting is a one-time action performed on a single set of processing results.

Only one sort (the last one applied) affects the final list of results provided for the OpMode to evaluate. Thus there is no benefit to using both pre-sort and post-sort.

To apply a sort, use two steps:

- define the sort name and criteria
- apply that sort to the existing Processor

The "Color Blob Locator" Processor must already be created; adding a sort is **not** part of the Builder pattern here. A pre-sort can be added before or after the VisionPortal is built.

In general, a pre-sort setting remains in place and cannot be removed or edited. To "change" a sort, simply define and apply another one, with the same or a unique name. The later `setSort()` will be in effect.

Reminder from this tutorial's *Explore* page: by default, the Sample OpMode sorts by **Contour Area** in descending order (largest is first). This is an internally programmed sort, not appearing in the Sample OpMode. This default is overridden or replaced by any pre-sort or post-sort specified in the OpMode.

Select and read the Blocks **or** Java section below:

Blocks

These pre-sort Blocks are in the Vision/ColorBlobLocator/Pre-processing toolbox:

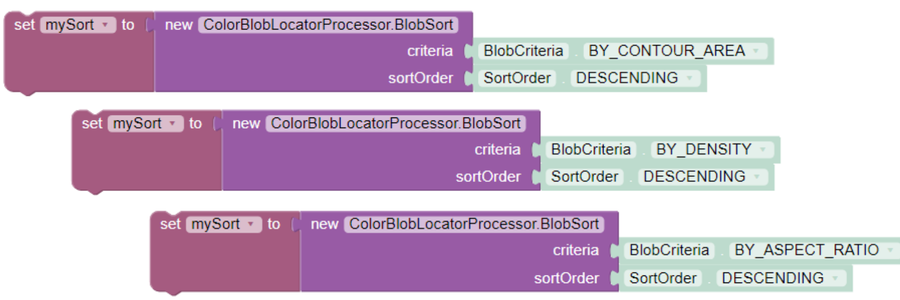


Fig. 219: Selecting the Sort Criteria

This Block for applying the named pre-sort is in the Vision/ColorBlobLocator/Processor toolbox:

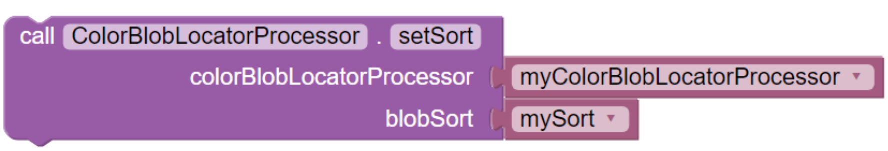


Fig. 220: Setting the Sort Criteria

Java

A generic pre-sort name works well, since only one sort can be in effect at a time:

```
mySort = new BlobSort(BlobCriteria.BY_CONTOUR_AREA, SortOrder.ASCENDING);
```

If you are experimenting with different pre-sort criteria, you might consider unique names:

```
myAreaSort = new BlobSort(BlobCriteria.BY_CONTOUR_AREA, SortOrder.ASCENDING);
myDensitySort = new BlobSort(BlobCriteria.BY_DENSITY, SortOrder.ASCENDING);
myRatioSort = new BlobSort(BlobCriteria.BY_ASPECT_RATIO, SortOrder.ASCENDING);
```

After defining a sort's criteria, apply the pre-sort to an already-existing Processor:

```
colorLocator.setSort(mySort);
```

Preview Colors

You can specify custom colors for the preview outlines of:

- Region of Interest (ROI)
- Blob contour
- boxFit rectangle

Select and read the Blocks **or** Java section below:

Blocks

Fig. 221: Setting custom outline colors

Java

Import the `Color` class if needed, then add any of the `.set...` methods to the Processor Builder pattern:

```
import android.graphics.Color;
.
.
.setBoxFitColor(Color.rgb(255, 120, 31))
.setRoiColor(Color.rgb(255, 255, 255))
.setContourColor(Color.rgb(3, 227, 252))
```

Use your own custom values, only from the RGB Color Space. See the separate tutorial page called **Color Spaces**.

boxFit Corners

An OpMode can access the four corner points of a boxFit rectangle.

Select and read the Blocks **or** Java section below:

Blocks

This Blocks Function retrieves, stores and displays the 4 corner points of the instant boxFit being processed by the OpMode:

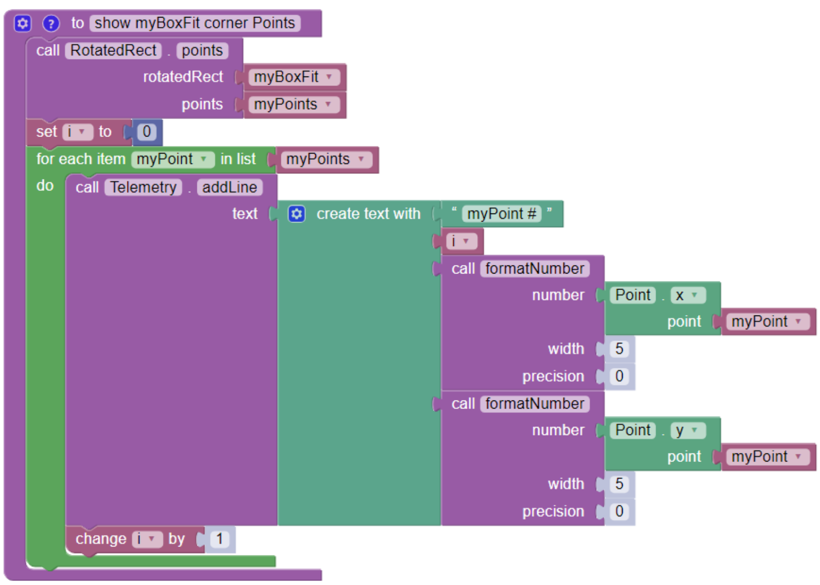


Fig. 222: Displaying Corner Points via Telemetry

The `.points` and `Point.x` and `Point.y` Blocks are in the "Vision/ColorBlobLocator/Blob data" toolbox.

The Function uses its own **For Loop** to cycle through the `myPoints` List of 4 points, clockwise from top left corner.

This Function operates inside the Sample OpMode's **For Loop** of all Blob results. The instant `myBoxFit` is the one being processed, returned from the preceding `.BoxFit` Block.

Java

This Java code retrieves, stores and displays the 4 corner points of the instant boxFit being processed by the OpMode:

```
// Display boxFit.points(), an array of the box's four (X, Y) corner points,
// clockwise from top left corner.
Point[] myBoxCorners = new Point[4];
boxFit.points(myBoxCorners);
// this points() method does not return values, it populates the argument
for (int i = 0; i < 4; i++)
{
    telemetry.addLine(String.format("boxFit corner %d (%d,%d)",
    i, (int) myBoxCorners[i].x, (int) myBoxCorners[i].y));
}
```

This code operates inside the Sample OpMode's **For Loop** of all Blob results. The instant `boxFit` is the one being processed, returned from the preceding `getBoxFit()` method.

Contour Vertices

Blob contours are irregular and hard to process in code; it's easier to work with boxFit rectangles.

But the contour's outer points can be accessed by an OpMode. The result is a list of (X, Y) coordinates, with origin at the top left corner of the camera's image. X increases to the right, Y increases downward.

Select and read the Blocks **or** Java section below:

Blocks

The following Blocks Function retrieves, stores and displays a List of all the vertex Points of the contour of the instant Blob being processed by the OpMode.

The List can be as short as 4 values, or dozens of values for jagged contours.

Note that the `.points` Block (used above for boxFit corners) retrieves and stores the List within the same Block. This `.ContourPoints` Block only retrieves the List, to be assigned to a separate variable Block.

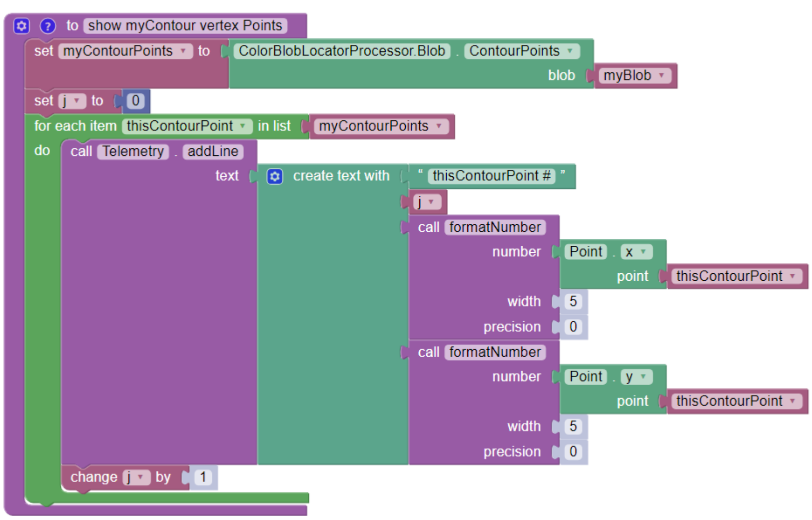


Fig. 223: Getting the list of contour points

The `.ContourPoints` and `Point.x` and `Point.y` Blocks are in the "Vision/ColorBlobLocator/Blob data" toolbox.

The Function uses its own **For Loop** to cycle through the `myContourPoints` List, of undetermined length (could be very long).

This Function operates inside the Sample OpMode's **For Loop** of all Blob results. The instant `myBlob` is the one being processed by that outer For Loop.

Java

This Java code retrieves, stores and displays a List of all the vertex Points of the contour of the instant Blob being processed by the OpMode.

```
import org.opencv.core.Point;
.
.
// Display getContourPoints(), an array of the contour's many (X, Y) vertices
Point[] myContourPoints;
myContourPoints = b.getContourPoints();
int j = 0;
for(Point thisContourPoint : myContourPoints)
{
    telemetry.addLine(String.format("contour vertex %d (%d,%d)",
          j, (int) thisContourPoint.x, (int) thisContourPoint.y));
          j += 1;
}
```

This Function operates inside the Sample OpMode's **For Loop** of all Blob results. The instant Blob b is the one being processed by that outer For Loop.

Not covered here is one feature available in Java only:

```
MatOfPoint myContour = getContour()
```

This method returns a matrix unique to the OpenCV library. The matrix object can convert itself to an array, as follows:

```
MatOfPoint myContour;
Point[] myContourPoints;
myContour = b.getContour();
myContourPoints = myContour.toArray();
```

This code seems to give the same set of points as `getContourPoints()` shown above.

boxFit Size and Angle

These simple fields were not demonstrated in the Sample OpMode.

The boxFit size variable contains two fields which must be accessed individually, as shown below.

If the boxFit is horizontal (parallel to the ROI), its angle might be 0 or 90 degrees, often jumping between the two values. At 90 degrees, height and width become switched. Your OpMode code needs to account for this scenario.

Likewise, the boxFit angle is sometimes reported as clockwise from vertical, rather than counterclockwise from horizontal. More discussion is here.

Select and read the Blocks **or** Java section below:

Blocks

These Blocks are in the "Vision/ColorBlobLocator/Blob data" toolbox.

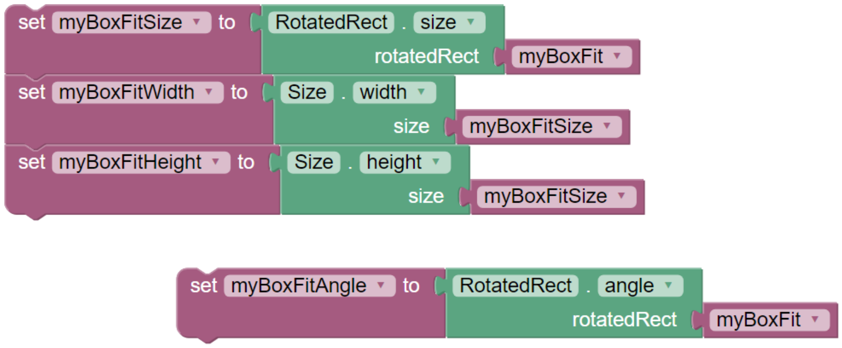


Fig. 224: boxFit properties

Java

Here's a modified version of the Sample OpMode's telemetry code, to display only the size and angle of the instant boxFit being processed.

```
org.opencv.core.Size myBoxFitSize;
for(ColorBlobLocatorProcessor.Blob b : blobs)
{
            RotatedRect boxFit = b.getBoxFit();
            myBoxFitSize = boxFit.size;
            telemetry.addData("width", myBoxFitSize.width);
            telemetry.addData("height", myBoxFitSize.height);
            telemetry.addData("angle", boxfit.angle);
}
```

The Java class `Size` here is different than another class of the same simple name. OnBot Java and Android Studio do not allow imports of identical simple classnames.

In fact OnBot Java will not allow the import of this version, even if the other version (*android.util.Size*) is not used in the OpMode. Instead, declare the variable with the full classname, as shown in the first line above.

Horizontal Rectangle

You might prefer to process only **horizontal** best-fit rectangles, parallel to the ROI, not tilted.

OpenCV can generate a best-fit rectangle for a boxFit, whether tilted or not. This is **not** a "forced horizonal" boxFit, rotated in place. The new horizontal rectangle simply touches and encloses the outer corners of the boxFit.

If the boxFit is tilted, the new horizontal rectangle will be larger. If the boxFit already had an angle of 0 (or 90) degrees, the new rectangle will be identical.

In Blocks and Java, the command `boundingRect()` accepts a boxFit of type `RotatedRect` and returns a horizontal rectangle of type `Rect`. The new rectangle is not drawn or depicted in the preview.

Select and read the Blocks **or** Java section below:

Blocks

These Blocks are in the "Vision/ColorBlobLocator/Blob data" toolbox.

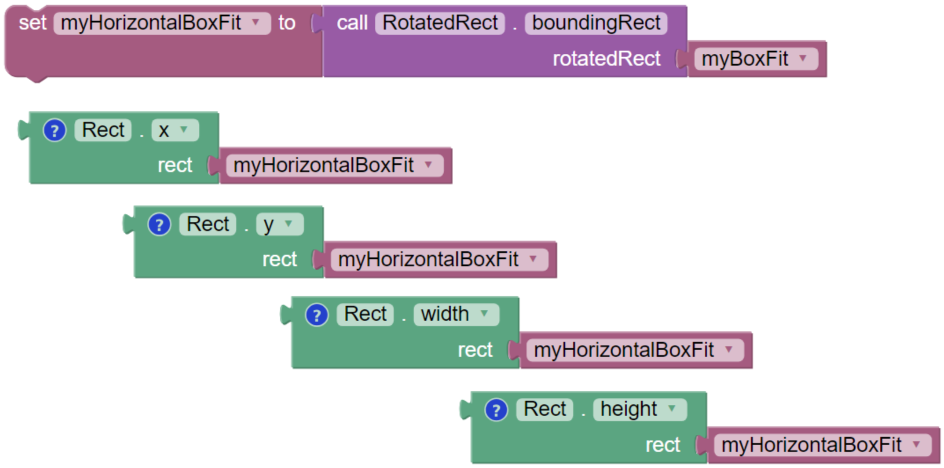


Fig. 225: Defining the horizontal box

The (X, Y) values are the top left corner of the new horizontal rectangle, in the full image reference frame.

Java

Here's a modified version of the Sample OpMode's telemetry code, to display only the **top left corner** and **size** of the horizontal rectangle around the boxFit being processed.

```
import org.opencv.core.Rect;
.
.
for(ColorBlobLocatorProcessor.Blob b : blobs)
{
                RotatedRect boxFit = b.getBoxFit();
                Rect myHorizontalBoxFit = boxFit.boundingRect();
                telemetry.addData("top left X", myHorizontalBoxFit.x);
                telemetry.addData("top left Y", myHorizontalBoxFit.y);
                telemetry.addData("width", myHorizontalBoxFit.width);
                telemetry.addData("height", myHorizontalBoxFit.height);
}
```

In this case, OnBot Java and Android Studio found no conflicts with the import of class `Rect`.

Pay attention to classes and fields:

- boxFit is of Java type `RotatedRect`, even though it's not usually rotated
- the new method `boundingRect()` returns an object of type `Rect`
- the `Size` and `Rect` classes both have fields named `height` and `width`

Advanced Development

Searching for multiple colors is possible by building **multiple processors** and adding them to the same VisionPortal. This allows different ROIs, for example, that can overlap if desired.

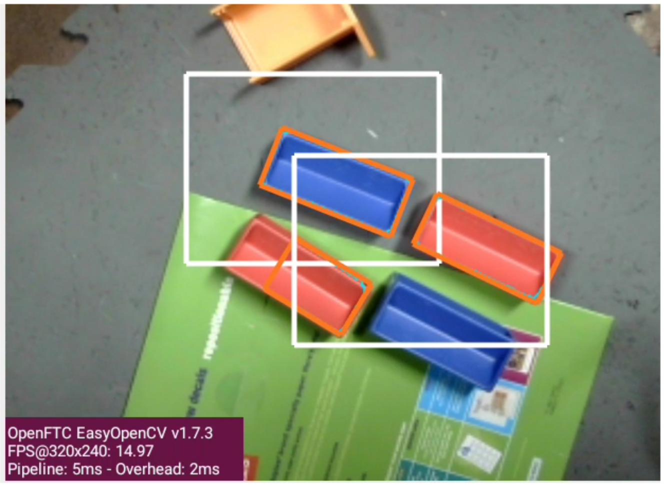


Fig. 226: Using two processors

This ends the tutorial's 3 pages on ColorLocator:

- *Discover*,
- *Explore*,
- **Challenge**

The final page of this tutorial provides optional info on *Color Spaces*.

Best of luck as you apply these tools to your Autonomous and TeleOp OpModes!

Questions, comments and corrections to westsiderobotics@verizon.net

Color Spaces

Overview

This page of the FTC Color Processing tutorial introduces **Color Spaces**.

OpenCV can process color information using any one of several Color Spaces, which are methods to describe an exact shade and brightness.

This page describes 3 choices available in the FTC SDK:

- RGB (Red, Green, Blue)
- HSV (Hue, Saturation, Value)
- YCrCb (Luminance, Chrominance red, Chrominance blue)

Each Color Space uses 3 numbers, 0 to 255, to describe a particular color.

RGB Color Space

RGB is a common Color Space, easy to understand. Its 3 components are:

- **Red** (0 - 255)
- **Green** (0 - 255)
- **Blue** (0 - 255)

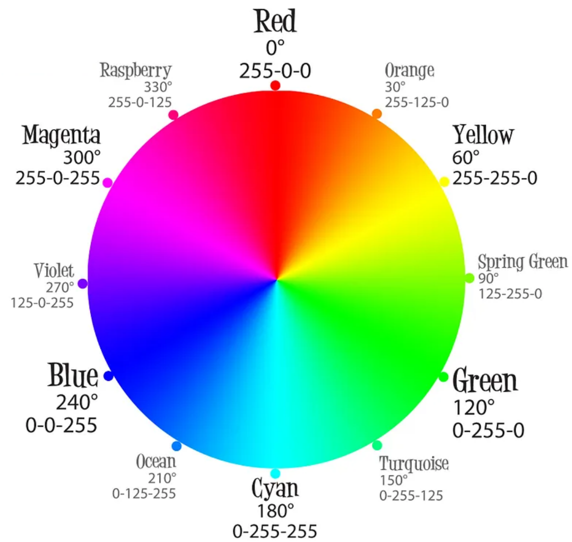


Fig. 227: RGB Color Wheel

Pure Red has values 255 red, 0 green, 0 blue. Pure Green has 0 red, 255 green, 0 blue.

Magenta is a blend of Red and Blue, so its values are 255 red, 0 green, and 255 blue.

Here's a useful way to visualize the RGB Color Space, with one axis for each component:

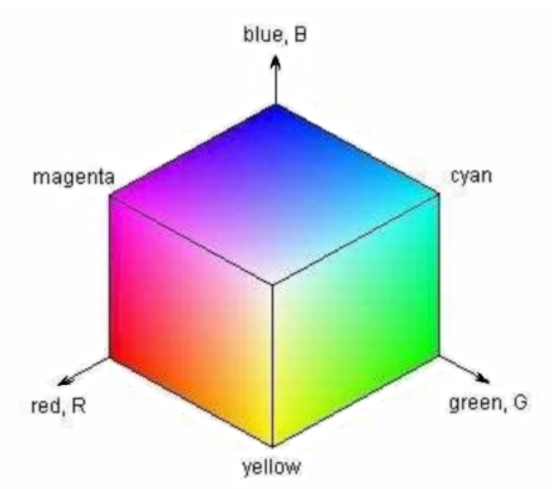


Fig. 228: RGB Cube Visualization

Each near-side external face of this box has the maximum value for one component. Every shade of color on the top face, for example, has a Blue component of 255.

The nearest corner is **White**, with RGB values of (255, 255, 255). Namely, full values of Red, Green and Blue light will combine to appear as white light.

Where is **Black**? It's the opposite corner, at the origin, hidden in this view. Its values are (0, 0, 0) – no color at all.

This RGB system is used only for light-based colors, including video. It does not apply for painted colors, or printed colors, which use other color systems.

Tip: Mixing red paint, green paint and blue paint will **not** create white paint!

Technical information is available at Wikipedia Color Spaces.

HSV Color Space

Another Color Space used by OpenCV is **HSV**: Hue, Saturation and Value.

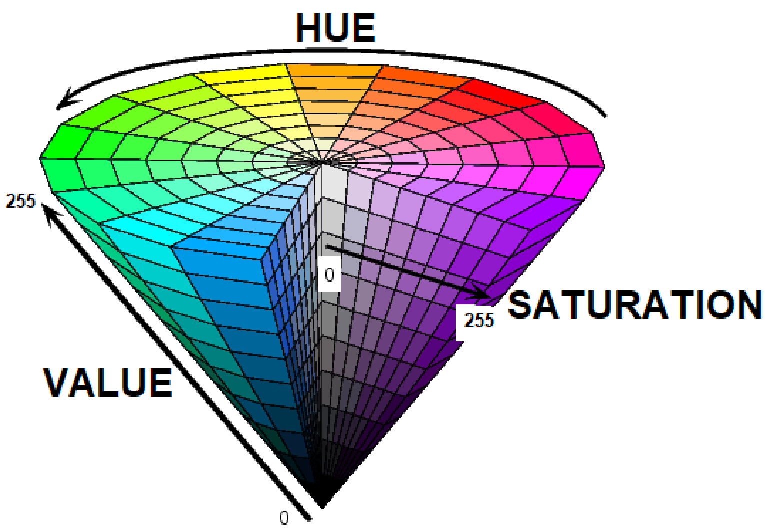


Fig. 229: HSV Cone Visualization

Hue is the actual shade of color; see the familiar color wheel on top.

Saturation measures the amount of white: a lower value is whiter, or more grey. On the HSV cone, see the outward arrow for Saturation. The highest value of 255 is the fully saturated color (no white).

Value measures brightness; see the upward arrow on the HSV cone. The top face of the cone (Value = 255), is the fully bright color. Black is found at the lower tip, Value = 0.

Technical information is available at Wikipedia HSL/HSV.

YCrCb Color Space

A third Color Space used by OpenCV is **YCrCb**.

The Y is **Luminance** or brightness, while Cr and Cb are red and blue components of **Chrominance**. Technical information is available at Wikipedia YCbCr.

The YCrCb Color Space offers efficient computation of color processing, and is widely used in video applications.

Some online documentation refers to a Color Space called **YCbCr**. This is the same system, with the last 2 values reversed.

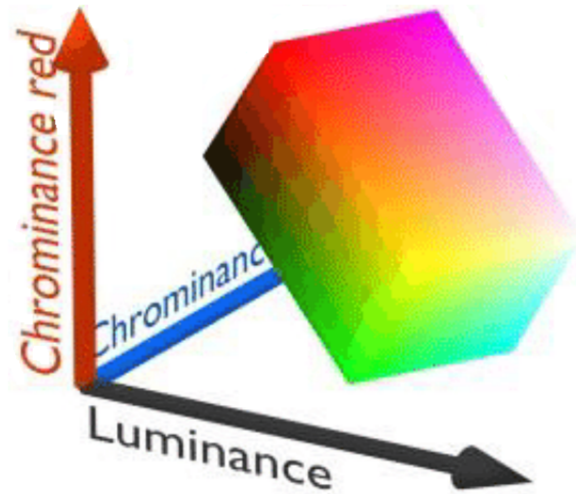


Fig. 230: YCrCb Visualization

How to choose?

Use the Color Space that's convenient for you. RGB is easy to understand, while YCrCb may offer better computational performance (if needed).

It's easy to find free public websites to convert the 3 values from one Color Space into the corresponding 3 values from another Color Space.

When converting to HSV, some online sites give Hue in degrees (0 to 360), and Saturation and Value as percentages (0 to 100). Apply these (as a proportion of the maximum) to 255, for values to use in the FTC **Color Locator** processor.

The **Color Locator** processor can use any of these three Color Spaces. The simple **Color Sensor** processor uses YCrCb internally, but reports results in RGB only.

Questions, comments and corrections to westsiderobotics@verizon.net

Much credit to developer and Sample OpMode author @gearsincorg, EasyOpenCV developer @Windwoes, FTC Blocks developer @lizlooney, and the open-source team at OpenCV.

Compatibility

This new software includes two Color Processors, each compatible with the FTC VisionPortal introduced in 2023. These processors can run alongside an AprilTag processor, and replace the TensorFlow processor (removed in 2024).

These new processors can be used on the usual FTC cameras:

- any UVC-compatible webcam
- the built-in camera of an FTC-supported Android phone (as Robot Controller)

This does **not** include vision sensors such as HuskyLens and LimeLight 3A, which do not use the FTC VisionPortal.

Two Processors

The new software includes these processors:

- **Color Sensor** - detects the color of a specified zone in the camera's view
- **Color Locator** - gives detailed info on clusters of a specified color, in a specified zone

This tutorial has a *Color Sensor* page, showing how to use the Sample OpMode called `ConceptVisionColorSensor`.

For the **Color Locator** processor, the color "clusters" are called **Blobs**. As listed above, this tutorial offers one page on Color Blob Concepts, and three pages covering the Sample OpMode called `ConceptVisionColorLocator`.

The Sample OpModes are available in **FTC Blocks**, and in **Java** for use in OnBot Java or Android Studio. Each programming section of this tutorial has a Blocks tab and a Java tab.

Next Steps

Time to get started!

Following this tutorial in order, first try the Sample OpMode for *Color Sensor*.

Then read about **Color Blob Concepts**, and try the **Color Locator** Sample OpMode.

Soon you'll be ready to add one or both features to your Autonomous OpModes – perhaps even to help automate your TeleOp!

Questions, comments and corrections to westsiderobotics@verizon.net

21.7 Advanced Topics

Advanced Topics for Programmers

21.7.1 Changing PID Coefficients

The REV Robotics Control Hub and REV Robotics Expansion Hub allow a user to change the PID coefficients used for closed loop motor control. The PID coefficients are channel and mode specific.

The following op mode uses an extended or enhanced DcMotor class (called "DcMotorEx") to change the PID coefficients for the RUN_USING_ENCODER mode for a motor named "left_drive". The op mode uses the setPIDCoefficients method of the DcMotorEx class to change the values. This method is not available with the standard DcMotor class.

Note that changes made to the PID coefficients do not persist if you power cycle the REV Robotics Control Hub or REV Robotics Expansion Hub. If you need your changes to the PID to persist, you should consider modifying your op mode to store state information on the Control Hub or Android phone. The Android Developer website has a tutorial on how to save data from your app onto an Android device here

```java
package org.firstinspires.ftc.teamcode;

import com.qualcomm.robotcore.eventloop.opmode.Autonomous;
import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.hardware.DcMotor;
import com.qualcomm.robotcore.hardware.DcMotorEx;
import com.qualcomm.robotcore.hardware.PIDCoefficients;

/**
```

(continues on next page)

(continued from previous page)

```
 * Created by tom on 9/26/17.
 * This assumes that you are using a REV Robotics Control Hub or REV Robotics Expansion Hub
 * as your DC motor controller.  This op mode uses the extended/enhanced
 * PID-related functions of the DcMotorEx class.
 */

@Autonomous(name="Concept: Change PID", group = "Concept")
public class ConceptChangePID extends LinearOpMode {

    // our DC motor.
    DcMotorEx motorExLeft;

    public static final double NEW_P = 2.5;
    public static final double NEW_I = 0.1;
    public static final double NEW_D = 0.2;

    public void runOpMode() {
        // get reference to DC motor.
        // since we are using the Control Hub or Expansion Hub,
        // cast this motor to a DcMotorEx object.
        motorExLeft = (DcMotorEx)hardwareMap.get(DcMotor.class, "left_drive");

        // wait for start command.
        waitForStart();

        // get the PID coefficients for the RUN_USING_ENCODER  modes.
        PIDCoefficients pidOrig = motorExLeft.getPIDCoefficients(DcMotor.RunMode.RUN_USING_ENCODER);

        // change coefficients using methods included with DcMotorEx class.
        PIDCoefficients pidNew = new PIDCoefficients(NEW_P, NEW_I, NEW_D);
        motorExLeft.setPIDCoefficients(DcMotor.RunMode.RUN_USING_ENCODER, pidNew);

        // re-read coefficients and verify change.
        PIDCoefficients pidModified = motorExLeft.getPIDCoefficients(DcMotor.RunMode.RUN_USING_
→ENCODER);

        // display info to user.
        while(opModeIsActive()) {
            telemetry.addData("Runtime", "%.03f", getRuntime());
            telemetry.addData("P,I,D (orig)", "%.04f, %.04f, %.0f",
                    pidOrig.p, pidOrig.i, pidOrig.d);
            telemetry.addData("P,I,D (modified)", "%.04f, %.04f, %.04f",
                    pidModified.p, pidModified.i, pidModified.d);
            telemetry.update();
        }
    }
}
```

Note that the actual change of the PID coefficients occurs on the motor controller that is controlling the selected motor. An alternate way to adjust the PID coefficients is to use the extended/enhanced PID-related methods of the DcMotorControllerEx class:

```
package org.firstinspires.ftc.teamcode;

import com.qualcomm.robotcore.eventloop.opmode.Autonomous;
import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.hardware.DcMotor;
import com.qualcomm.robotcore.hardware.DcMotorControllerEx;
import com.qualcomm.robotcore.hardware.DcMotorEx;
```

(continues on next page)

(continued from previous page)

```java
import com.qualcomm.robotcore.hardware.PIDCoefficients;

/**
 * Created by tom on 9/26/17.
 * This assumes that you are using a REV Robotics Control Hub or REV Robotics Expansion Hub
 * as your DC motor controller. This op mode uses the extended/enhanced
 * PID-related functions of the DcMotorControllerEx class.
 */

@Autonomous(name="Concept: Change PID Controller", group = "Examples")
public class ConceptChangePIDController extends LinearOpMode {

    // our DC motor.
    DcMotor motorLeft;

    public static final double NEW_P = 2.5;
    public static final double NEW_I = 0.1;
    public static final double NEW_D = 0.2;

    public void runOpMode() {
        // get reference to DC motor.
        motorLeft = hardwareMap.get(DcMotor.class, "left_drive");

        // wait for start command.
        waitForStart();

        // get a reference to the motor controller and cast it as an extended functionality␣
↪controller.
        // we assume it's a REV Robotics Control Hub or REV Robotics Expansion Hub (which supports␣
↪the extended controller functions).
        DcMotorControllerEx motorControllerEx = (DcMotorControllerEx)motorLeft.getController();

        // get the port number of our configured motor.
        int motorIndex = ((DcMotorEx)motorLeft).getPortNumber();

        // get the PID coefficients for the RUN_USING_ENCODER  modes.
        PIDCoefficients pidOrig = motorControllerEx.getPIDCoefficients(motorIndex, DcMotor.RunMode.
↪RUN_USING_ENCODER);

        // change coefficients.
        PIDCoefficients pidNew = new PIDCoefficients(NEW_P, NEW_I, NEW_D);
        motorControllerEx.setPIDCoefficients(motorIndex, DcMotor.RunMode.RUN_USING_ENCODER, pidNew);

        // re-read coefficients and verify change.
        PIDCoefficients pidModified = motorControllerEx.getPIDCoefficients(motorIndex, DcMotor.
↪RunMode.RUN_USING_ENCODER);

        // display info to user.
        while(opModeIsActive()) {
            telemetry.addData("Runtime", "%.03f", getRuntime());
            telemetry.addData("P,I,D (orig)", "%.04f, %.04f, %.0f",
                    pidOrig.p, pidOrig.i, pidOrig.d);
            telemetry.addData("P,I,D (modified)", "%.04f, %.04f, %.04f",
                    pidModified.p, pidModified.i, pidModified.d);
            telemetry.update();
        }
    }
}
```

21.7.2 Changing PIDF Coefficients

The REV Robotics Control Hub or REV Robotics Expansion Hub allows a user to change the PIDF coefficients used for closed loop motor control. The PIDF coefficients are specific to each channel (motor port) and to each RunMode.

The following sample OpMode uses an extended or enhanced DcMotor class (called "DcMotorEx") to change the PIDF coefficients for the RUN_USING_ENCODER RunMode for a motor named "left_drive". The OpMode uses the setPIDFCoefficients method of the DcMotorEx class to change the values. This method is not available with the standard DcMotor class.

Note that changes made to the PIDF coefficients do not persist if you power cycle the REV Robotics Control Hub or REV Robotics Expansion Hub. If you need your changes to persist, consider modifying your OpMode to store state information on the Control Hub or Android phone. The Android Developer website has a tutorial on how to save data from your app onto an Android device here

```
package org.firstinspires.ftc.teamcode;

import com.qualcomm.robotcore.eventloop.opmode.Autonomous;
import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.hardware.DcMotor;
import com.qualcomm.robotcore.hardware.DcMotorEx;
import com.qualcomm.robotcore.hardware.PIDFCoefficients;

/**
 * Created by Tom on 9/26/17.  Updated 9/24/2021 for PIDF.
 * This assumes that you are using a REV Robotics Control Hub or REV Robotics Expansion Hub
 * as your DC motor controller.  This OpMode uses the extended/enhanced
 * PIDF-related functions of the DcMotorEx class.
 */

@Autonomous(name="Concept: Change PIDF", group = "Concept")
public class ConceptChangePIDF extends LinearOpMode {

    // our DC motor
    DcMotorEx motorExLeft;

    public static final double NEW_P = 2.5;
    public static final double NEW_I = 0.1;
    public static final double NEW_D = 0.2;
    public static final double NEW_F = 0.5;
    // These values are for illustration only; they must be set
    // and adjusted for each motor based on its planned usage.

    public void runOpMode() {
        // Get reference to DC motor.
        // Since we are using the Control Hub or Expansion Hub,
        // cast this motor to a DcMotorEx object.
        motorExLeft = (DcMotorEx)hardwareMap.get(DcMotor.class, "left_drive");

        // wait for start command
        waitForStart();

        // Get the PIDF coefficients for the RUN_USING_ENCODER RunMode.
        PIDFCoefficients pidfOrig = motorExLeft.getPIDFCoefficients(DcMotor.RunMode.RUN_USING_
↪ENCODER);

        // Change coefficients using methods included with DcMotorEx class.
        PIDFCoefficients pidfNew = new PIDFCoefficients(NEW_P, NEW_I, NEW_D, NEW_F);
        motorExLeft.setPIDFCoefficients(DcMotor.RunMode.RUN_USING_ENCODER, pidfNew);

        // Re-read coefficients and verify change.
```

(continues on next page)

(continued from previous page)

```
        PIDFCoefficients pidfModified = motorExLeft.getPIDFCoefficients(DcMotor.RunMode.RUN_USING_
↪ENCODER);

        // display info to user
        while(opModeIsActive()) {
            telemetry.addData("Runtime (sec)", "%.01f", getRuntime());
            telemetry.addData("P,I,D,F (orig)", "%.04f, %.04f, %.04f, %.04f",
                    pidfOrig.p, pidfOrig.i, pidfOrig.d, pidfOrig.f);
            telemetry.addData("P,I,D,F (modified)", "%.04f, %.04f, %.04f, %.04f",
                    pidfModified.p, pidfModified.i, pidfModified.d, pidfModified.f);
            telemetry.update();
        }
    }
}
```

Note that the actual change of the PIDF coefficients occurs **on the motor controller** that is controlling the selected motor. An alternate way to adjust the PIDF coefficients is to use the extended/enhanced **PIDF-related methods of the DcMotorControllerEx class**, as follows:

```
package org.firstinspires.ftc.teamcode;

import com.qualcomm.robotcore.eventloop.opmode.Autonomous;
import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.hardware.DcMotor;
import com.qualcomm.robotcore.hardware.DcMotorControllerEx;
import com.qualcomm.robotcore.hardware.DcMotorEx;
import com.qualcomm.robotcore.hardware.PIDFCoefficients;

/**
 * Created by Tom on 9/26/17.  Updated 9/24/2021 for PIDF.
 * This assumes that you are using a REV Robotics Control Hub or REV Robotics Expansion Hub
 * as your DC motor controller.  This OpMode uses the extended/enhanced
 * PIDF-related functions of the DcMotorControllerEx class.
 */

@Autonomous(name="Concept: Change PIDF Controller", group = "Concept")
public class ConceptChangePIDFController extends LinearOpMode {

    // our DC motor
    DcMotor motorLeft;

    public static final double NEW_P = 2.5;
    public static final double NEW_I = 0.1;
    public static final double NEW_D = 0.2;
    public static final double NEW_F = 0.5;
    // These values are for illustration only; they must be set
    // and adjusted for each motor based on its planned usage.

    public void runOpMode() {
        // get reference to DC motor.
        motorLeft = hardwareMap.get(DcMotor.class, "left_drive");

        // wait for start command.
        waitForStart();

        // Get a reference to the motor controller and cast it as an extended functionality␣
↪controller.
        // We assume it's a REV Robotics Control Hub or REV Robotics Expansion Hub, which supports␣
↪the extended controller functions.
```

(continues on next page)

(continued from previous page)

```
        DcMotorControllerEx motorControllerEx = (DcMotorControllerEx)motorLeft.getController();

        // Get the port number of our configured motor.
        int motorIndex = ((DcMotorEx)motorLeft).getPortNumber();

        // Get the PIDF coefficients for the RUN_USING_ENCODER RunMode.
        PIDFCoefficients pidfOrig = motorControllerEx.getPIDFCoefficients(motorIndex, DcMotor.
↪RunMode.RUN_USING_ENCODER);

        // change coefficients
        PIDFCoefficients pidfNew = new PIDFCoefficients(NEW_P, NEW_I, NEW_D, NEW_F);
        motorControllerEx.setPIDFCoefficients(motorIndex, DcMotor.RunMode.RUN_USING_ENCODER,␣
↪pidfNew);

        // Re-read coefficients and verify change.
        PIDFCoefficients pidfModified = motorControllerEx.getPIDFCoefficients(motorIndex, DcMotor.
↪RunMode.RUN_USING_ENCODER);

        // Display info to user.
        while(opModeIsActive()) {
            telemetry.addData("Runtime (sec)", "%.01f", getRuntime());
            telemetry.addData("P,I,D,F (orig)", "%.04f, %.04f, %.04f, %.04f",
                    pidfOrig.p, pidfOrig.i, pidfOrig.d, pidfOrig.f);
            telemetry.addData("P,I,D,F (modified)", "%.04f, %.04f, %.04f, %.04f",
                    pidfModified.p, pidfModified.i, pidfModified.d, pidfModified.f);
            telemetry.update();
        }
    }
}
```

Note: As of SDK 7.0, the former PID-only methods are still available, but deprecated.

21.7.3 Automatically Loading a Driver Controlled Op Mode

A FIRST Tech Challenge match consists of a 30 second autonomous period followed by a 2 minute driver controlled (i.e., tele-operated or teleop) period. Previously, teams had to manually select their teleop op mode after the autonomous portion their match was over.

Teams can now preselect their teleop op mode, and have the Driver Station automatically load this op mode as soon as their autonomous run has completed. This feature can help a team avoid selecting the wrong op mode during a match.

To use this feature, verify that you are using version 6.1 or greater of the SDK software (Robot Controller and Driver Station).

Select an autonomous program to use during your match. The preselect button will appear in the lower left corner of the screen. It will be translucent and have no text adjacent to it, indicating the feature is inactive.

Note that in order for the preselect button to be visible, the selected op mode must be designated as an autonomous op mode either by using the _@Autonomous_ annotation if it is written using Java or by selecting the *Autonomous* option in the Blocks editor. If you do not see the preselect button, verify that your currently selected op mode has been designated as autonomous.

To activate it, simply tap the (translucent) button and select an op mode. The button will then become fully opaque and the name of the preselected op mode will appear adjacent to the button. This indicates the feature is active.

Should you then wish to disable it, simply long press the preselect button. It will become translucent again and the text adjacent to it will disappear.

After the Autonomous program ends, the Driver Station changes the queued OpMode to the TeleOp program which was preselected before the start of Autonomous. The auto-preselection will be aborted if the user presses stop (either the main

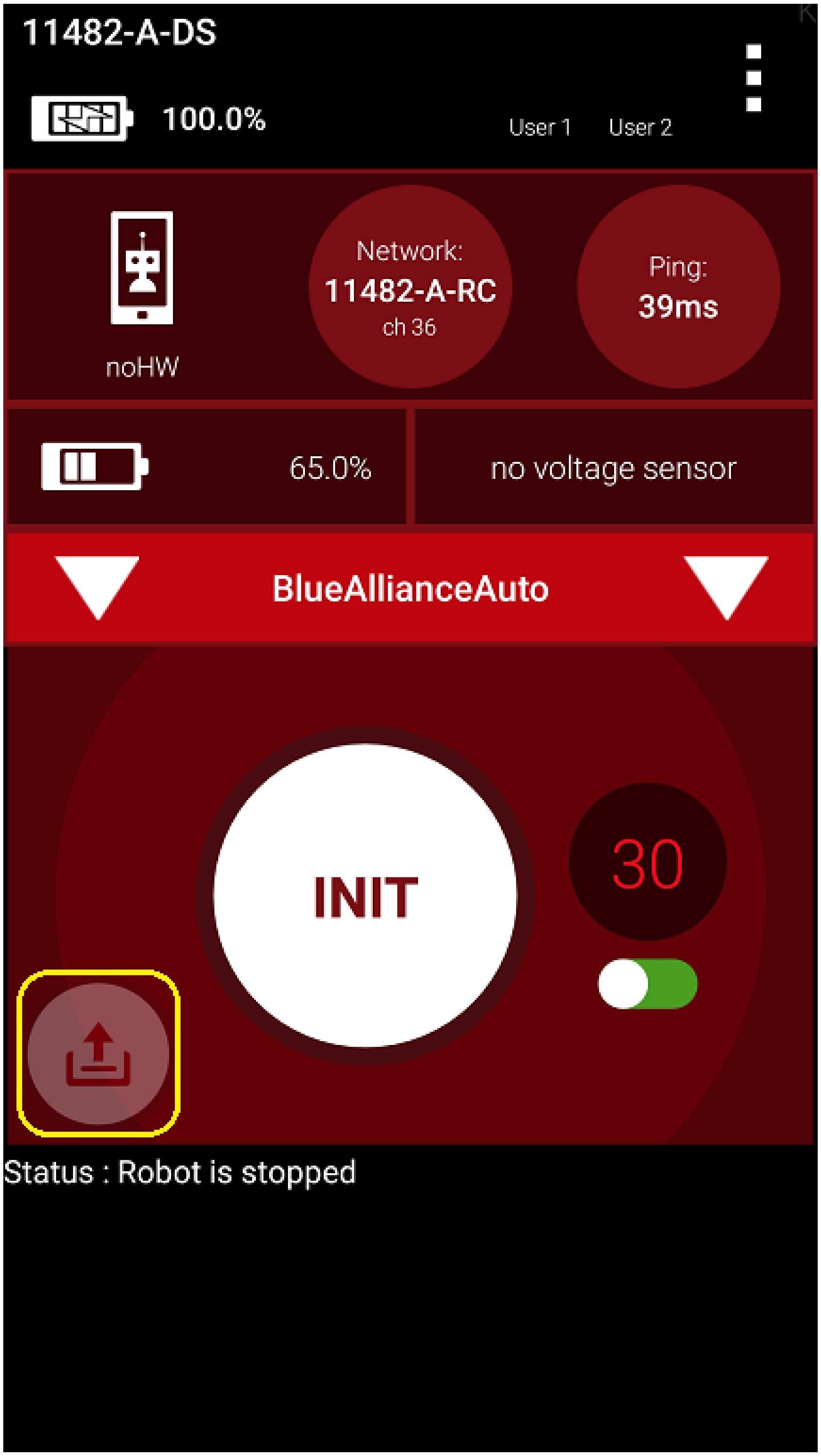


Fig. 231: The preselect button will appear once an autonomous op mode has been selected.

Fig. 232: The selected op mode must be designated as Autonomous in order for the preselect button to be visible.

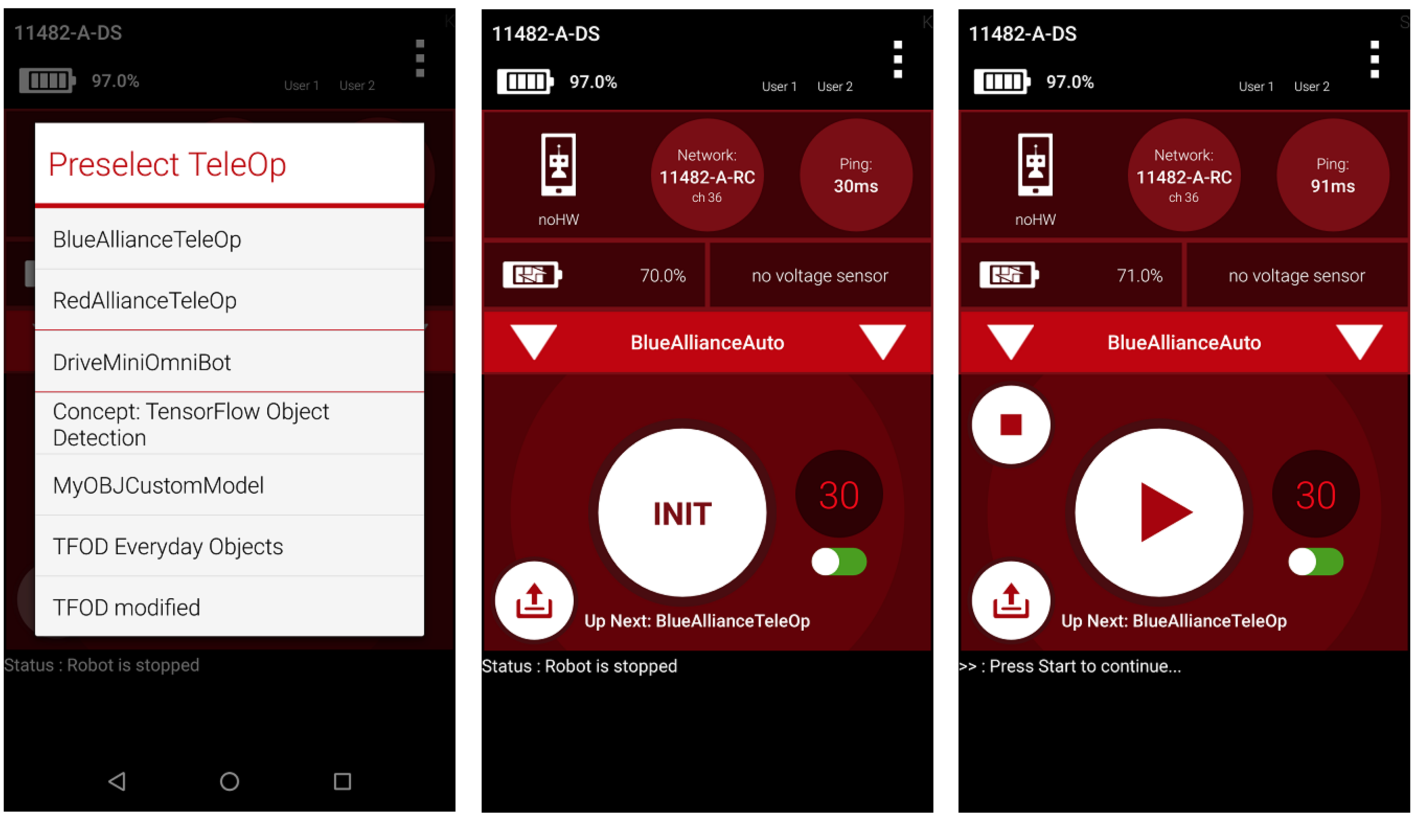


Fig. 233: The driver controlled op mode to be auto-loaded.

stop or init stop buttons). It will only transition if the OpMode either self-exits, or is terminated by the 30s timer. Drive Teams must still press Init and start the op mode manually, for safety reasons.

Should you wish to not be required to manually enable and configure the preselection feature each time you want to run your Autonomous program, you can edit your OpMode annotation to include `preselectTeleOp="My TeleOp Name"`. The Driver Station will then automatically activate the preselection feature and configure it to preselect the OpMode specified in the annotation.

Listing 1: Use the preselectTeleOp parameter to specify a preselected op mode.

```
@Autonomous(name="Blue Alliance Auto", group="Pushbot", preselectTeleOp="BlueAllianceTeleOp")
```

Blocks users can make use of this feature as well, through a new dropdown in the Blocks program editor.

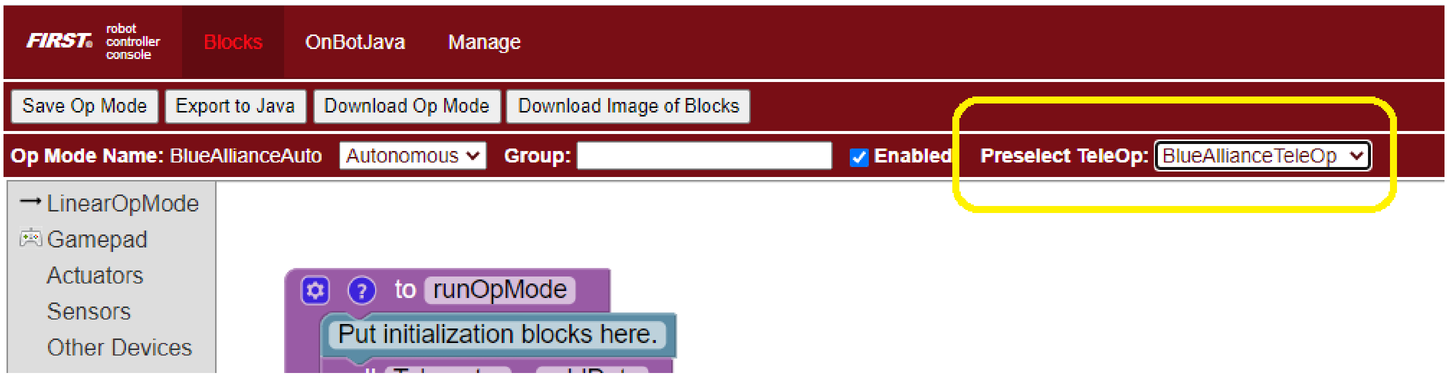


Fig. 234: You can preselect a teleop op mode mode using the Blocks editor.

Note that there is an option in the Settings menu of the Driver Station app called "OpMode Auto Queue". If this option is enabled, then the Driver Station will automatically load an autonomous op mode's preselected teleop op mode as designated by the `preselectTeleOp` parameter. If this option is disabled, then the Driver Station will not automatically load the preselected teleop op mode. If the "Op Mode Auto Queue" option is disabled, a team can still select a teleop op mode by using the preselect button on the main Driver Station activity.

21.7.4 Custom Blocks (myBlocks)

Introduction

This tutorial shows how to make **custom Blocks**, to be used in regular Blocks programs. These **"myBlocks"** are programmed in Java, with OnBot Java or Android Studio.

A myBlock can add **advanced capability** previously available only to teams using all-Java code. Or, a single myBlock can serve as a **'super-Function'**, containing robot instructions that previously needed many regular Blocks. Now your team's Blocks code can be more powerful, and simpler!

Also, myBlocks programming allows some team members to begin learning and using Java, contributing valuable new features. The other team members can continue learning and working in Blocks, producing the team's official code. Nobody is held back, or left behind.

Hats off to Google engineer Liz Looney for this major development!

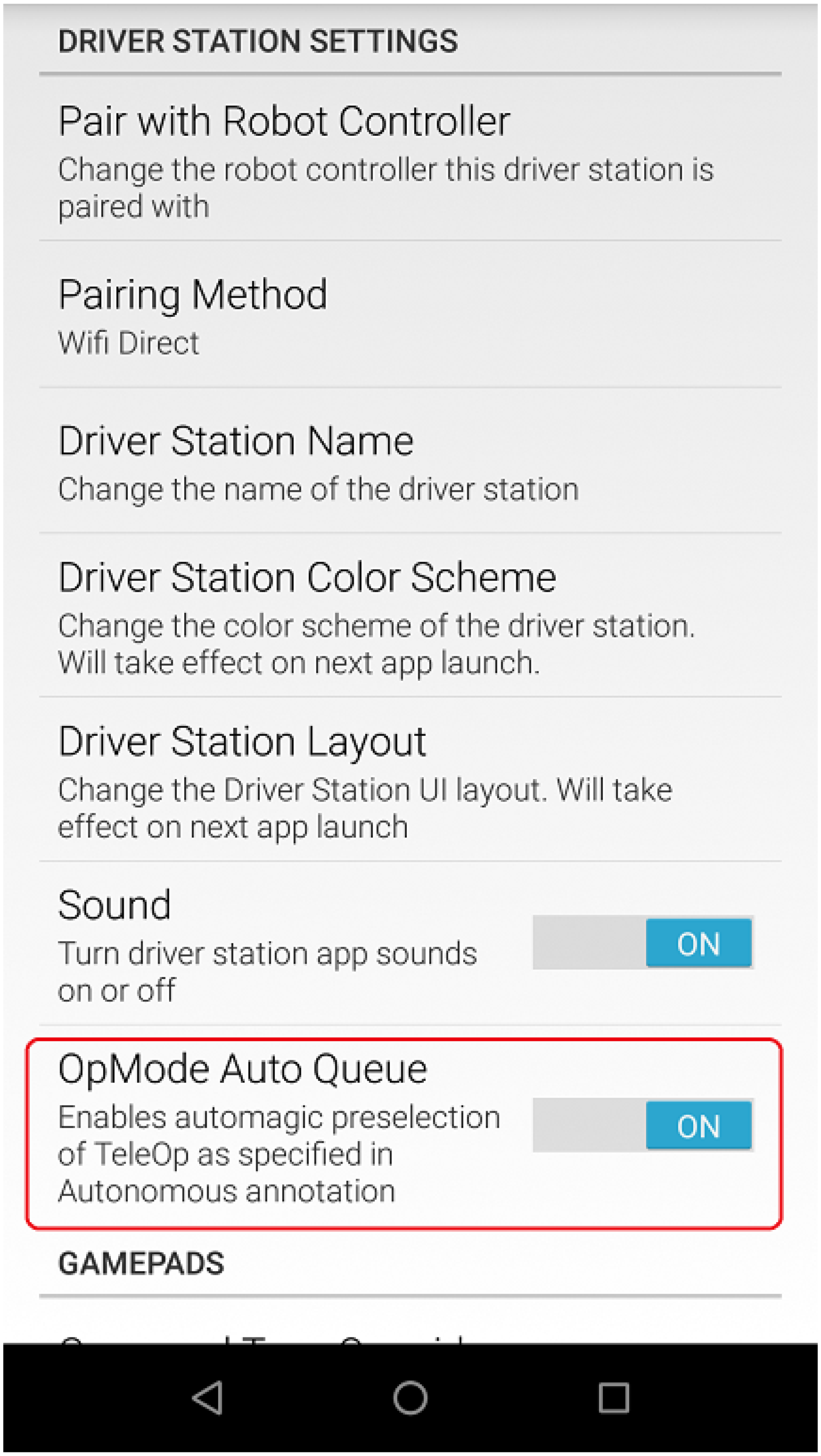


Fig. 235: If the OpMode Auto Queue option is enabled, the Driver Station will automatically load the preselectTeleOp op mode.

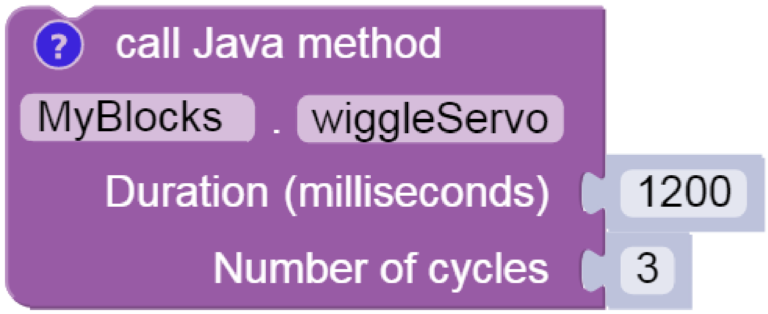


Fig. 236: sample myBlock: operate a servo, no value returned

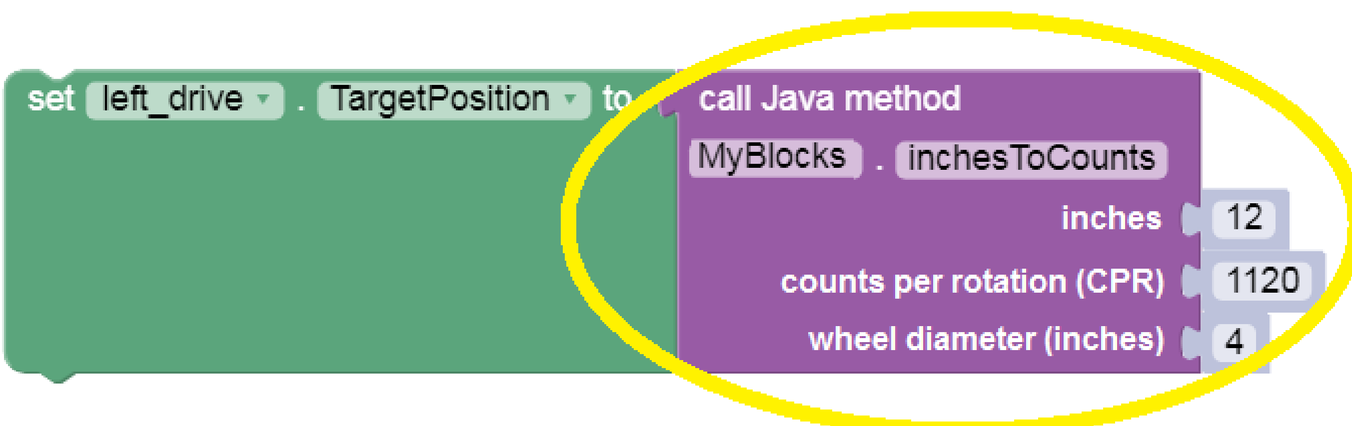


Fig. 237: sample myBlock: return encoder target value based on inputs

Notes on Java

- This tutorial builds myBlocks with *OnBot Java*, a programming tool running on the Control Hub or Robot Controller (RC) phone. Students already using *Android Studio* can easily follow the same programming.
- This tutorial does not teach Java or OnBot Java (OBJ), beyond the bare minimum needed for basic myBlocks.

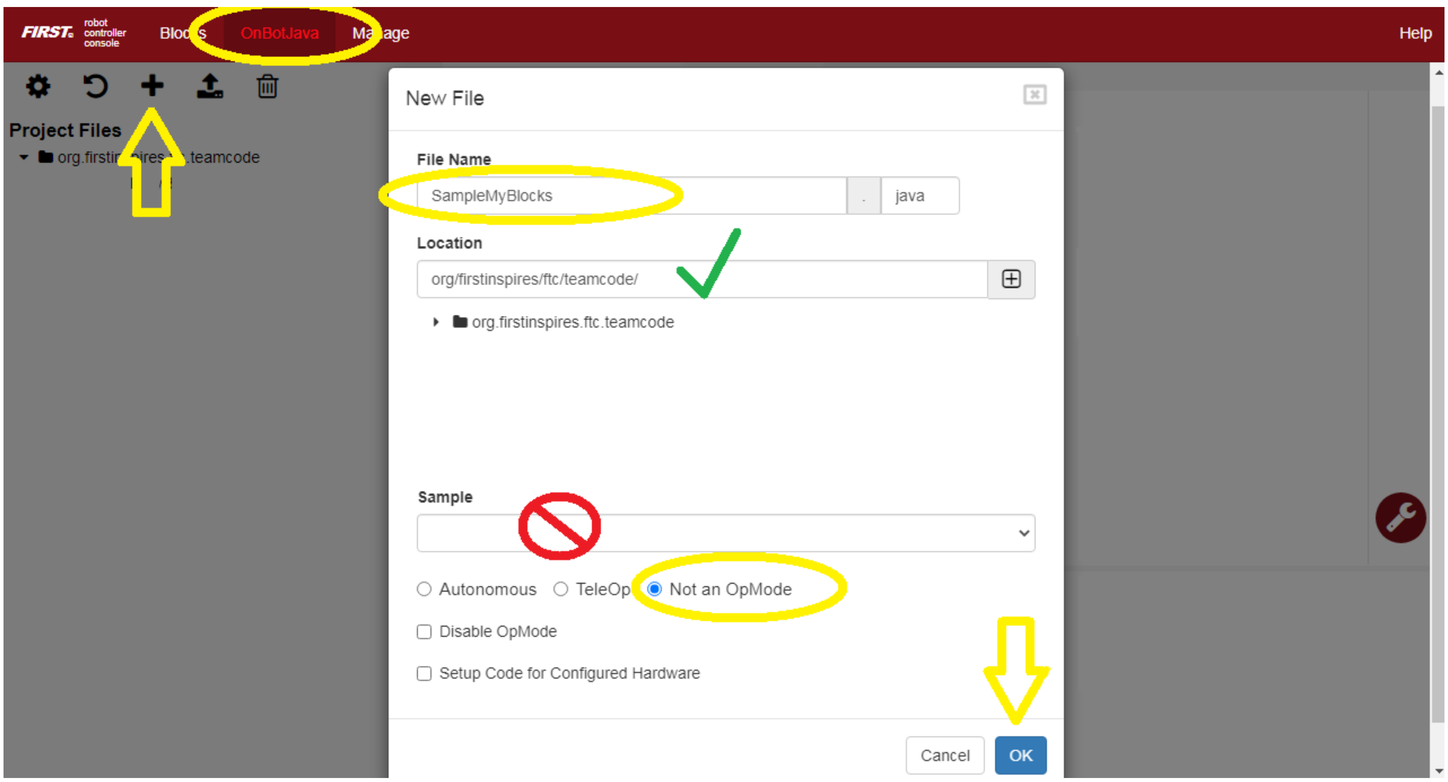
Simple Example: create myGreeting

Start with a simple myBlock that creates a greeting “Hello World” (of course!).

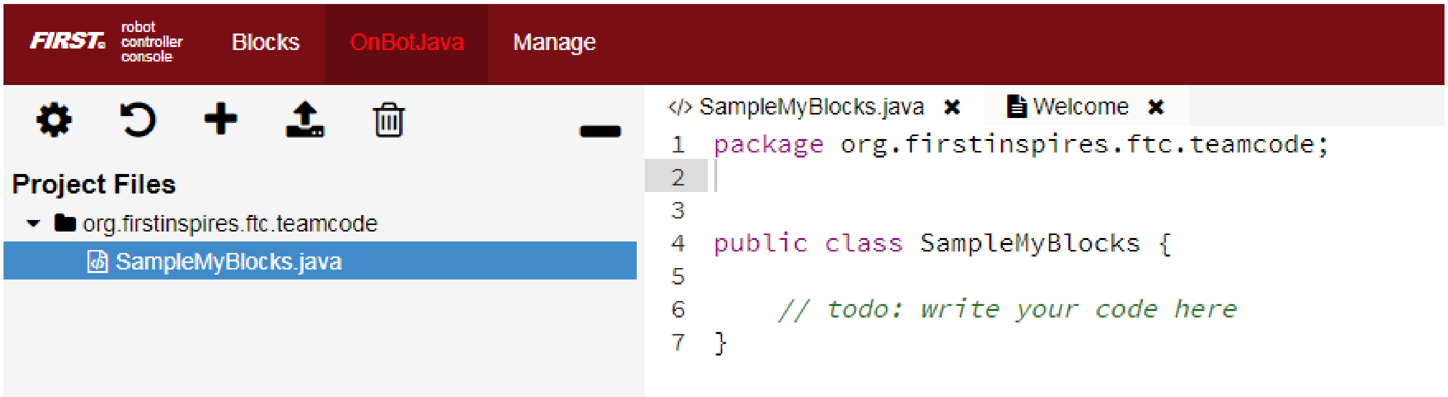
Open a Chrome browser connected via Wi-Fi to a Control Hub or RC phone. Go to the address **http://192.168.43.1:8080** (CH) or **http://192.168.49.1:8080** (RC), and click the **OnBot Java** tab.

Note: A computer can usually connect to only one Wi-Fi network at a time. To follow this tutorial while programming please use the PDF version of FTC Docs. If you need internet and programming together, connect an Ethernet cable to an internet router **or** try adding a USB Wi-Fi dongle.

Click the large **plus-sign icon** to open a new file; call it **SampleMyBlocks.java**. Use the default ‘teamcode’ folder location. Don’t choose a Sample OpMode, and use the default setting ‘Not an OpMode’. Click OK.



In the work area you see a simple/empty Java program.



Line 1 shows the default storage folder 'teamcode', and Line 4 shows the **class name**, same as the filename. It's `public` so other classes can access it. Notice the **left curly brace** at Line 4 and **right curly brace** at Line 7. Place all your code between these curly braces.

The two forward-slash marks // indicate a **comment line**, all ignored by the Java software. Good programmers use lots of comments, to communicate with your teammates and with **your future self**! You will not remember every little detail of your programs… and will thank yourself later for commenting heavily!

> Programming note: A **class** describes **methods** (actions) and **fields** (properties) that can be used by **objects** (examples or **instances** of the class). A class called 'dogs' might contain methods 'run' and 'sleep', and fields 'friendliness' and 'appetite'. Your pets Spot and Rover would be objects or instances of the 'dogs' class.

After the class name, type `extends BlocksOpModeCompanion`. This declares your new class as a **subclass** or **child** of a higher **superclass** or **parent**. The parent class BlocksOpModeCompanion contains useful tools to be **inherited** by your new subclass.

When you enter that line, the OBJ software **automatically** creates an `import` statement, making the parent class available. Convenient!

> Programming note: classes inherited from BlocksOpModeCompanion include OpMode, LinearOpMode, Telemetry, HardwareMap, and Gamepad. All very useful! Your myBlock method can directly use **objects** or **instances** of these classes without declaring them. Examples follow below.

Inside the curly braces, type new lines as follows:

```
@ExportToBlocks (
        comment = "Here is a greeting for you.",
        tooltip = "Greet a person or group.",
        parameterLabels = {"Recipient"}
)
```

These are optional labels to appear on your new myBlock; you'll see below. Even if you don't want to use any of these features, you still need the **annotation** line `@ExportToBlocks`.

When you typed that annotation, OBJ automatically added the `import` statement.

Now you're ready to create the method, namely your first myBlock. Type the following lines:

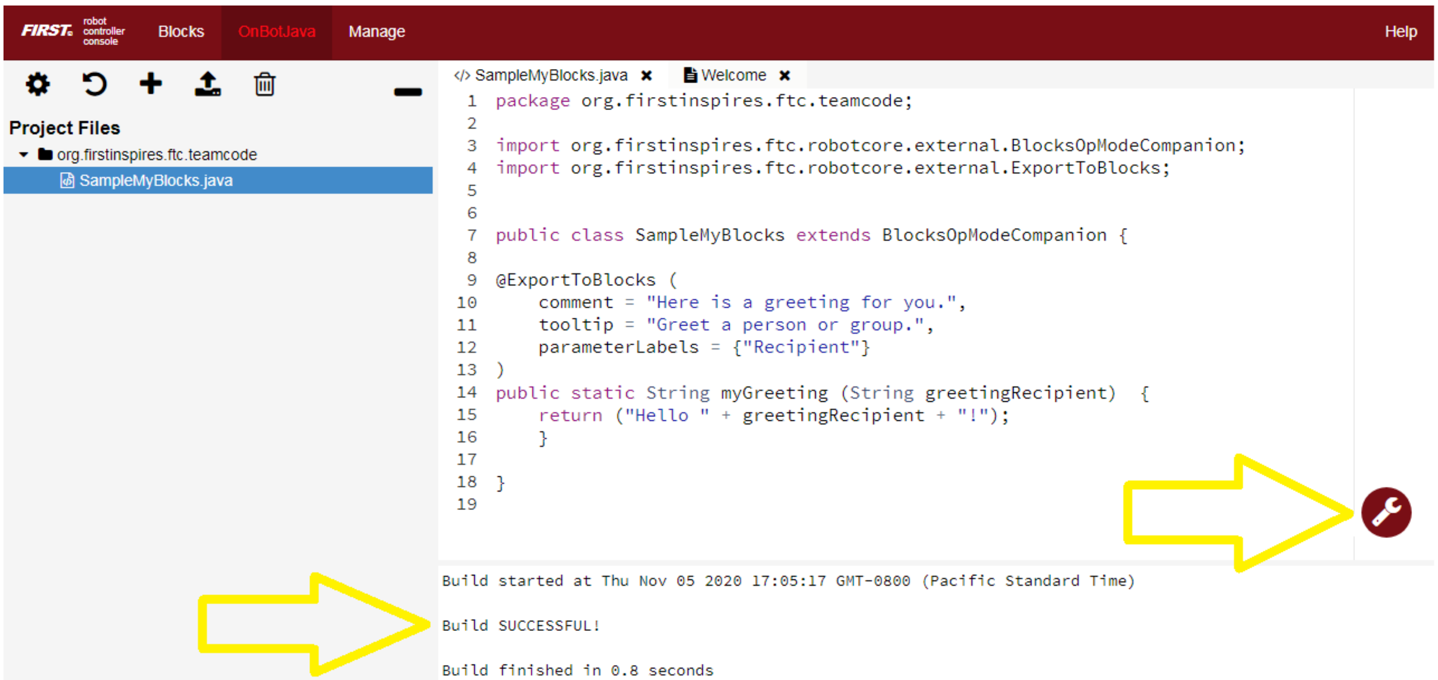
```
public static String myGreeting (String greetingRecipient)  {
        return ("Hello " + greetingRecipient + "!");
}
```

The method's name is `myGreeting`. It is a `public` method, so it can be used or **called** from other classes. And it's a `static` method, required for all myBlock methods.

The first usage of the word `String` indicates the method gives or **returns** one **output** of type String or text. The second usage is inside the parentheses, indicating the method takes one **input** named `greetingRecipient`, also of type String.

> Programming note: the method's name and list of parameters (inside the parentheses) is together called the **method signature**.

The method contains only one line of instruction, on Line 15: **three text items are joined to form a single text string**. The middle text item is the input parameter greetingRecipient, to be entered by the Blocks user. The longer combined string is returned to the program that called this method. Namely, the combined string is provided to the Block that uses your new myBlock.



That's it for the Java! Click the wrench icon to **Build Everything** including your new class. If there are error messages, read carefully and fix any mistakes. When you see "Build Successful!", your new myBlock is ready to use.

Example Code

`SampleMyBlocks_v00.java`

```
package org.firstinspires.ftc.teamcode;

import org.firstinspires.ftc.robotcore.external.BlocksOpModeCompanion;
import org.firstinspires.ftc.robotcore.external.ExportToBlocks;


public class SampleMyBlocks_v00 extends BlocksOpModeCompanion {

@ExportToBlocks (
    comment = "Here is a greeting for you.",
```

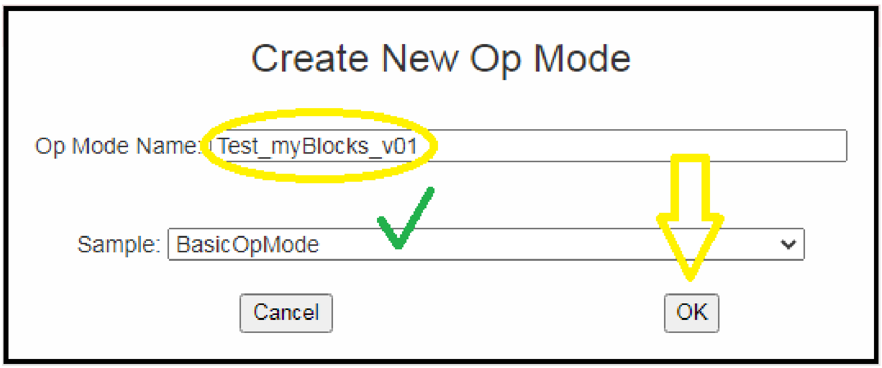
(continues on next page)

(continued from previous page)

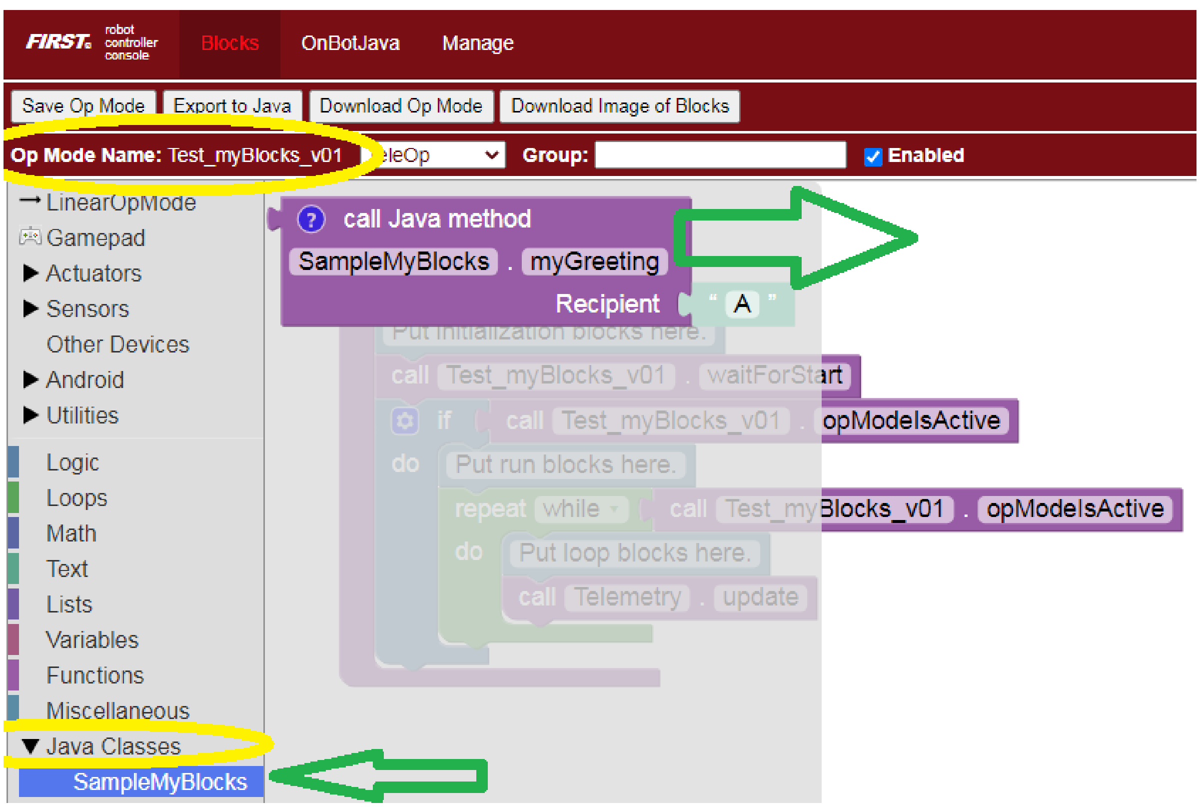
```
    tooltip = "Greet a person or group.",
    parameterLabels = {"Recipient"}
)
public static String myGreeting (String greetingRecipient)  {
    return ("Hello " + greetingRecipient + "!");
}

}
```

Simple Example: run myGreeting

In the browser still connected to the RC phone or Control Hub, - click the **Blocks** tab - click **Create New OpMode**, name it **Test_myBlocks_v01** - use the default Sample, called **BasicOpMode** - click **OK**



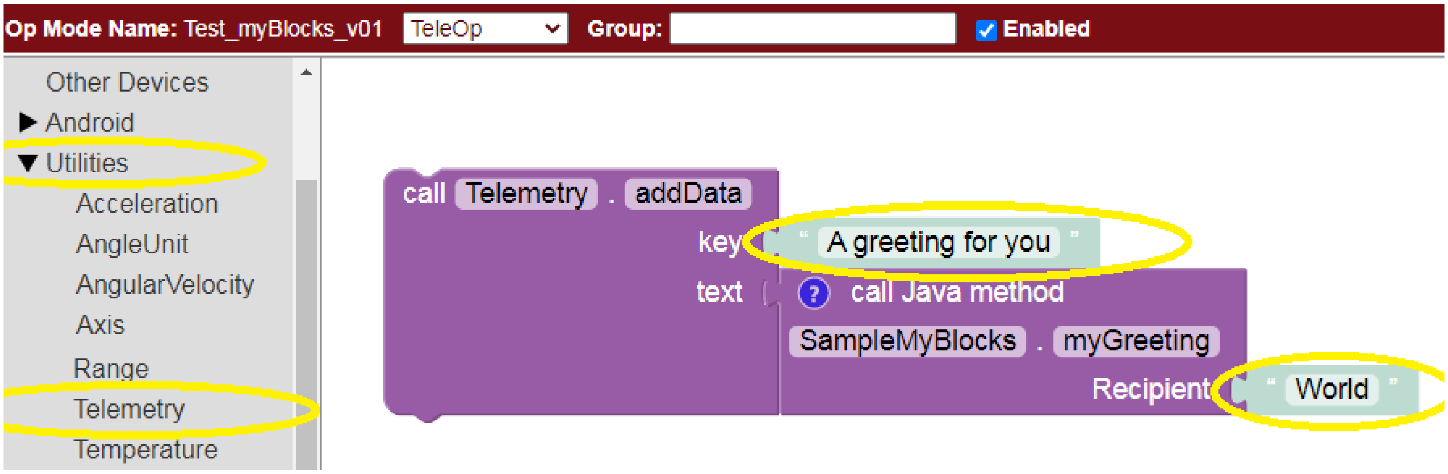
You will now see a new menu choice at the bottom, called **Java Classes**. Open that, to see the class you created, called SampleMyBlocks. Click that, and drag your new myBlock out to the work area.



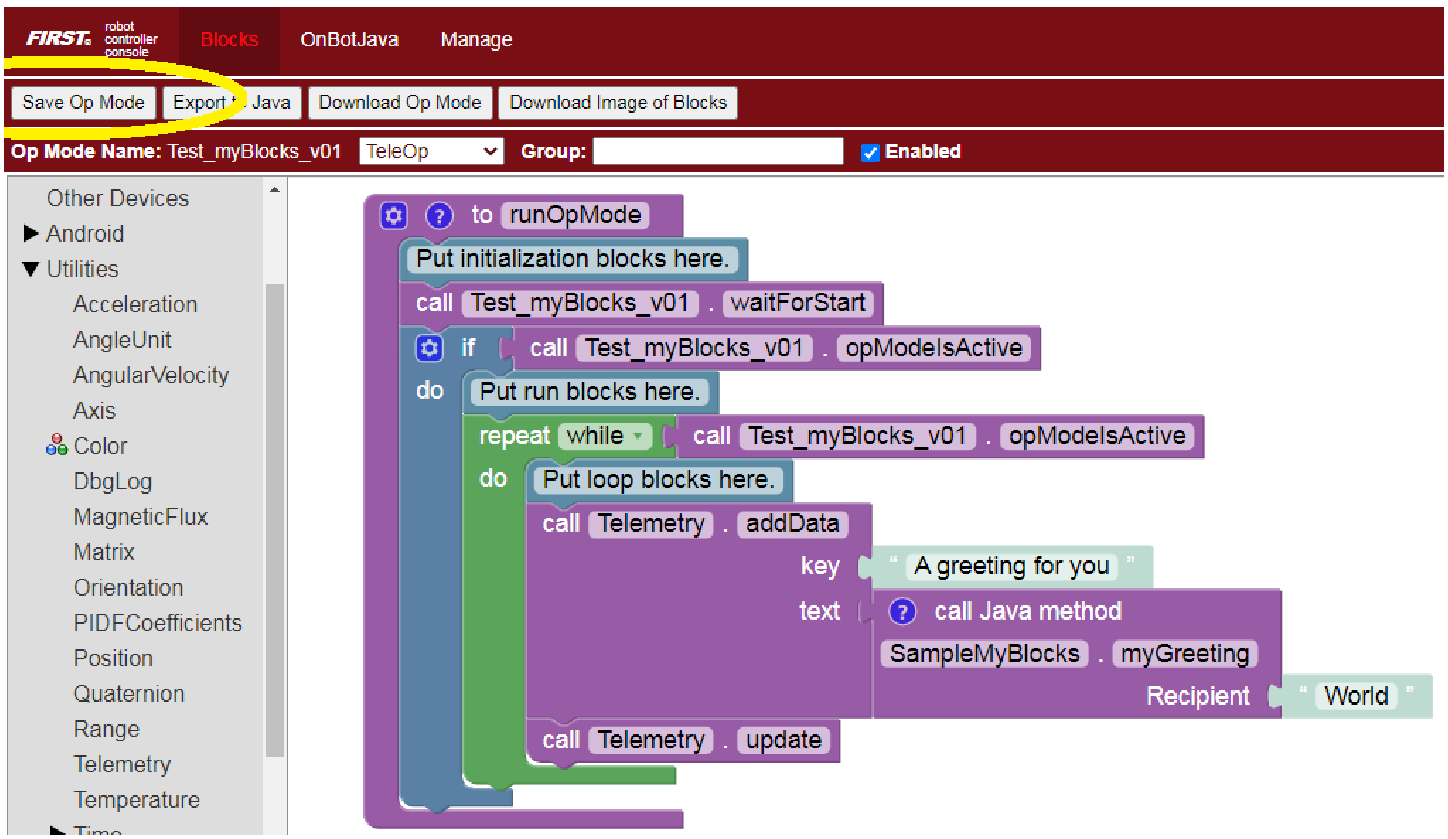
This myBlock has one grey input field or **socket**, containing the letter A to indicate a String or text input. Type the greeting recipient, **World**.

To display the myBlock's String or text output, look under **Utilities** for the **Telemetry** menu. Drag out the **Telemetry.addData** Block that **displays text** (not numbers).

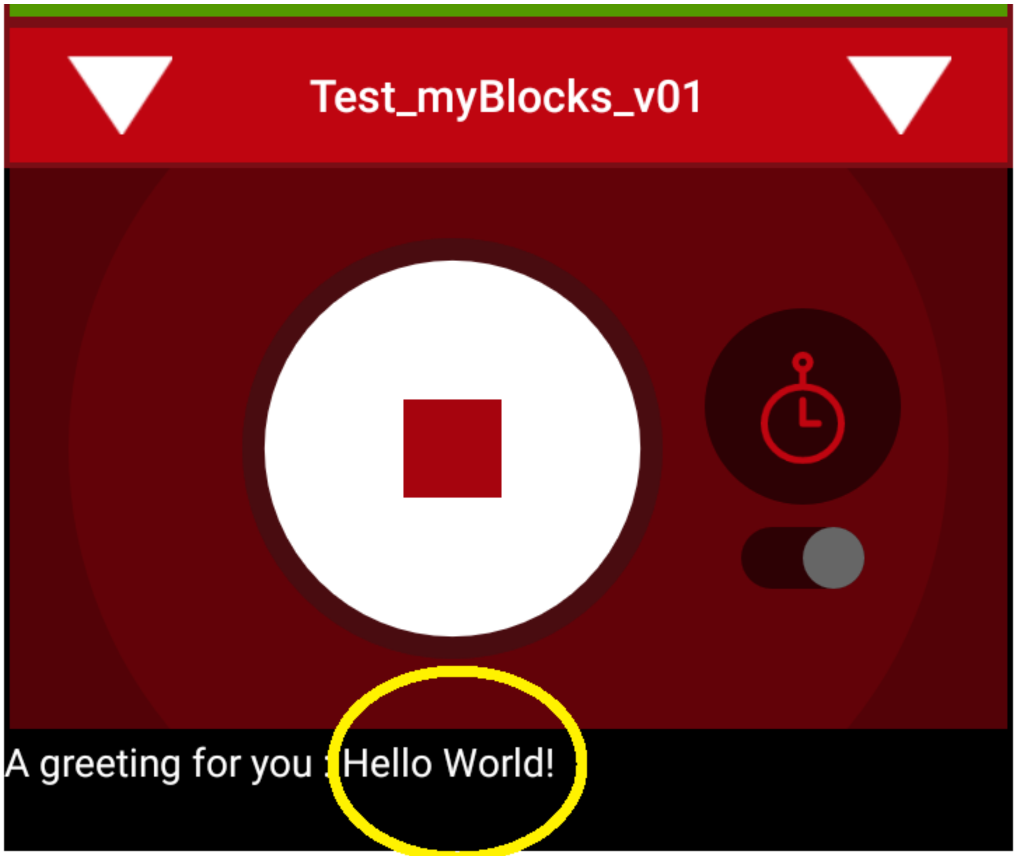
In the `key` socket, type `A greeting for you`. At the `text` socket, drag and connect your new myBlock. The myBlock's **text output** will be read and displayed by the **text** version of the Telemetry.addData Block.



Place these Blocks in the **repeat while** (loop) section of your OpMode, before **Telemetry.update**. Click **Save OpMode**.



On a connected Driver Station device, select this OpMode called Test_myBlocks_v01, touch **INIT** and the **Start Arrow**. Look at the Driver Station (DS) screen to see the traditional greeting for new programmers.



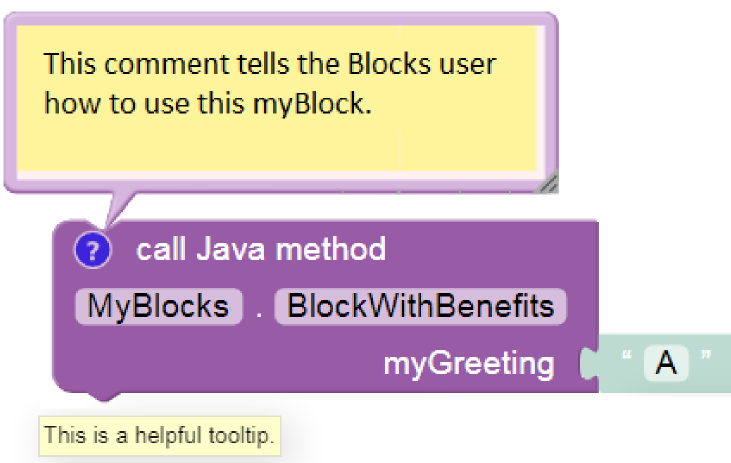
Congratulations! You are now an OnBot Java programmer and myBlocks creator.

> For extra fun: try the **Telemetry.speak** Block, followed by a 1500 millisecond `.sleep` Block. You can learn more about DS spoken telemetry at this separate tutorial.

This tutorial has three more sections with myBlocks guidelines, followed by **six examples** for you to re-type in OnBot Java and test in Blocks. Enjoy!

Annotation Details

The required **annotation** `@ExportToBlocks` has optional fields, which may be listed in any order. These fields allow a myBlock to have a custom **comment**, **tooltip**, and **parameter labels**.



Comment

- The **comment** text appears in a balloon when the Blocks user clicks the blue question-mark icon. Tell the user **how to use your myBlock**.
- Must be entered on a **single line**, with no 'line breaks'. This requirement can be met by **joining text strings**; an example is *here*.
- The blue icon will appear only if a custom comment is specified. The Blocks user can add and remove the blue icon, and can edit its text in the (re-sizeable) balloon.

Tooltip

- A **tooltip** appears with a **mouseover**: hovering the mouse cursor over an image or icon. Every Block has a short tooltip to **indicate its purpose**.
- Must be entered on a **single line**, with no line breaks.
- If a custom tooltip is not specified, the default tooltip will name the method, its enclosing class, and return type.
- Another tooltip, for the grey input socket (at right), is auto-generated based on parameter type.

Parameter Labels

- The **parameterLabels** text appears on the myBlock, each next to its grey input **socket**.
- Multiple labels are separated by a comma. Line breaks may be used between labels.
- For a single parameter, this also works: `parameterLabels = "sampleParameter"`.

In the Hello World example, you may have noticed that the parameter label **Recipient** was not the same as the Java input parameter name **greetingRecipient**. They don't need to be the same. One is for the Blocks user, the other is for the Java programmer. Just list them in the correct/same order, so their meanings correspond.

In fact you don't need to label every input; a default label will instead show the declared type (e.g. String, boolean, int, double, etc.). In any case each grey socket will contain a sample of the required type (e.g. A, false, 0), with an appropriate tooltip.

If the number of parameter labels does not match the actual number of Java parameters, **all** custom label will be ignored. Instead the default labels will be displayed.

A myBlock may have up to 21 parameters… not recommended! Keep things simple.

Again, the annotation `@ExportToBlocks` **must** appear immediately before each myBlock method, even if not using the optional fields.

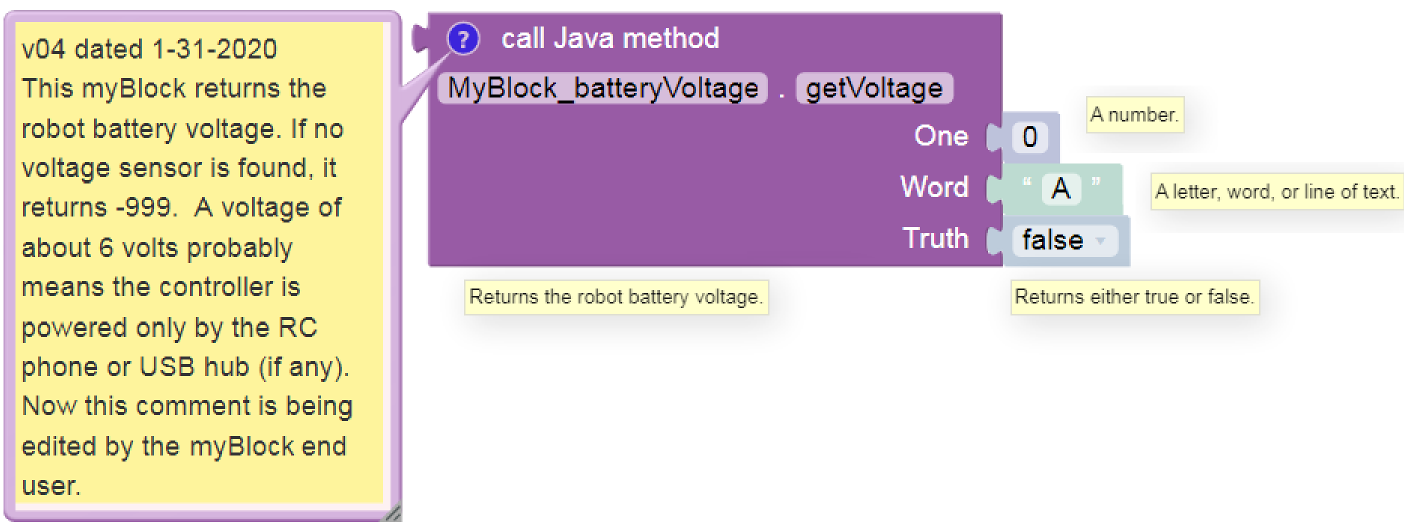
Two more optional annotation labels, not illustrated here, are:

- `heading`, such as "My Amazing myBlock". The default heading is "call Java method".
- `color`, without quotes, just a color number (hue). For example 155 is green, 255 is blue. Default is 289. Check it out!

More about Parameter Types

Do not type or run the following myBlock example. Its dummy inputs simply illustrate various **parameter types**. This myBlock does correctly read the robot battery voltage, but Blocks now offers a **VoltageSensor** Block in the **Sensors** menu.

```
public class MyBlock_batteryVoltage extends BlocksOpModeCompanion {

    static double batteryVoltage = -999;

    @ExportToBlocks (
    comment = "v04 dated 1-31-2020  This myBlock returns the robot battery
    voltage. If no voltage sensor is found, it returns -999.  A voltage of
    about 6 volts probably means the controller is powered only by the RC phone
    or USB hub (if any).",
    tooltip = "Returns the robot battery voltage.",
    parameterLabels = {"One", "Word", "Truth"}
    )
    public static double getVoltage (double uno, String parola, boolean verita) {

        List <VoltageSensor> voltageSensors;
        voltageSensors = hardwareMap.getAll(VoltageSensor.class);

        if (voltageSensors.size() > 0) {
            VoltageSensor myVoltSensor = voltageSensors.get(0);
            batteryVoltage = myVoltSensor.getVoltage();
            }        // end IF

        return batteryVoltage;

    }   // end method getVoltage()
}       // end class
```



Notice that the Java **parameters** `uno`, `parola` and `verita` have **myBlock labels** `One`, `Word` and `Truth`. They are allowed to be different.

The **comment** field explains this myBlock to the Blocks user, who can edit or delete the comment. Only for display here, this sample text appears on multiple lines; normally it must be typed as a single line of text or as joined quotes (example *here*).

A myBlock **tooltip** should be brief. Note: the four tooltips don't all appear at the same time; each appears with a mouseover. One is custom, three are auto-generated based on input type.

Each input socket shows a default value of its parameter type, with a corresponding tooltip. As shown in the method signature, parameter uno is Java type `double` (a number), `parola` is type `String` (text), and `verita` is type `boolean` (true or false).

> Programming tip: unlike primitive types, Strings must be compared with `Object.equals()` rather than ==. That's because a **text** parameter is actually an object or instance of the String class, which has its own methods equivalent to basic Java operators like ==, >, <, etc.

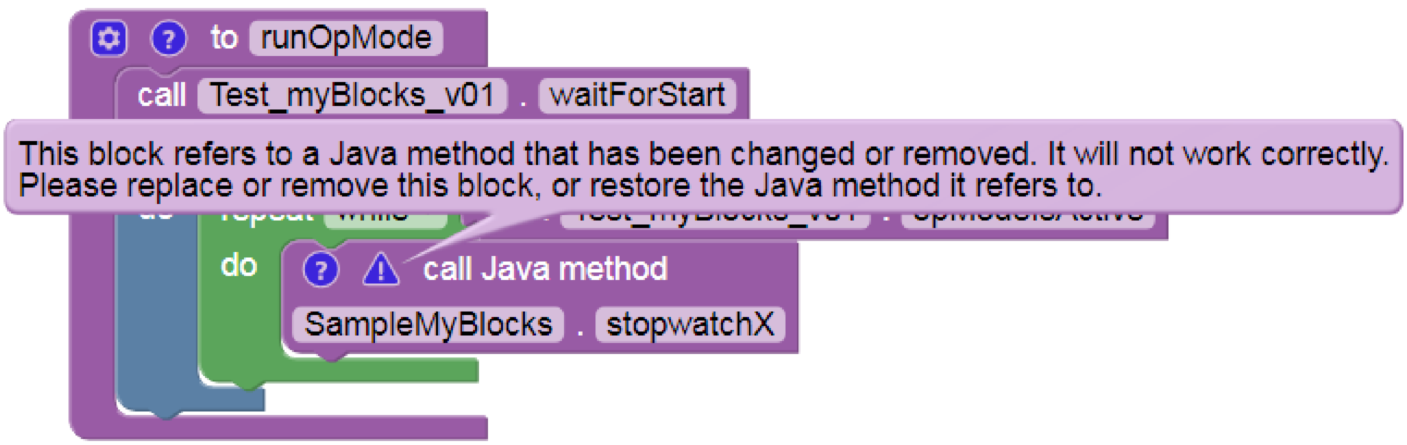
> Programming tip: In this example, the variable `batteryVoltage` is declared and initialized **outside** the method, and thus could be used by other methods in this class.

Some final notes about **parameter types**: - If your myBlock method uses a parameter declared as type `boolean` or `java.lang.Boolean`, the myBlock's input socket will accept any Block that returns (supplies) a Boolean value. - For method parameters declared as `float`, `java.lang.Float`, `double`, or `java.lang.Double`, the myBlock will accept any input Block that returns a number. - For method parameters declared as `byte`, `java.lang.Byte`, `short`, `java.lang.Short`, `int`, `java.lang.Integer`, `long`, or `java.lang.Long`, the myBlock will accept any input Block that returns a number and will round that value to the nearest whole number. - If your myBlock method uses a parameter with only **one text character**, you may use (instead of type `String`) type `char` or `java.lang.Character`. In that case, the myBlock's input socket will accept any Block that returns text and will use **only the first character** in the text string.

Editing a myBlock

If you edit and re-Build a myBlock's Java code, you might need to **replace** that myBlock in the Blocks OpMode. It depends on whether you change the myBlock's visible or external features: annotation fields, input parameters or returned outputs.

If your Java change does affect external features, its updated myBlock is available only from the Java Classes menu in Blocks. Any such myBlock **already placed in an OpMode** is obsolete and may generate a **Blocks warning**; replace it with the new myBlock. In some cases you may need to re-open the OpMode from the top-level Blocks listing.



If your edit affects only the myBlock's **internal** processing, it might update automatically after "Build Everything", without needing a fresh replacement from the Java Classes menu. In some cases you might not even need to click Save OpMode in the Blocks screen – you could simply re-run the OpMode on the Driver Station with INIT and Start. This can allow very fast testing of minor/internal changes to the myBlock.

In any case, consider adding **versions** to your myBlock names, such as myGreeting_v01. Copy and paste before editing, to keep all related myBlock methods in the **same Java class**. In Blocks, all uniquely named versions will be available in the Java Classes menu, under that single class name.

Keep the class name **short and generic**, such as MyBlocks, SampleMyBlocks, Team8604MyBlocks, DrivingMyBlocks, etc. It will contain all or many related myBlocks, not just one myBlock per the simple examples above.

In that single class, each myBlock method must appear after its own annotation `@ExportToBlocks`. That class may contain other methods that are not myBlocks; omit the annotation before any non-myBlock methods. Such methods might be used to initialize variables, or might be (shared) submethods called by one or more myBlocks. An example is shown *here*.

This tutorial has covered these basic requirements so far: - create/store in **org.firstinspires.ftc.teamcode** folder/package - class **extends BlocksOpModeCompanion** - each myBlock method needs annotation **@ExportToBlocks** - method must be **public** and **static** (must not be abstract) - replace myBlocks after external edits

The rest of this tutorial gives **examples** that you can **re-type in OnBot Java** and **test in Blocks**. Try making changes and adding features!

Hardware Example: control a servo

Here's a very simple example to illustrate how a myBlock can access the **robot hardware**. Here, the Blocks user enters the servo's name as a **parameter** of the myBlock.

SampleMyBlocks.java

```java
package org.firstinspires.ftc.teamcode;

import org.firstinspires.ftc.robotcore.external.BlocksOpModeCompanion;
import org.firstinspires.ftc.robotcore.external.ExportToBlocks;
import com.qualcomm.robotcore.hardware.Servo;

public class SampleMyBlocks extends BlocksOpModeCompanion {

    @ExportToBlocks (
    comment = "Move a conventional servo back and forth. Assumes servo starts" +
              " from position 0. Servo name must be in the active configuration.",
    tooltip = "Wiggle a user-designated servo.",
    parameterLabels = {"Servo name", "Duration (milliseconds)", "Number of cycles"}
    )
    public static void wiggleServo (String servoName, int duration, int cycles) {

        Servo myServo = hardwareMap.get(Servo.class, servoName);

        // count up to 'cycles' AND while opMode was not stopped
        for (int i = 0; i < cycles && linearOpMode.opModeIsActive(); i++)  {

            myServo.setPosition(0.5);                  // move servo clockwise
            linearOpMode.sleep(duration);              // wait for 'duration'
            myServo.setPosition(0);                    // move servo counterclockwise
            linearOpMode.sleep(duration);              // wait for 'duration'
        }
    }   // end method wiggleServo()
}       // end class SampleMyBlocks

```

Example Code

SampleMyBlocks_v01.java

```java
package org.firstinspires.ftc.teamcode;

import org.firstinspires.ftc.robotcore.external.BlocksOpModeCompanion;
import org.firstinspires.ftc.robotcore.external.ExportToBlocks;
import com.qualcomm.robotcore.hardware.Servo;
```

(continues on next page)

(continued from previous page)

```java
public class SampleMyBlocks_v01 extends BlocksOpModeCompanion {

    @ExportToBlocks (
    comment = "Move a conventional servo back and forth. Assumes servo starts" +
              " from position 0. Servo name must be in the active configuration.",
    tooltip = "Wiggle a user-designated servo.",
    parameterLabels = {"Servo name", "Duration (milliseconds)", "Number of cycles"}
    )
    public static void wiggleServo (String servoName, int duration, int cycles) {

        Servo myServo = hardwareMap.get(Servo.class, servoName);

        // count up to 'cycles' AND while opMode was not stopped
        for (int i = 0; i < cycles && linearOpMode.opModeIsActive(); i++)  {

            mySeervo.setPosition(0.5);              // move servo clockwise
            linearOpMode.sleep(duration);          // wait for 'duration'
            myServo.setPosition(0);                // move servo counterclockwise
            linearOpMode.sleep(duration);          // wait for 'duration'
        }
    }   // end method wiggleServo()
}       // end class SampleMyBlocks_v01
```

Lines 10-11 contain two strings of text (each in quotes), joined with a **"+"** character to form a **single text string**. This is an alternate way to meet the requirement that a comment field must be a **single line** of text, with no 'line break'. Shorter strings allow all the text to be visible on-screen, without scrolling sideways.

Line 15: this method has 3 inputs and no outputs (keyword **void**).

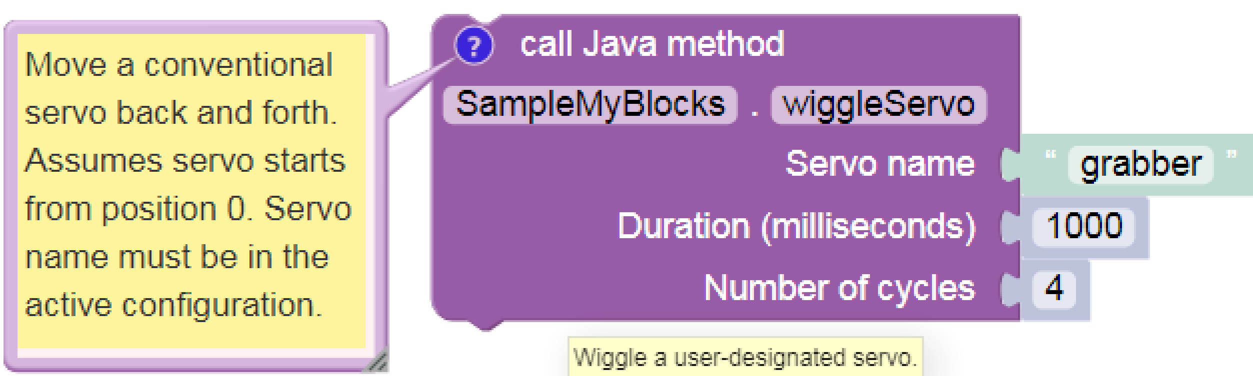
Line 17 shows how to access **hardwareMap**, the configured devices list provided from BlocksOpModeCompanion. That single line of Java does this: - declare a new variable called myServo, of type (class) Servo - **get** the properties (methods and variables) of the named servo from hardwareMap - assign those properties to the new variable myServo

Line 20 is a **for loop**, which you can learn about here or here. It runs the specified servo back and forth, using the specified duration and number of cycles. This **for loop** has the added condition `opModeIsActive()`, to monitor and verify the OpMode has not been stopped.

Lines 22 and 24: the object myServo uses a method `setPosition()` from the Servo class.

Lines 23 and 25: the object linearOpMode uses a method `sleep()` from the class inherited from BlocksOpModeCompanion.

The Blocks user must enter the exact device name from the **active configuration**. Hardware device names (motors, servos, sensors) are found in the Configure Robot menu of the RC app or paired DS app. Or, it might be easier to retype the name from any Blocks drop-down list containing those device types. For example, a green Servo `set .Position` Block will display all configured servo names – make sure the correct configuration was made active **before** entering the Blocks session.



As an alternate, you could 'hard-code' the servo's name directly into the Java method, instead of the Blocks user entering the servo name as a parameter.

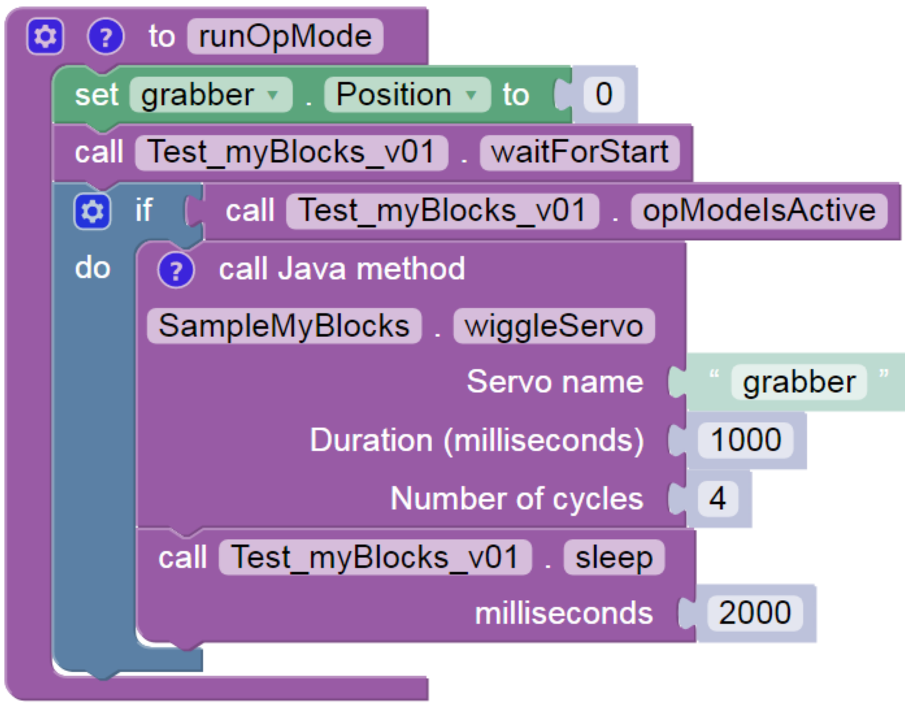
PROs of hard-coding: - myBlock is simpler - Blocks user doesn't need to know or enter the servo name

CONs of hard-coding: - you need to know the exact servo name in advance - if the name ever changes, your myBlock cannot find the servo

---

Note: As a programmer, you will constantly face choices like this, with pros and cons. This is part of software design, a key professional skill and career path.

A **different version** (gamepad-controlled, fully commented) of the above Java program is provided below. It illustrates using 5 of the 6 objects provided by BlocksOpModeCompanion, including **telemetry** and the **gamepads**. This longer example, or the short version above, could be used in an OpMode like this:



The final `.sleep` Block allows any telemetry to remain visible on the DS screen, before this sample OpMode ends.

Different Version of Example Code

SampleMyBlocks_v02.java

```
/*
This is a sample Java program for an FTC myBlocks tutorial.  This class
contains methods that define myBlocks for FTC Blocks programming.

Demonstrates using 5 of the 6 objects inherited from BlocksOpModeCompanion:
linearOpMode, hardwareMap, telemetry, gamepad1, gamepad2.

Each of these 5 objects allows direct/convenient use of its commands (methods).
*/

// a myBlocks class must exist in the 'teamcode' folder/package
package org.firstinspires.ftc.teamcode;
```

(continues on next page)

(continued from previous page)

```java
// these are (usually!) automatically listed by OnBot Java when needed
import org.firstinspires.ftc.robotcore.external.BlocksOpModeCompanion;
import org.firstinspires.ftc.robotcore.external.ExportToBlocks;
import com.qualcomm.robotcore.hardware.Servo;

// BlocksOpModeCompanion provides 6 classes useful for myBlocks
public class SampleMyBlocks_v02 extends BlocksOpModeCompanion {

    // annotation required for method to be a myBlock; 3 features optional
    @ExportToBlocks (
    comment = "Move a conventional servo back and forth. Assumes servo starts" +
              " from position 0. Servo name must be in the active configuration.",
    tooltip = "Wiggle a user-designated servo.",
    parameterLabels = {"Servo name", "Duration (milliseconds)", "Number of cycles"}
    )
    // this is a myBlock method with 3 inputs and no outputs (void)
    public static void wiggleServo (String servoName, int duration, int cycles) {

        /*
        1. Declare new object called myServo, of type (class) Servo.
        2. Get properties of named servo from hardwareMap (configuration).
        3. Assign those properties to new object myServo.
        */
        Servo myServo = hardwareMap.get(Servo.class, servoName);

        // Display confirming messages and instructions for user.
        telemetry.addData("Servo name", servoName);
        telemetry.addData("Servo cycle duration", duration);
        telemetry.addData("Servo cycles to run", cycles);
        telemetry.addData(": : : : PRESS BUTTON X TO BEGIN : : :", null);
        telemetry.update();

        while ( !gamepad1.x && !gamepad2.x             // X buttons not pressed
                && linearOpMode.opModeIsActive() )   {
                  // empty while loop, waiting for operator input
                }

        // Wiggle the servo using specified duration and cycles,
        // and while the opMode was not stopped.
        for (int i = 0; i < cycles && linearOpMode.opModeIsActive(); i++)  {

            telemetry.addData("Servo current cycle", i+1);
            telemetry.update();             // display progress to user

            myServo.setPosition(0.5);       // move servo clockwise
            linearOpMode.sleep(duration);   // hold for duration
            myServo.setPosition(0);         // move servo counterclockwise
            linearOpMode.sleep(duration);   // hold for duration
        }

        // Display final info for user.
        telemetry.addData("Servo name", servoName);
        telemetry.addData("Servo cycle duration", duration);
        telemetry.addData("Servo cycles completed", cycles);
        telemetry.update();

    }   // end method wiggleServo()
```

(continues on next page)

(continued from previous page)

```
}        // end class SampleMyBlocks_v02
```

Driving Example

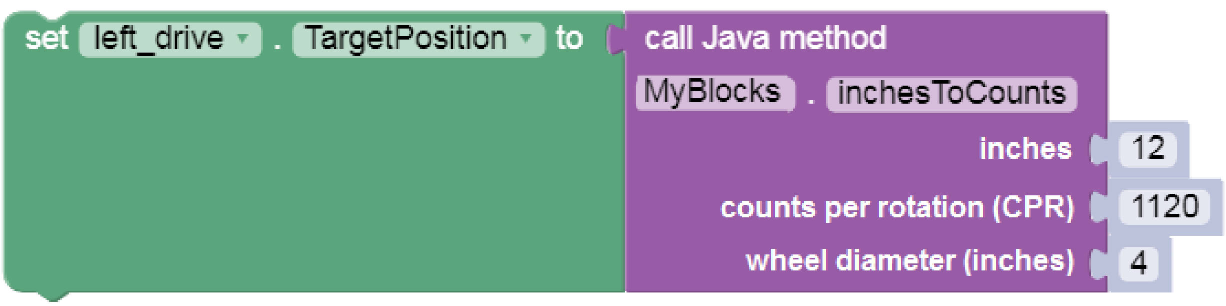
Here is the Java code (method only) for converting an **inches of driving** target into an **encoder counts** target. The conversion depends on the drive motors' counts-per-rotation (CPR), and the diameter of the drive wheels. This example assumes 1:1 gear ratio between the motor and wheel.

```
public static int inchesToCounts(int inchesToDrive,
int countsPerWheelRotation, double wheelDiameter) {
    double circumference = wheelDiameter * Math.PI;
    double rotations = inchesToDrive / circumference;
    double countsToDrive = rotations * countsPerWheelRotation;
    return (int) countsToDrive;
}
```

This method takes three inputs from the Blocks user, and **returns** one output (of type `int` or integer) to the regular Block that **calls** the myBlock.

Tip: Notice the calculation uses the variable or **constant** named PI, from the inherited class Math. This holds the fixed numeric value 3.14159….

Here is an example of typical usage.



Tip: Notice the `(int)` operator at the `return` command. This converts or **casts** the countsToDrive variable of type `double` to type `int`, to be compatible with the required **return type**. Learn more about **type casting** here or here.

As programmer, you could modify this example in many ways, such as: - handle a **gear ratio** between the drive motors and wheels - the second and third parameters could be **'hard-coded'** into the myBlock, if they will never change - those 2 variables could be initialized in a **non-myBlock method** and used by multiple myBlock methods in that same Java class

Timer Example

FTC **timers** offer much more capability than the familiar `.sleep` Block. Java programmers can learn about timers from this Blocks tutorial; you can easily apply its lessons to Java programs.

When creating myBlocks, be careful when converting or 'packaging' a section of existing Java code into a myBlock method. As a programmer, you must consider **where** your myBlock might be placed in the OpMode. For example, if the myBlock is placed inside a **repeat while loop**, the Java method will be called many times – this may or may not be what you intended. Use the annotation **comment** to tell the Blocks user how your myBlock should be run, including looping (or not).

A particular caution with timers: creating or **instantiating** a new FTC timer also starts or **resets** that timer. If a timer is created inside a myBlock that's used in a Blocks **repeat loop**, that timer will constantly reset and never advance to the intended time limit.

The following example separates the **create timer** task from the **reset timer** task.

```
13 public class SampleMyBlocks extends BlocksOpModeCompanion {
14
15     private static ElapsedTime myStopwatch = new ElapsedTime();
16
17     @ExportToBlocks (
18     comment = "Place this myBlock inside a 'repeat loop'." +
19               " Press button X to reset timer.",
20     tooltip = "Stopwatch on gamepad button X."
21     )
22     public static void stopwatchX()  {
23
24         telemetry.addData("Stopwatch timer", "%.2f", myStopwatch.time());
25         telemetry.addData("To reset stopwatch", "press button X");
26         telemetry.update();
27
28         if (gamepad1.x || gamepad2.x)  {
29             myStopwatch.reset();
30         }
31
32     }   // end of method stopwatchX()
33
34 }       // end of class SampleMyBlocks
```

Line 15: this single line of Java does all this: - declare a field called myStopwatch, of type (class) ElapsedTime - the field is **private**, can be used only in this class SampleMyBlocks - the field is **static**, can be used in static methods such as myBlocks - call the **constructor** method ElapsedTime() to **instantiate** a **new** ElapsedTime instance - assign that **instance** to the field myStopwatch

Lines 18-19 again show two strings of text (each in quotes), joined with a **"+"** character to form a **single text string**. This is an alternate way to meet the requirement that a comment field must be a **single line** of text, with no 'line break'.

Line 22: this method has **no inputs** (empty parentheses) and **no outputs** (keyword **void**). This is why the annotation @ExportToBlocks was missing the **parameterLabels** field.

In Line 24 the data is displayed using a **formatting code**, indicated by the percent sign. The **.2f** will display a numeric value with 2 digits to the right of the decimal point.

Also on Line 24, the object myStopwatch uses a method `time()` to retrieve that timer's current value in seconds.

Line 28: the double-strokes operator || means "OR". Other operators include **&&** ("AND"), **==** ("EQUALS"), and **!=** ("NOT EQUAL TO").

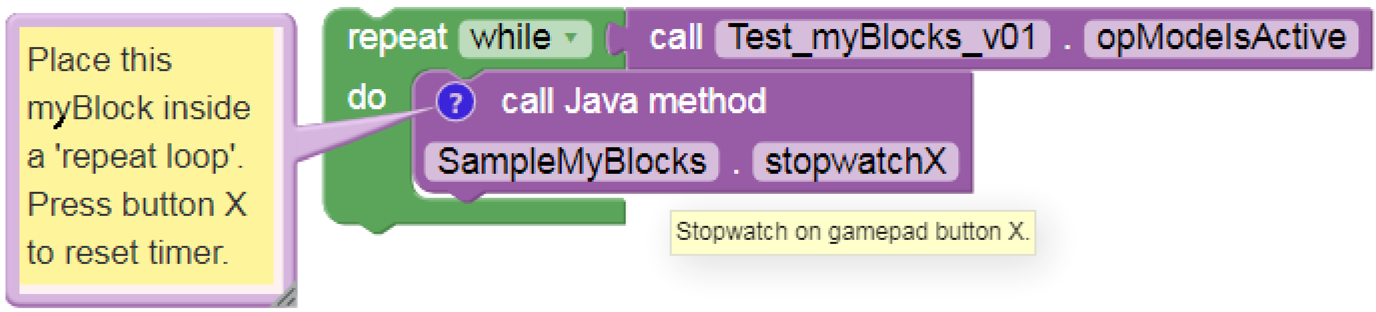
Line 29: the object myStopwatch uses a method `reset()` to start the timer again from zero.

So, what was the danger? A programmer might naturally place Line 15 **inside** the method, perhaps at Line 23. But that would reset the timer at every cycle of the **repeat while** loop. The stopwatch would always show **zero**.

Or, a programmer might use Line 15 to **replace** Line 29, since they "do the same thing". But the object **myStopwatch** is needed at Line 24 also, for telemetry. Moving the telemetry to be **after** Line 29 does not help. If the operator has not yet pressed gamepad button X, the object does not exist and the program will crash.

When you clicked "Build Everything" in OnBot Java, all of the code in your SampleMyBlocks class was processed. That included creating the object myStopwatch, which became available for any method in that class. It was not necessary to declare it inside the myBlock method. In this case, it **needed** to be outside the method.

Here's the myBlock in a repeat loop, with its **comment** and **tooltip**:



Again, the comment field is the only way to communicate with future users of your myBlock. They cannot see your Java code or its Java comments. Keep your myBlocks interface simple, and the instructions clear.

Note: This tutorial intends for you to **manually type** the Java code above. OnBot Java helps by suggesting some code as you type, and by entering import statements when classes are used. Android Studio helps even more. If you require pre-typed text of this example see below. The linked copy includes more Java comments, omitted above to focus on the Java code. Also not shown are the package and import statements.

Example Code

SampleMyBlocks_v03.java

```java
/*
This example is used in a tutorial on FTC myBlocks.
A gamepad button operates a simple timer.
The Blocks user places this myBlock in a loop.
*/

package org.firstinspires.ftc.teamcode;

import org.firstinspires.ftc.robotcore.external.BlocksOpModeCompanion;
import com.qualcomm.robotcore.util.ElapsedTime;
import org.firstinspires.ftc.robotcore.external.ExportToBlocks;

public class SampleMyBlocks_v03 extends BlocksOpModeCompanion {

    private static ElapsedTime myStopwatch = new ElapsedTime();

    @ExportToBlocks (
    comment = "Place this myBlock inside a 'repeat loop'. Press button X" +
              " to reset timer.",
    tooltip = "Stopwatch on gamepad button X."
    )
```

(continues on next page)

(continued from previous page)

```
    public static void stopwatchX()  {

        telemetry.addData("Stopwatch timer", "%.2f", myStopwatch.time());
        telemetry.addData("To reset stopwatch", "press button X");
        telemetry.update();

        if (gamepad1.x || gamepad2.x)  {
            myStopwatch.reset();
        }

    }   // end of method stopwatchX()

}       // end of class SampleMyBlocks_v03
```

Example: non-myBlock methods

Your Java class may also contain methods that are not myBlocks. Consider this if you have multiple myBlocks that perform a shared internal process or calculation. This is a good programming practice in general, not specifically related to myBlocks.

To illustrate, consider the Driving Example above. Imagine you want to create myBlocks to support **two** different robots. - Robot A has **4-inch** drive wheels with AndyMark **NeveRest 40** motors. - Robot B has **3-inch** drive wheels with NeveRest **Orbital 20** motors. - You want the myBlocks to be **very simple** for your Blocks programming teammates.

Your solution: - One MyBlock per robot. - Each Blocks user needs to specify only the distance to drive, in inches. - Each myBlock uses the appropriate wheel size and motor encoder CPR. - The myBlocks share a 'utility' method to convert distance to encoder counts.

```
public class SampleMyBlocks extends BlocksOpModeCompanion {

    @ExportToBlocks (
        comment = "FOR ROBOT A ONLY. Enter inches to drive.",
        tooltip = "Robot A convert inches to encoder counts",
        parameterLabels = "Drive Distance (inches)"
        )
    public static int inchesToCountsRobotA (double inchesToDriveA) {
        final double WHEEL_DIAMETER_A = 4.0;
        final double COUNTS_PER_ROTATION_A = 1120;
        int countsToDriveA = calculateCounts (inchesToDriveA, COUNTS_PER_ROTATION_A, WHEEL_DIAMETER_A);
        return countsToDriveA;
    }

    @ExportToBlocks (
        comment = "FOR ROBOT B ONLY. Enter inches to drive.",
        tooltip = "Robot B convert inches to encoder counts",
        parameterLabels = "Drive Distance (inches)"
        )
    public static int inchesToCountsRobotB (double inchesToDriveB) {
        final double WHEEL_DIAMETER_B = 3.0;
        final double COUNTS_PER_ROTATION_B = 537.6;
        int countsToDriveB = calculateCounts (inchesToDriveB, COUNTS_PER_ROTATION_B, WHEEL_DIAMETER_B);
        return countsToDriveB;
    }

    // This method is NOT a myBlock; it is called by other methods.
    private static int calculateCounts (double inchesToDrive, double countsPerWheelRotation, double wheelDiameter) {
        double circumference = wheelDiameter * Math.PI;
        double rotations = inchesToDrive / circumference;
        double encoderCounts = rotations * countsPerWheelRotation;
        return (int) encoderCounts;
    }

}       // end of class SampleMyBlocks
```

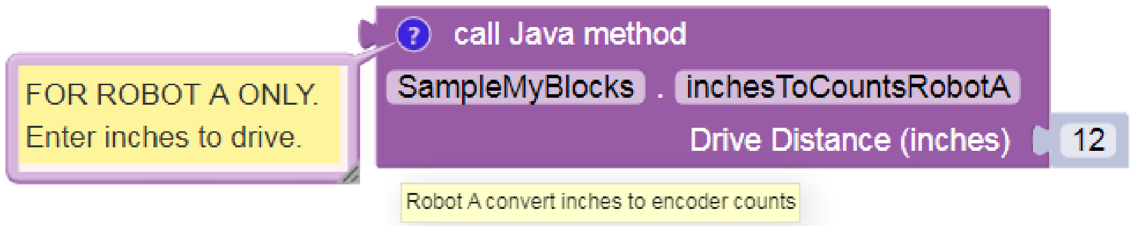
Line 34 shows the shared method that is **not** a myBlock. Simply omit the annotation @ExportToBlocks. The keyword `private` means the method can be called only from inside the same class. Use this whenever possible.

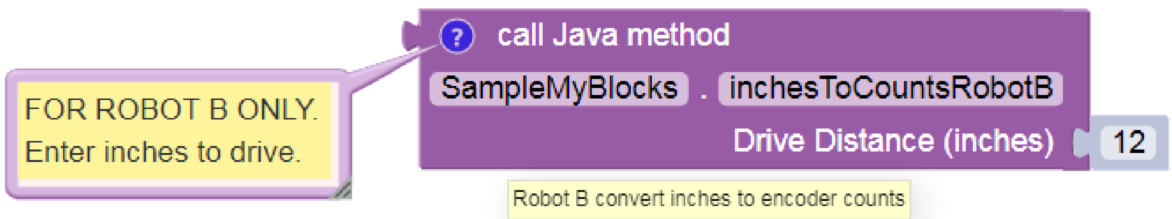
Lines 17 and 29 call the shared method. The method calls provide 3 parameters, which do not have the same **names** as the input parameters of the 'utility' method – but their types should match.

At line 38, **(int)** converts, or **casts**, a decimal number to integer type. This is called **type casting**. Programmers must pay close attention to compatible data types. For example, a DC motor `set .TargetPosition` Block should be given an encoder value as a simple integer, not a decimal number.

At line 15 and others, the keyword `final` indicates a Java **constant**: a variable that cannot change value. Java constants are traditionally ALL CAPS. Can you find the Math constant in this program?

Here are the Robot A and Robot B myBlocks, each with its **comment** balloon and **tooltip**. Very simple, as you wanted!





Note: This tutorial intends for you to **manually type** the Java code above. If you require pre-typed text of this example, click here. The linked copy includes a proper/full amount of Java commenting, omitted above to focus on the Java code. Also not shown are the package and import statements.

Example Code

SampleMyBlocks_v04.java

```
/*
This example is used in a tutorial on FTC myBlocks.
It shows how a non-myBlock shared 'utility' method can be called by myBlock methods.
This also has examples of Java constants and type casting.

In this example, the Java programmer has two goals:
1. Support two robots with different wheels and motors.
2. Keep the myBlocks very simple for the users.

Solution: one MyBlock per robot, specify only the distance to drive.
Each myBlock uses the appropriate wheel size and motor encoder CPR.
The myBlocks share a 'utility' method to convert distance to encoder counts.

*/

package org.firstinspires.ftc.teamcode;
```

(continues on next page)

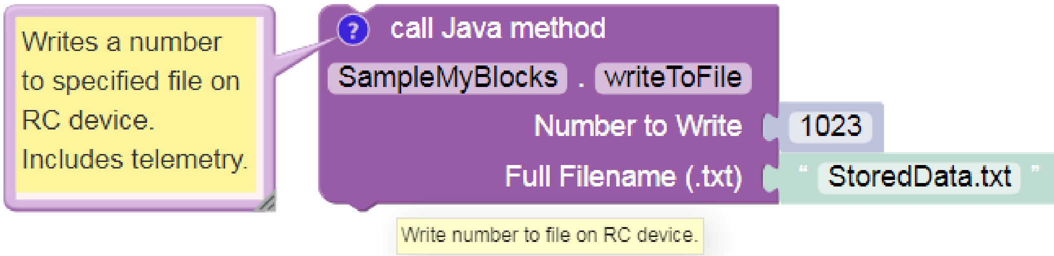
(continued from previous page)

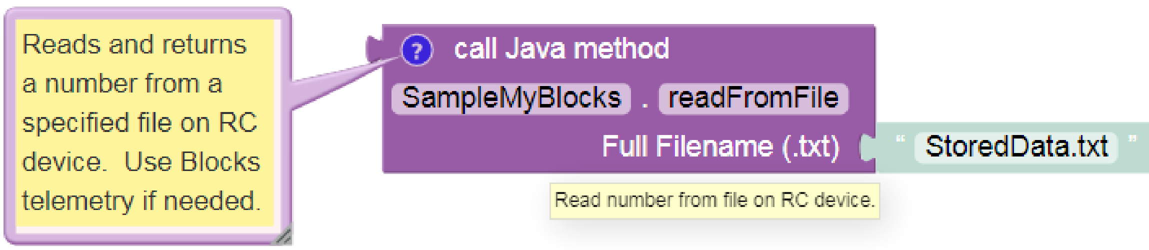
```
// OBJ and Android Studio automatically create these import statements.
import org.firstinspires.ftc.robotcore.external.ExportToBlocks;
import org.firstinspires.ftc.robotcore.external.BlocksOpModeCompanion;

// BlocksOpModeCompanion provides many useful FTC objects to this class.
public class SampleMyBlocks_v04 extends BlocksOpModeCompanion {

    // This annotation must directly precede a myBlock method
    @ExportToBlocks (
        comment = "FOR ROBOT A ONLY. Enter inches to drive.",
        tooltip = "Robot A convert inches to encoder counts",
        parameterLabels = "Drive Distance (inches)"
        )
    // This is a myBlock method with one input and one output.
    // The keyword 'final' indicates a Java constant: a variable that cannot change value.
    // Java constants are traditionally ALL CAPS.
    public static int inchesToCountsRobotA (double inchesToDriveA) {
        final double WHEEL_DIAMETER_A = 4.0;        // inches
        final double COUNTS_PER_ROTATION_A = 1120;  // CPR for NeveRest 40
        // call the shared utility method
        int countsToDriveA = calculateCounts (inchesToDriveA, COUNTS_PER_ROTATION_A, WHEEL_DIAMETER_
↪A);
        return countsToDriveA;                      // give the result to Blocks
    }   // end of method

    // This annotation must directly precede a myBlock method
    @ExportToBlocks (
        comment = "FOR ROBOT B ONLY. Enter inches to drive.",
        tooltip = "Robot B convert inches to encoder counts",
        parameterLabels = "Drive Distance (inches)"
        )
    // This is another myBlock method, also with one input and one output.
    // Both myBlocks will appear in the Blocks menu for this Java Class.
    public static int inchesToCountsRobotB (double inchesToDriveB) {
        final double WHEEL_DIAMETER_B = 3.0;        // inches
        final double COUNTS_PER_ROTATION_B = 537.6; // CPR for NeveRest Orbital 20
        // call the shared utility method
        int countsToDriveB = calculateCounts (inchesToDriveB, COUNTS_PER_ROTATION_B, WHEEL_DIAMETER_
↪B);
        return countsToDriveB;                      // give the result to Blocks
    }   // end of method

    // This is NOT a myBlock, it's a shared 'utility' method that is called by other methods.
    // It has 3 inputs (decimal numbers) and one output (integer).
    // The keyword 'private' means this method can be called only within this class.
    private static int calculateCounts (double inchesToDrive, double countsPerWheelRotation, double␣
↪wheelDiameter) {
        double circumference = wheelDiameter * Math.PI;
        double rotations = inchesToDrive / circumference;
        double encoderCounts = rotations * countsPerWheelRotation;
        return (int) encoderCounts;         // (int) casts or converts the decimal number to␣
↪integer type
    }   // end of method

}       // end of class SampleMyBlocks_v04
```

Example: Read-Write File Access

The current version of regular Blocks (SDK 7.0) does not provide **read/write access to an external file**, other than automatic Log or Match Log file entries. File access is a useful capability, available so far to Java programmers only. Now it can be done with myBlocks!

Here's an example pair of myBlocks. One myBlock **writes** a numeric value to a specified filename, and a companion myBlock can later **read** that value from the same file.





The file is stored on the Control Hub or RC phone, in the FIRST/settings folder. It exists separately from the RC app, OpModes, and other files.

Write and read actions can happen in the same OpMode or **different OpModes**, allowing various scenarios:

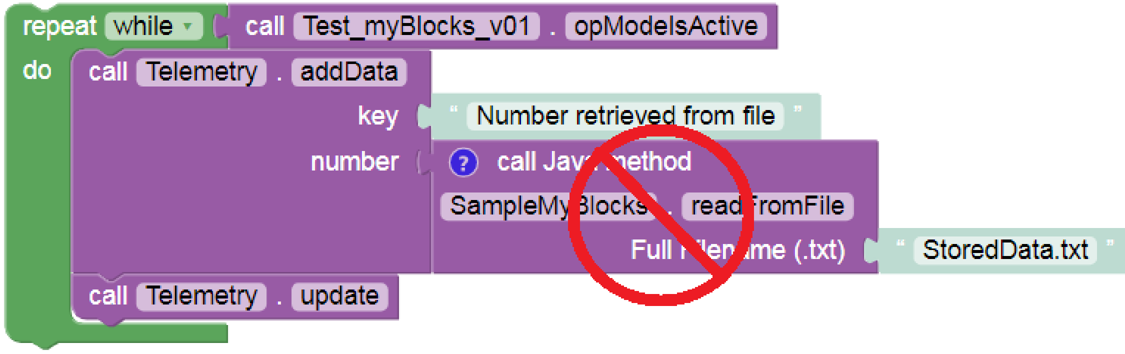
- Autonomous passes information to TeleOp. For example, what was the latest value of a sensor or encoder?
- A special **set-up OpMode** allows gamepad input to choose an autonomous strategy and adjust key parameters. The robot could then be idle for a long time, even turned off. When the match begins, the Autonomous OpMode would read those settings and implement the chosen/adjusted actions.
- A **dedicated log file** reports key sensor data in a custom format, with optional time-stamps. For program development and debugging, this could be more efficient than working with the large standard logs or Match Logs.

The Java code for this example is available below, with **extensive comments** that explain some unfamiliar Java expressions. The code can be copied and pasted directly into OnBot Java or Android Studio.

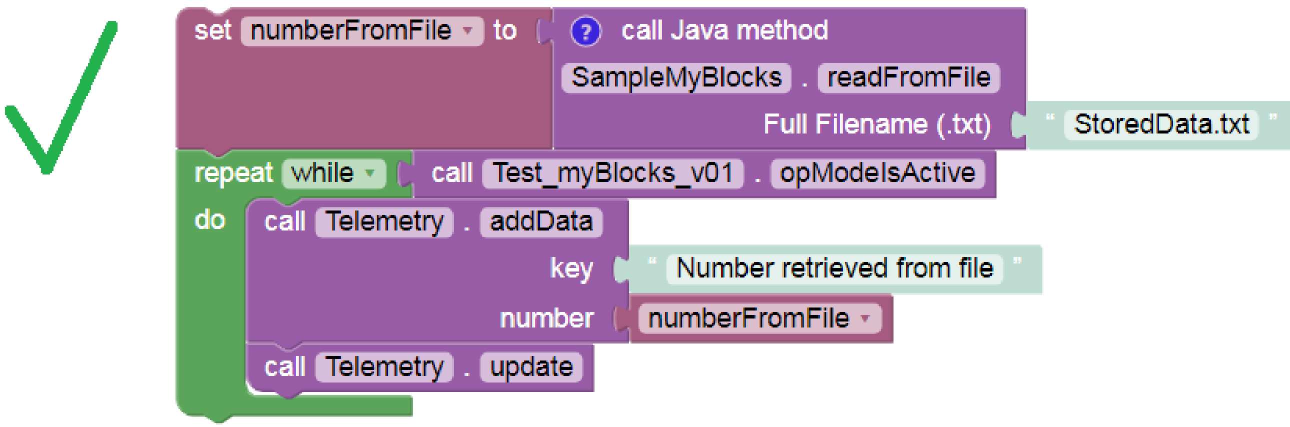
> Programming tip: Instead of memorizing every possible Java command, programmers often study and modify existing code for a similar task. Unfamiliar commands are explored with an internet search, reference book, at the Javadoc reference, or at the official Oracle Javadoc.

This simple example supports only a single numeric value per filename. Better versions would allow multiple values and data types – a good programming challenge!

Be careful about placing **myBlocks inside loops**. Expanding on the current example, your myBlock might read a larger amount of (unchanging) data from a file. If your OpMode needs that data only once, reading the file in a loop needlessly adds cycle time and might increase the risk of a corrupt or interrupted read operation.



Instead, read the file once and store the relevant data in a variable or array. Then process the variable as needed, inside the loop.



The same suggestion might apply to reading sensors and encoders, if the data are not changing and are needed only once.

Example Code

SampleMyBlocks_v05.java

```
/*
This example is used in a tutorial on FTC myBlocks.
It shows how one myBlock can write a number to a file on the RC phone
or Control Hub, and another myBlock can read that number from the file.

This is not possible with regular Blocks, in FTC app version 6.1.

This example assumes a team wants to store and retrieve a *number*, not text.
The file operations shown here are intended for storing and retrieving
text information. So this example stores a number as text, requiring conversion
when writing and when reading.

If a team instead wants to store an actual text string, this example could be
simplified (conversions not needed).

A challenge for the student: write and read *multiple* values, that might
be used for robot set-up, calibration, or choices of autonomous program.


Note 1: The method getSettingsFile() retrieves the settings (including location)
        of the named file.  If the file doesn't already exist, it is created
```

(continues on next page)

(continued from previous page)

```
        in the FIRST/settings folder.  Put the filename in quotes if it's not
        already a declared variable of type String.

Note 2: There is also a method copyFile(File fromFile, File toFile).

Note 3: The method String.valueOf() reads a numerical value as a String.

Note 4: The parseDouble() method interprets a String value as a double.
        The trim() method removes any leading or trailing white space.

Note 5: The write and read myBlocks can omit the filename parameter, if a team
        always uses the same file.  In such case the filename can be declared
        once at the class level (must be static), and used by all myBlock
        methods.  Like this:
        static File myFileName = AppUtil.getInstance().getSettingsFile("myTestFile.txt");

Note 6: The class ReadWriteFile does not appear to have a method for
        appending to a file.  Might need to use java.io.Writer.write() or a
        java.io.FileWriter method.
*/

package org.firstinspires.ftc.teamcode;

// these are (usually!) added automatically by OnBotJava when needed
import org.firstinspires.ftc.robotcore.external.BlocksOpModeCompanion;
import org.firstinspires.ftc.robotcore.external.ExportToBlocks;

import com.qualcomm.robotcore.util.ReadWriteFile;
import org.firstinspires.ftc.robotcore.internal.system.AppUtil;
import java.io.File;

public class SampleMyBlocks_v05 extends BlocksOpModeCompanion {

    // This Annotation must appear immediately before any myBlock method.
    // It's optional to add a comment, tooltip, and/or parameterLabels.
    // Comment must appear on a single line, no rollovers.
    @ExportToBlocks (
    comment = "Writes a number to specified file on RC device. Includes telemetry.",
    tooltip = "Write number to file on RC device.",
    parameterLabels = {"Number to Write", "Full Filename (.txt)"}
    )
    // This myBlock method writes a number (as text) to a file.
    // It has 2 inputs and no outputs (keyword void).
    public static void writeToFile (double myNumber, String toFileName) {

        // Using the properties of the specified "to" file name,
        // declare a filename to be used in this method.  See Note 1 above.
        File myFileName = AppUtil.getInstance().getSettingsFile(toFileName);

        // Write the provided number to the newly declared filename.
        // See Note 3 above.
        ReadWriteFile.writeFile(myFileName, String.valueOf(myNumber));

        telemetry.addData("Filename", toFileName);
        telemetry.addData("Number being written", myNumber);
        telemetry.update();         // display info on Driver Station screen

    }   // end of method writeToFile()
```

(continues on next page)

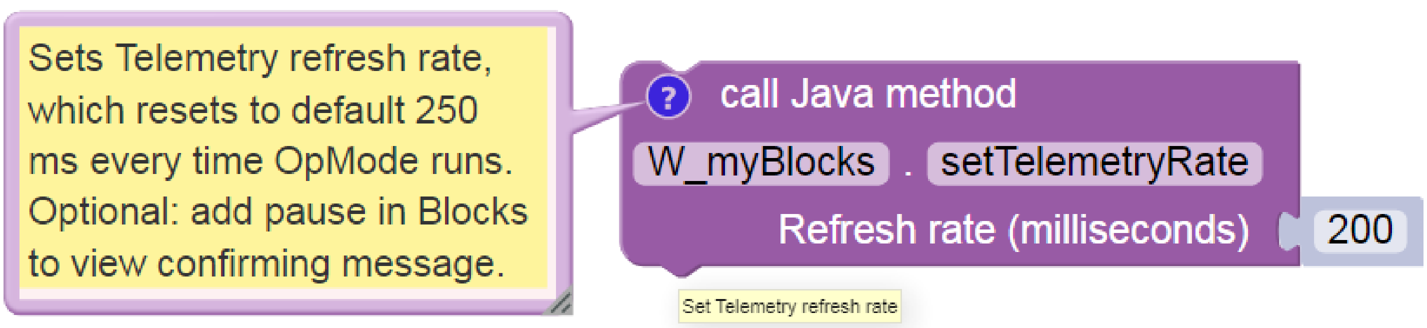
(continued from previous page)

```
    @ExportToBlocks (
    comment = "Reads and returns a number from a specified file on RC device." +
              " Use Blocks telemetry if needed.",
    tooltip = "Read number from file on RC device.",
    parameterLabels = "Full Filename (.txt)"
    )
    // This myBlock method reads a number (as text) from a file.
    // It has 1 input and 1 output (type double).
    public static double readFromFile (String fromFileName) {

        // Using the properties of the specified "from" file name,
        // declare a filename to be used in this method.  See Note 1 above.
        File myFileName = AppUtil.getInstance().getSettingsFile(fromFileName);

        // Read and store a number from the newly declared filename.
        // See Note 4 above.
        double myNumber = Double.parseDouble(ReadWriteFile.readFile(myFileName).trim());

        return myNumber;       // provide the number to the Block calling this myBlock

    }  // end of method readFromFile()

}   // end of class SampleMyBlocks_v05
```

Example: Modify Telemetry Settings

Telemetry messages are sent from the Robot Controller to the Driver Station up to **four time per second**, by default. This maximum refresh rate can be changed with Android Studio or OnBot Java, but **not** with regular Blocks. Now a myBlock can provide that capability too!

This simple example allows a Blocks user to change the standard time interval from 250 milliseconds to any other interval.



A lower time interval can allow faster update of sensor or encoder data. A higher interval can ease the RC-DS communication bandwidth load.

Here's the Java code for the method only:

```java
    // This method has 1 input and no outputs (keyword void).
    public static void setTelemetryRate (int myRate) {

        // Get and store the existing (default) minimum interval between
        // Telemetry transmissions from Robot Controller to Driver Station.
        int oldRate = telemetry.getMsTransmissionInterval();

        // Set the minimum interval, provided by the Blocks user.
        telemetry.setMsTransmissionInterval(myRate);

        // For confirmation, get and store the updated interval.
        int newRate = telemetry.getMsTransmissionInterval();

        telemetry.addData("TELEMETRY REFRESH RATE in milliseconds", null);
        telemetry.addData("Default/previous rate", oldRate);
        telemetry.addData("Requested rate", myRate);
        telemetry.addData("Confirmed new rate", newRate);
        telemetry.update();         // display info on Driver Station screen

    }   // end of method setTelemetryRate()
```

Note: This tutorial intends for you to **manually type** the Java code above. If you require pre-typed text of this example, click below. The linked copy includes the usual class declaration and package/import statements.

Example Code

W_myBlocks.java

```java
/*
v01 dated 12/20/2020

This FTC myBlocks example allows the Blocks user to modify the Telemetry
refresh rate from the standard 4 cycles per second (4 Hz or 250 ms interval).
A lower time interval can allow faster update of sensor or encoder data.
A higher interval can ease the RC-DS communication bandwidth load.

This feature is not available with regular Blocks, in FTC app version 6.1.

For more controls, click Telemetry in the left side column here:
https://first-tech-challenge.github.io/FtcRobotController/6.0.1/RobotCore/index.html

The top-level API Documentation for the FTC SDK is here:
https://first-tech-challenge.github.io/FtcRobotController/
*/

package org.firstinspires.ftc.teamcode;

import org.firstinspires.ftc.robotcore.external.BlocksOpModeCompanion;
import org.firstinspires.ftc.robotcore.external.ExportToBlocks;
// Don't need to import Telemetry class, provided with BlocksOpModeCompanion.

public class W_myBlocks extends BlocksOpModeCompanion {
```

(continues on next page)

(continued from previous page)

```
    // This Annotation must appear immediately before any myBlock method.
    // Optional to add a comment, tooltip, and/or parameterLabels.
    // Comment must be a single text line, concatenation (+) allowed.
    @ExportToBlocks (
    comment = "Sets Telemetry refresh rate, which resets to default 250 ms " +
    "every time OpMode runs.  Optional: add pause in Blocks to view " +
    "confirming message.",
    tooltip = "Set Telemetry refresh rate",
    parameterLabels = {"Refresh rate (milliseconds)"}
    )
    // This method has 1 input and no outputs (keyword void).
    public static void setTelemetryRate (int myRate) {

        // Get and store the existing (default) minimum interval between
        // Telemetry transmissions from Robot Controller to Driver Station.
        int oldRate = telemetry.getMsTransmissionInterval();

        // Set the minimum interval, provided by the Blocks user.
        telemetry.setMsTransmissionInterval(myRate);

        // For confirmation, get and store the updated interval.
        int newRate = telemetry.getMsTransmissionInterval();

        telemetry.addData("TELEMETRY REFRESH RATE in milliseconds", null);
        telemetry.addData("Default/previous rate", oldRate);
        telemetry.addData("Requested rate", myRate);
        telemetry.addData("Confirmed new rate", newRate);
        telemetry.update();         // display info on Driver Station screen

    }   // end of method setTelemetryRate()

}   // end of class W_myBlocks
```

Want to verify this actually works? Another, slightly more advanced myBlock allows measuring the time between Telemetry updates; it's posted below. That myBlock can be used in a Blocks program like the one attatched below; download the raw **.blk file** and click the **Upload Op Mode** button at the main Blocks menu. Read all comments and instructions.

Example Code

`W_myBlocks_Telemetry_v02.java`

`W_Telemetry_myBlocks_v02.blk`

```
/*

v01 dated 12/20/2020

This FTC myBlocks example allows the Blocks user to modify the Telemetry
refresh rate from the standard 4 cycles per second (4 Hz or 250 ms interval).
A lower time interval can allow faster update of sensor or encoder data.
A higher interval can ease the RC-DS communication bandwidth load.

This feature is not available with regular Blocks, in FTC app version 6.1.

For more controls, click Telemetry in the left side column here:
https://first-tech-challenge.github.io/FtcRobotController/6.0.1/RobotCore/index.html
```

(continues on next page)

(continued from previous page)

```
The top-level API Documentation for the FTC SDK is here:
https://first-tech-challenge.github.io/FtcRobotController/


v02 dated 12/20/2020

Add myBlock "telemetryAction" to allow cycle testing.

*/

package org.firstinspires.ftc.teamcode;

import org.firstinspires.ftc.robotcore.external.BlocksOpModeCompanion;
import org.firstinspires.ftc.robotcore.external.ExportToBlocks;

public class W_myBlocks_Telemetry_v02 extends BlocksOpModeCompanion {

    // This Annotation must appear immediately before any myBlock method.
    // Optional to add a comment, tooltip, and/or parameterLabels.
    // Comment must be a single text line, concatenation (+) allowed.
    @ExportToBlocks (
    comment = "Sets Telemetry refresh rate or interval, which resets to " +
    "default 250 ms every time OpMode runs.  Optional: add pause in Blocks " +
    "to view confirming message.",
    tooltip = "Set Telemetry refresh interval",
    parameterLabels = {"Refresh interval (milliseconds)"}
    )
    // This myBlock method has 1 input and no outputs (keyword void).
    public static void setTelemetryRate (int myRate) {

        // Get and store the existing (default) minimum interval between
        // Telemetry transmissions from Robot Controller to Driver Station.
        int oldRate = telemetry.getMsTransmissionInterval();

        // Set the minimum interval, provided by the Blocks user.
        telemetry.setMsTransmissionInterval(myRate);

        // For confirmation, get and store the updated interval.
        int newRate = telemetry.getMsTransmissionInterval();

        telemetry.addData("TELEMETRY REFRESH RATE in milliseconds", null);
        telemetry.addData("Default/previous rate", oldRate);
        telemetry.addData("Requested rate", myRate);
        telemetry.addData("Confirmed new rate", newRate);
        telemetry.update();         // display info on Driver Station screen

    }   // end of method setTelemetryRate()



    // initialize toggle indicating end of current Telemetry interval
    static boolean readyToBroadcast = false;

   @ExportToBlocks (
    comment = "At each scheduled Telemetry update, return value 1 " +
              "to increment counter.  Otherwise return 0.",
    tooltip = "Action before Telemetry update"
    )
    // This myBlock method has no inputs and one output of type int (integer).
```

(continues on next page)

(continued from previous page)

```
    public static int telemetryAction() {

        // Create a named list of actions to be run when specified.
        Runnable myActions = new Runnable()  {
            @Override
            public void run()  {
                // one action here, could be a list
                readyToBroadcast = true;        // toggle: end of interval
            }
        };

        // The method addAction() runs the indicated action list only if
        // the Telemetry interval (of the Blocks OpMode) has elapsed.
        telemetry.addAction (myActions);

        if (readyToBroadcast) {             // Telemetry interval has elapsed
            readyToBroadcast = false;       // reset the interval toggle
            return 1;                       // send a 1 for cycle counter
        }
        else return 0;                      // send 0 if interval not elapsed

    }   // end of method telemetryAction()

}   // end of class W_myBlocks_Telemetry_v02
```

Ideas for Other myBlocks

MyBlocks offer great potential for creativity and robot capability. Start by programming myBlocks for tasks that an existing group of Blocks can do. Later, add functions that are **not available** with regular Blocks. Here are some examples of both:

- Set one or more program variables during INIT, **using the gamepad**. This can be done with regular Blocks, but a good User Interface (UI) requires multiple long and complex Functions.
- Create driving actions with **multiple sensor controls**. For example, gyro-based steering towards a distance goal (ultrasonic or vision target). Or Run_To_Position while following a line. A myBlock can provide Blocks users with controls previously considered too complex.
- Provide access to **External Libraries**, new for SDK 7.0. More info is *here*.
- One of the above examples controls a servo specified by the Blocks user. This could lead to a **family of separate myBlocks** to interact with 1 device, 2 devices, etc. Or a generic single myBlock could interact with, say, up to 4 DC motors. The Java method would process only those DC motors with a filled-in parameter name.
- Control the **LED flashlight** on the RC phone?
- Could **telemetry.speak** have a myBlock equivalent of the Boolean `AndroidTextToSpeech.isSpeaking()`?

Looking for ideas? The top-level API Documentation for the SDK is here. Click **RobotCore** to see many commonly used classes in the left-side menu, and you can also check other sections.

Do you have suggestions or a good example to share? Send to westsiderobotics@verizon.net.

Here are some tips for efficiency, from the developer Liz Looney:

- Limit the number of method calls. Calling a single myBlock that does 5 tasks uses less overhead than calling 5 myBlocks that each do one task.
- Limit the number of parameters. If your myBlock needs certain information that won't change during the OpMode, use an **initialize method** that's called once at the start of the OpMode. The initialize method stores that information, to avoid repeatedly passing the same parameter each time the myBlock is called.

Summary: Benefits of myBlocks

1. MyBlocks now provide access to the full range of Java in the Software Development Kit (SDK). Blocks programming can now perform tasks **previously unavailable** to Blocks-only teams. This now includes *External Libraries*.
2. MyBlocks can neatly package previously **long or complex Functions** in Blocks.
3. MyBlocks programming allows some team members to begin learning and using Java, contributing valuable new features. The other team members can continue learning and working in Blocks, producing the team's official code. Nobody is held back, or left behind.
4. MyBlocks can be created with **OnBot Java**, which runs on the RC phone or Control Hub. Building and testing are very fast. Many teams do not have easy access to Android Studio, for reasons including school computers that prevent software installation.
5. By developing and sharing myBlocks, experienced teams could **help new teams** in a more direct way, beyond simply posting a link to their Java library. The *FIRST* Tech Challenge community might ultimately benefit from a curated repository for tested, well documented myBlocks. Perhaps the "Blocks Store"?

Hats off to Google engineer Liz Looney for this major development!

Questions, comments and corrections to westsiderobotics@verizon.net

21.7.5 External Libraries in OnBot Java and Blocks

Introduction

Blocks and OnBot Java programmers can use external libraries, starting with SDK 7.0 released for the Freight Frenzy season. This capability previously existed for programmers using Android Studio.

An external library is a collection of specialized software ready for public use, and typically available from a website or repository, called a 'repo'. You don't need to know its inner workings, just what it does and how to use it.

This beginner-level tutorial shows how to incorporate a library's features into your Op Modes, and provides simple examples. It does not teach Java.

Many thanks to Liz Looney who developed this capability, along with myBlocks and many other useful features of the software.

Note: This new capability exists for a Robot Controller (RC) running Android 7 & and higher. Moto G 2nd Gen and Moto G 3rd Gen RC phones cannot use this feature.

Overview

It's a simple process with three basic steps.

Step 1, find a library with features you want to use, and get its .jar or .aar file.

Step 2, upload that file in OnBot Java.

Step 3 has three variations, depending on your planned use of the library.

Step 3A assumes you want to use a library function, or *method*, in OnBot Java only. Namely you are **not** planning to make that method available for Blocks users.

Step 3B assumes that you **do** want to provide the library method to a Blocks user, by creating a special Block called a myBlock. MyBlocks are not new with 7.0, but now you can use library code in that myBlock. You can create your myBlock to have the exactly same inputs and outputs as the library method, or slightly different inputs and outputs.

Step 3C is a different scenario, where the library method is made available directly to the Blocks user, **without creating a myBlock**. This can be done, if the library is specially annotated by its author.

Step 1 - Select library, get archive file

There are hundreds of code libraries available on the internet. In Step 1, you find a library with specialized functions (methods) that you want to use. Start with a web search, or get suggestions from other teams.

It's best to choose a repo with a well documented interface. Review its documentation or API, learn what the methods can do, and learn about inputs required and output provided.

If that's something you want to use, then try to find the .jar or .aar file. The `.jar` suffix means Java Archive and `.aar` means a similar format called Android Archive. This is a compressed file containing the entire collection of code that you want, in a single file. It's similar to a *zip file* that you may have used for general web downloads.

If you're having trouble locating that .jar file, or if you're not sure how to use that library, you are encouraged to contact the repo owner or developer. They often enjoy hearing how programmers plan to use their code, and would be happy to help you get started. Don't be intimidated, they are often willing to help.

Then just download that .jar or .aar file to local storage such as your laptop or to a team Google drive.

Step 2 - Upload archive file

Copy the .jar or .aar file to your programming laptop, if the file is not stored there already.

Connect your laptop via Wi-Fi to a Robot Controller device that's running the RC app, version 7.0 or higher (see instructions at `Program and Manage`, on the RC phone or its paired Driver Station device). In the Chrome browser, open OnBot Java.

In OnBot Java (OBJ) click the **upload icon**, normally used to upload a regular Java file to the `teamcode` folder.

Fig. 238: Upload icon in OnBot Java

Instead of a Java file, select the .jar or .aar library file to upload from the laptop.

OnBot Java will recognize that it's an archive file, and will automatically create a folder called `ExternalLibraries`. This folder will appear above the `teamcode` folder.

Step 3 - Programming

The programming step has several variations depending on how you're planning to use the External Library code.

Step 3A assumes you want to use the library method only in your Java program or OpMode, and **not** provide it to Blocks users. Or perhaps your team doesn't work with Blocks at all.

So, you can just start programming!

First import the class. You know the **class name** because you read about it in the library documentation. And you know the **method names** and how they work, including the inputs and outputs (and their Java types). This tutorial does not offer Java programming instruction.

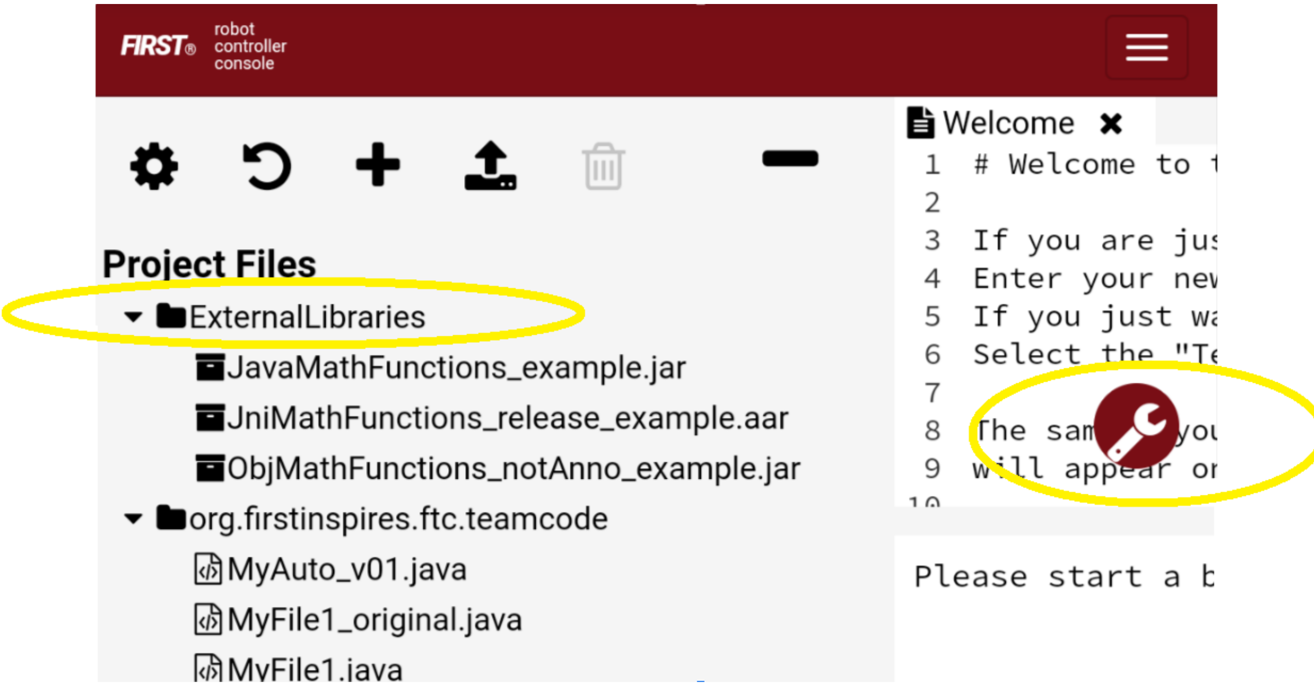


Fig. 239: ExternalLibraries folder and Build Everything icon

Remember, when all your programming is finished, click the `Build Everything` icon (see image above).

In **Step 3B** you create a myBlock, which can modify the library method, or not. It depends on the functionality you want to provide to your Blocks programmer.

Unmodified means that your myBlock simply calls the library method, using exactly the same inputs and outputs. This might be called a *wrapper method*.

Or, you can use the library method as a **utility function**, performing a specific task in or for your larger myBlock method. In other words, your myBlock **supplements** what the library method can do.

Alternatively, the library method might have **more** parameters than your Blocks user needs, so you want a **simpler** interface. Your OBJ code can satisfy extra parameters without exposing them to the user of your myBlock.

As a reminder, creating a myBlock requires only these two items:

- Extend your existing OpMode class with `BlocksOpModeCompanion`.
- Place the annotation `@ExportToBlocks` immediately before each myBlock method.

For more info, see the separate myBlock tutorial *here*.

Step 3C assumes the library methods should be provided **directly** to the Blocks user, without even creating a myBlock. This scenario doesn't need to be enabled with your Java code at all.

Instead you must ask the **library developer** to add two annotations, then provide you a fresh .jar or .aar file. The changes are:

- Place the annotation `@org.firstinspires.ftc.robotcore.external.ExportClassToBlocks` directly before the library class declaration.
- Place the annotation `@org.firstinspires.ftc.robotcore.external.ExportToBlocks` directly before each library method to be exposed (shared or passed through to Blocks).

When you have that archive file with its annotations, upload it in OnBot Java, and click `Build Everything`. That's it!

It doesn't matter which Java file (if any) is currently open in OBJ; no such file is needed for this feature. Those library-annotated "pass-through" methods will automatically appear in the Blocks toolbox (menu), with the method's actual inputs and outputs.

See further below for examples of these 3 scenarios.

User Training & Documentation

For Step 3B or Step 3C, your **Blocks users** must be taught how to use the new myBlocks or the new pass-through Blocks. That's **your job**, as the Java developer who implemented this feature.

Start with good documentation. For myBlocks, use the existing tools to make helpful labels (for input parameters), clear tooltips and detailed comments. The comments appear in a text box that expands after clicking the blue question-mark icon on that Block. Tell your users it's there.

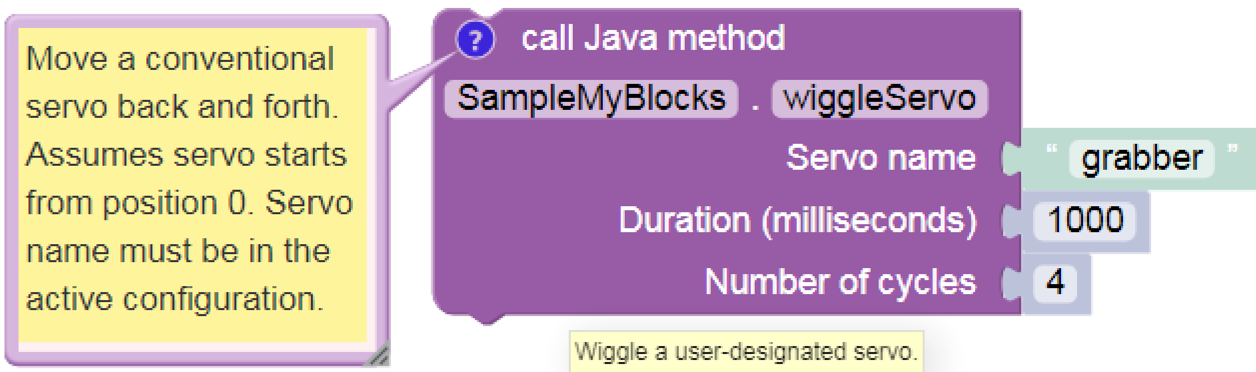


Fig. 240: myBlock documented using comment, tooltip and input labels

Meet with your team's Blocks programmer(s) to explain the new features. Consider writing a short description, for their future reference. Encourage users to give you feedback, to improve your code. Congratulations, you are now a Java developer!

Benefits

Obviously this External Libraries feature provides advanced functions previously available only to Android Studio teams.

Secondly, more of your team members can continue programming the robot in Blocks. Meanwhile other students (like you), if they want to, can advance their Java skills and still contribute to the team's actual robot programming.

Often, teams have one student who has moved far ahead with their Java skills, and becomes the team programmer – the **only** programmer. Then nobody else has the chance to learn and contribute basic programming.

This feature can allow a new arrangement: "nobody is left out, and nobody is held back".

As a third benefit, judges love to hear about Outreach. For example your team could develop useful Blocks for beginner teams. Or, you share ideas and tips with other advanced teams who are doing the same kind of development. And, you are encouraged to communicate with library developers. This is a good opportunity for real-world interaction with specialists: sharing your needs, and receiving expert guidance. Scientists, engineers, doctors, entrepreneurs – nobody needs to reinvent the wheel. Professional life is built on these interactions.

Example 1 - non-annotated library

The first example uses a very basic "homemade" library called Geometry For OBJ. To get your own copy, click `here`.

As with any current real-world library, this one is **not annotated** for use. You can use it in OnBot Java only (Step 3A), **or** you can create a myBlock (Step 3B) to share its capabilities with Blocks programmers. Lacking annotations, this library does not provide direct "pass-through" methods (Step 3C) to Blocks.

This library contains a class called `com.example.google.ftc.Geometry`, with three methods: - `circleCircumference()` accepts radius, returns circumference - `circleArea()` accepts radius, returns area - `hypot()` accepts 2 lengths, returns hypotenuse of right triangle

Under **Step 3A**, you would use, for example, the `hypot()` method for your own OnBot Java programming, not providing it to Blocks.

Add this to your list of import statements:

```
import com.example.google.ftc.Geometry;
```

Then simply use the method in your Java code:

```
double A = 3.0;
double B = 4.0;
double myHypotenuse = Geometry.hypot(A, B);
```

Under **Step 3B**, let's create your own custom Block called "myHypotenuse". *This is just an exercise; regular Blocks could easily calculate this value.*

You will still need the `import` statement, same as above in Step 3A.

Then, extend the main class:

```
public class librariesExample extends BlocksOpModeCompanion {
```

The myBlock method might read:

```
@ExportToBlocks (
    comment = "This myBlock returns the hypotenuse (longest side) of the right triangle" +
              " with legs whose lengths are specified by the two given numbers.",
    tooltip = "calculate hypotenuse of 2 sides",
    parameterLabels = {"side a", "side b"}
)
public static double myHypotenuse(double a, double b) {
    return Geometry.hypot(a, b);
}
```

This myBlock contains only the library method and uses the same inputs and output, an example of a 'wrapper method'.

Note that myHypotenuse() is a `static` method, required for all myBlock methods. Also note that parameter labels are allowed to be different than the actual method parameters. Learn more about myBlocks *here*.

Here is the myBlock that will appear in the Blocks toolbox (menu):

On your own, you can try this with the two remaining methods. Use myBlocks to show telemetry output of various input values.

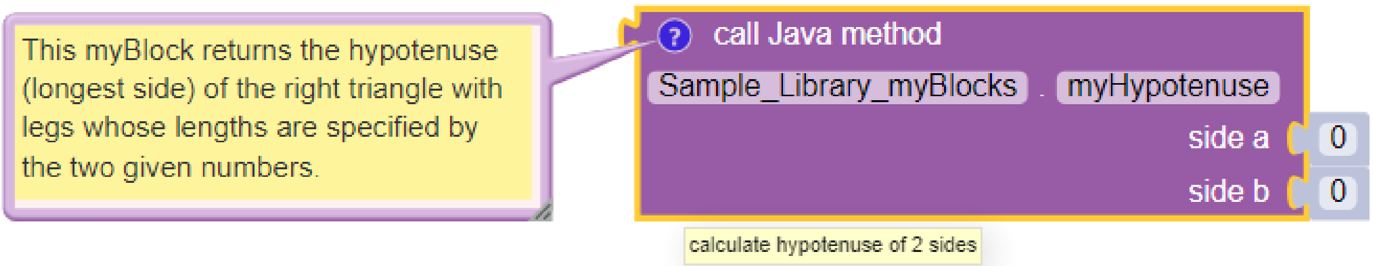


Fig. 241: myBlock using library method `Geometry.hypot()`

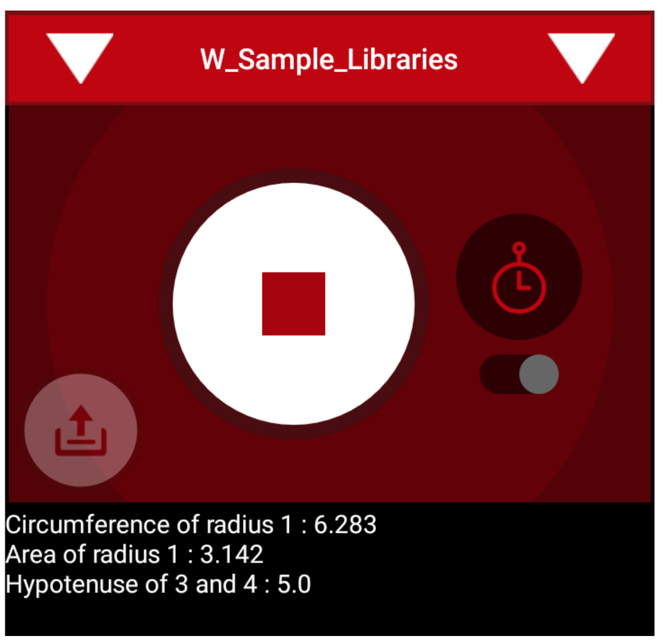


Fig. 242: Telemetry of myBlocks using Geometry library

Example 2 - *FIRST* Tech Challenge-annotated library

Now let's try another "homemade" library that **does** already contain the annotations. This one is called Arithmetic For Blocks; click `here`.

This library contains a class name `com.example.google.ftc.MoreMath`, with public methods `sum`, `min`, `max` and `average`. Each accepts two numbers and provides a numeric result.

This library **is annotated** specifically for team use, as described above. After you upload the .aar file and `Build Everything`, its 4 "pass-through" methods will automatically appear as Blocks:

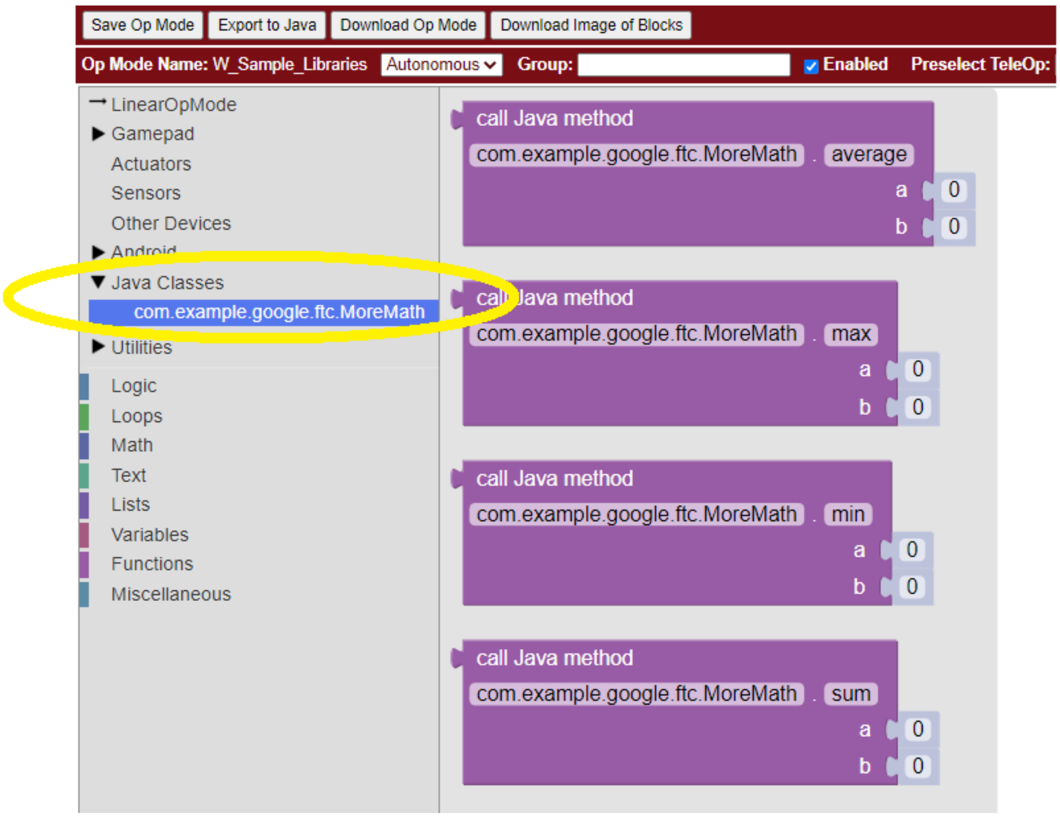


Fig. 243: Pass-through methods from class MoreMath in annotated library Arithmetic For Blocks

You **could** also use these methods in OnBot Java, including to create a myBlock. For example, perhaps you want to also provide a custom version of a pass-through method. But you **don't need** an OnBot Java file to support this library or its methods; that's done automatically by OnBot Java when it processes the library at upload.

What if you have an annotated library, and don't want **any** of its methods to appear as a Blocks pass-through? Just `Build Everything`, then delete the .jar or .aar file.

Here are two other "homemade" libraries, both **annotated**. Feel free to experiment with these.

- `JniExample.aar` contains a class named `com.example.google.ftc.IntegerMath`, with methods for simple arithmetic operations, implemented in native C++ code. Its public methods are `add`, `subtract`, `multiply`, and `divide`. Each accepts two integers and provides an integer result.
- `RevPotentiometer.aar` contains a class named `com.example.google.ftc.RevPotentiometer`, which is a hardware device class for the REV Potentiometer. It uses `AnalogSensorType` and `DeviceProperties` annotations to make this sensor appear in the "Configure Robot" menu of the RC app or paired DS app. After the .aar file has been uploaded (and `Build Everything`), configure your robot's Analog Input Devices and choose `REV Potentiometer`. It has a public method `getRotation` with parameter of type AngleUnit.

Real-world libraries

External Libraries have unique content and structure. Each may pose special challenges as you try to use it in robot code. Communication with the library developer will be very helpful, perhaps essential.

Ideally, the library's .jar or .aar file encompasses all the classes you'll need, without external dependencies. A good example is **EasyOpenCV**, designed and ready for use. See the simple instructions here.

General external libraries might involve a longer journey. For example, Apache Commons is a vast public repo, basically a library of libraries, focused on the Java programming language. Complications can arise even when choosing a simple math-only library.

Apache libraries are organized into Modules, typically each with one or more .jar files. It may not be sufficient to upload only the .jar file that seems to contain the class and methods you want to use.

If the library code refers to a class **not contained** in that .jar file, OnBot Java's auto-complete feature may eventually throw a 'class not found' exception, causing your RC app to crash. The exception triggered by this 'hidden dependency' may occur within minutes or hours, whenever OBJ encounters the 'missing' class – even if your OpMode does not directly or indirectly use that class. After that point, your RC app will not operate. It can operate again only by manually deleting the .jar file and its associated folder, directly on the RC device.

Starting over, you can find and upload the .jar file containing the 'missing' class. But that may expose further dependencies, requiring more .jar files.

Also, be aware that the SDK already contains some common Apache classes. OnBot Java may detect this duplication, preventing upload of your .jar file. On the bright side, your desired methods should already be available!

So, be prepared for these and other challenges that may arise. Again, it's helpful to communicate with the library developer where possible.

Advanced

Here are some technical details that might apply to very advanced use of the External Libraries feature. These are not covered in this basic tutorial.

- .aar files with assets are not supported
- External libraries can include .so files for native code
- External libraries can add new hardware devices with these annotations:

```
com.qualcomm.robotcore.hardware.configuration.annotations.AnalogSensorType
com.qualcomm.robotcore.hardware.configuration.annotations.DeviceProperties
com.qualcomm.robotcore.hardware.configuration.annotations.DigitalIoDeviceType
com.qualcomm.robotcore.hardware.configuration.annotations.I2cDeviceType
com.qualcomm.robotcore.hardware.configuration.annotations.MotorType
com.qualcomm.robotcore.hardware.configuration.annotations.ServoType
```

- External libraries can add new functionality to the Robot Controller with these annotations:

```
org.firstinspires.ftc.ftccommon.external.OnCreate
org.firstinspires.ftc.ftccommon.external.OnCreateEventLoop
org.firstinspires.ftc.ftccommon.external.OnCreateMenu
org.firstinspires.ftc.ftccommon.external.OnDestroy
org.firstinspires.ftc.ftccommon.external.WebHandlerRegistrar
```

Summary

Blocks and OnBot Java programmers can benefit and learn from this new capability with external libraries.

You are encouraged to submit other examples and suggestions that worked for you.

Questions, comments and corrections to westsiderobotics@verizon.net

21.7.6 Universal IMU Interface

Introduction

In September 2022, REV Robotics began shipping Control Hubs with a different internal Inertial Measurement Unit (IMU). The new IMU chip is designated BHI260AP, replacing the existing Hub's IMU chip BNO055. Both are from Bosch Sensortec. An IMU can measure many aspects of device motion; this explanatory document focuses primarily on **rotation**.

The Software SDK **version 8.1** introduced a **universal interface** that supports both the BHI260AP and BNO055 IMU. This basic tutorial introduces some new features:

- robot configuration allows selection of IMU type
- universal classes and methods supporting both IMU types
- three ways to specify Hub mounting orientation on the robot

Teams wanting to use the newer IMU are required to:

- use SDK 8.1 or newer
- update the Control Hub OS to 1.1.3 or newer.

However **all teams** are encouraged to begin using the universal IMU classes and methods for **new** Blocks and Java code. And, migrating **existing code** would allow you to switch easily (and perhaps urgently) to a new Control Hub during the season.

Don't know which IMU you have? Check the **Manage** page under `Program & Manage` in any of these places:

- connected Driver Station (DS) app
- connected computer's Chrome browser, at `http://192.168.43.1:8080` (Control Hub) or `http://192.168.49.1:8080` (RC phone)
- REV Hardware Client (when Hub LED is green)

Each Hub's IMU type is listed there, as of SDK 8.0.

Note: *Reminder: REV Expansion Hubs purchased after December 2021 have no internal IMU.*

Do you have existing OpModes using the original IMU? Your code can run unchanged, using Hubs with the BNO055. The new SDK 8.1 fully supports legacy Blocks and Java code using classes and methods for the BNO055 IMU.

The SDK 8.1 README provides more technical background:

> Unlike the old `BNO055IMU` interface, which only worked correctly when the REV Hub was mounted flat on your robot, the `IMU` interface allows you to specify the orientation of the REV Hub on your robot. It will account for this, and give you your orientation in a Robot Coordinate System, instead of a special coordinate system for the REV Hub. As a result, your pitch and yaw will be 0 when your *robot* is level, instead of when the REV Hub is level, which will result in much more reliable orientation angle values for most

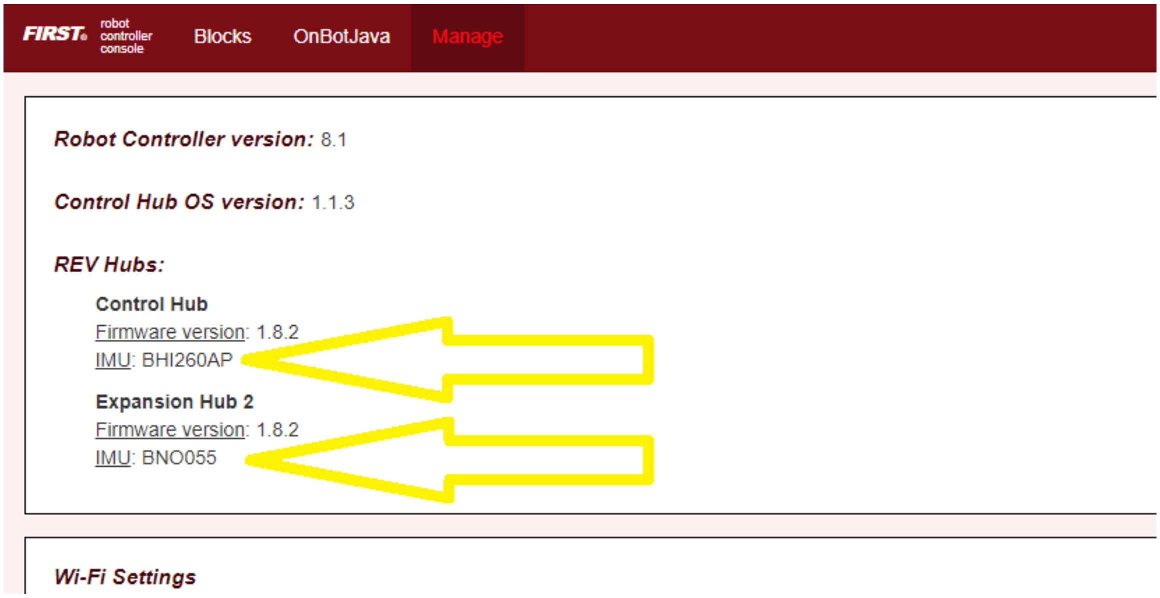


Fig. 244: Sample Control Hub and Expansion Hub display

mounting orientations.
…
If you have calibrated your BNO055, you can provide that calibration data to the new `IMU` interface by passing a `BNO055IMUNew.Parameters` instance to `IMU.initialize()`.
…
Because of the new robot-centric coordinate system, the pitch and roll angles returned by the `IMU` interface will be different from the ones returned by the `BNO055IMU` interface. When you are migrating your code, pay careful attention to the documentation.

Potential Usage

FIRST Tech Challenge robots drive mostly on a flat playing field, typically using the IMU to monitor or control **Heading** (Yaw or Z-angle).

Heading is preserved between OpMode runs, unless the robot or Robot Controller (RC) app are restarted. This can be useful between Autonomous and TeleOp. Heading can be reset during an OpMode, as discussed below.

Heading can drift slowly over time. An absolute reference is not available from gravity or from a magnetometer, which can be affected by nearby motors. This 'Yaw drift' is discussed below.

The IMU can help with more than Heading! Some *FIRST* Tech Challenge games have placed robots on **tilted surfaces**:

Such fields, and special circumstances in **any** *FIRST* Tech Challenge game, may cause teams to seek IMU readings for **Pitch** and **Roll** angles.

Examples might include:

- robot's left wheels are raised, on an obstacle
- robot is tilted forward on its front 4 wheels (of 6-wheel West Coast Drive)
- robot has tipped over (!)
- robot's secondary Expansion Hub (with IMU) is mounted on a tilting mechanism

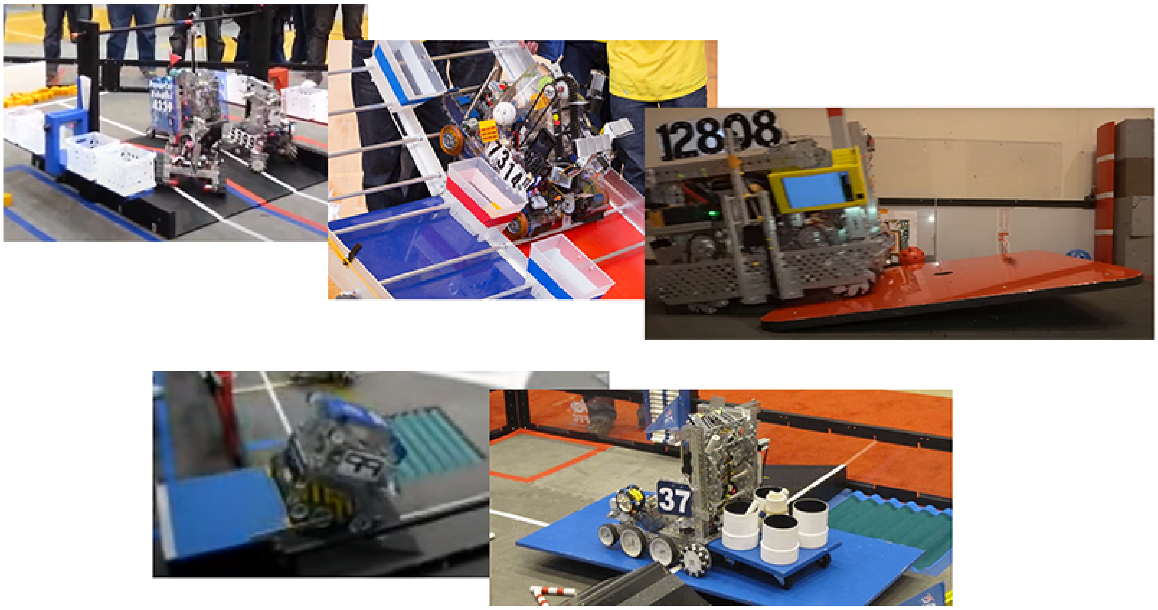


Fig. 245: Sample images from previous games utilizing tilted surfaces
(Block Party!, *FIRST* RES-Q, Relic Recovery, Face Off!, Get Over It!)

The Software SDK can also provide values for **angular velocity**, which is the rate of change (degrees per second) for Roll, Pitch or Yaw.

Let's get started!

Configure IMU

Robot configuration of the IMU is **automatic**, and shouldn't need changes. But here's how to confirm or rename your configured IMU.

In a connected DS app, touch the 3-dots icon at top right, then touch `Configure Robot`. For any new or existing Configuration, touch `Control Hub Portal`, then select the Hub with the IMU you want to use. Typically this will be the Control Hub, whether old or new.

- **Yellow**: The internal IMU is (always) connected at I2C Bus 0, Port 0. If you want another I2C device also on Bus 0, plug it into the Hub and use the Add button.
- **Green**: The default IMU type shown will reflect the actual unit in this Hub; fix this only if it was incorrectly modified. Your IMU OpModes **require a correct choice here**.
- **Purple**: The default device name is "imu", used by all Sample OpModes for Blocks and Java. You may enter a custom name here, but you must then **update** all your OpModes that reference the IMU.

When done, **save** and **activate** this configuration.

If a Blocks OpMode is open at the computer's programming screen, close and re-open that OpMode to capture this updated configuration. Blocks are provided only for devices in the configuration that's active **upon opening** *an OpMode.*

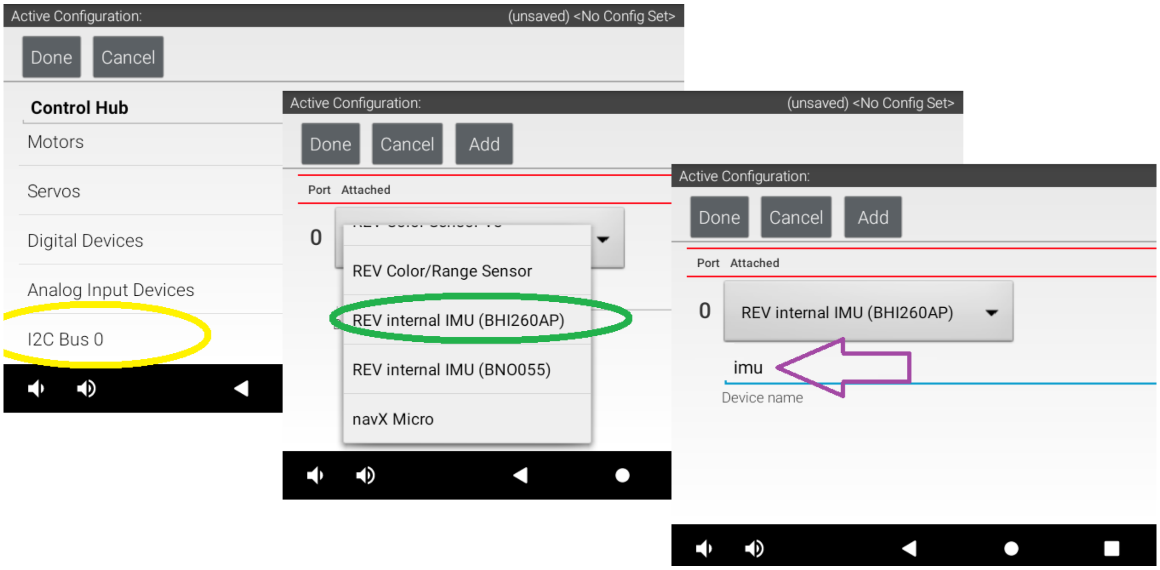


Fig. 246: REV IMU Robot Configuration Validation

Axes Definition

Robot orientation is defined using the Robot Coordinate System, with 3 axes that are **orthogonal** (at 90 degrees to each other), with origin inside the robot.

You must decide which face or direction is **"forward"** on your robot (which could be round!).

Tip: Placing a tape label "FRONT" at the **team-agreed front face** or front edge of the robot can avoid confusion later – really!

- Heading, or Yaw, is the measure of rotation about the Z axis, which points **up** toward the ceiling.
- Pitch is the measure of rotation about the X axis, which points **out the right side** of the robot.
- Roll is the measure about the Y axis, which points **out the front** of the robot.

These are Robot axes, different than (and not aligned with) the Hub axes used by the legacy `BNO055IMU` *driver.*

Rotation follows the traditional **right-hand rule**: with the thumb pointing along the positive axis, the fingers curl in the direction of **positive** rotation.

Hint: *Fun fact: the IMU is located approximately under the word "PROUD", near the lower right corner of the Hub.*

This tutorial will **not** discuss the *FIRST* Tech Challenge *Field Coordinate System*. Your OpModes might relate robot orientation to the overall field or 'global coordinates' for navigation, but that's beyond the focus here on using the IMU.

Physical Hub Mounting

Under SDK 8.1, you can specify the **physical orientation** of the Hub on the robot. This allows you to receive IMU angle values expressed in **robot axes**, useful for understanding and managing the robot's movement.

Before jumping into programming, let's discuss your options for physically mounting the Hub on the robot. In general, the Hub's mounting can be considered **Orthogonal** or **Non-Orthogonal**.

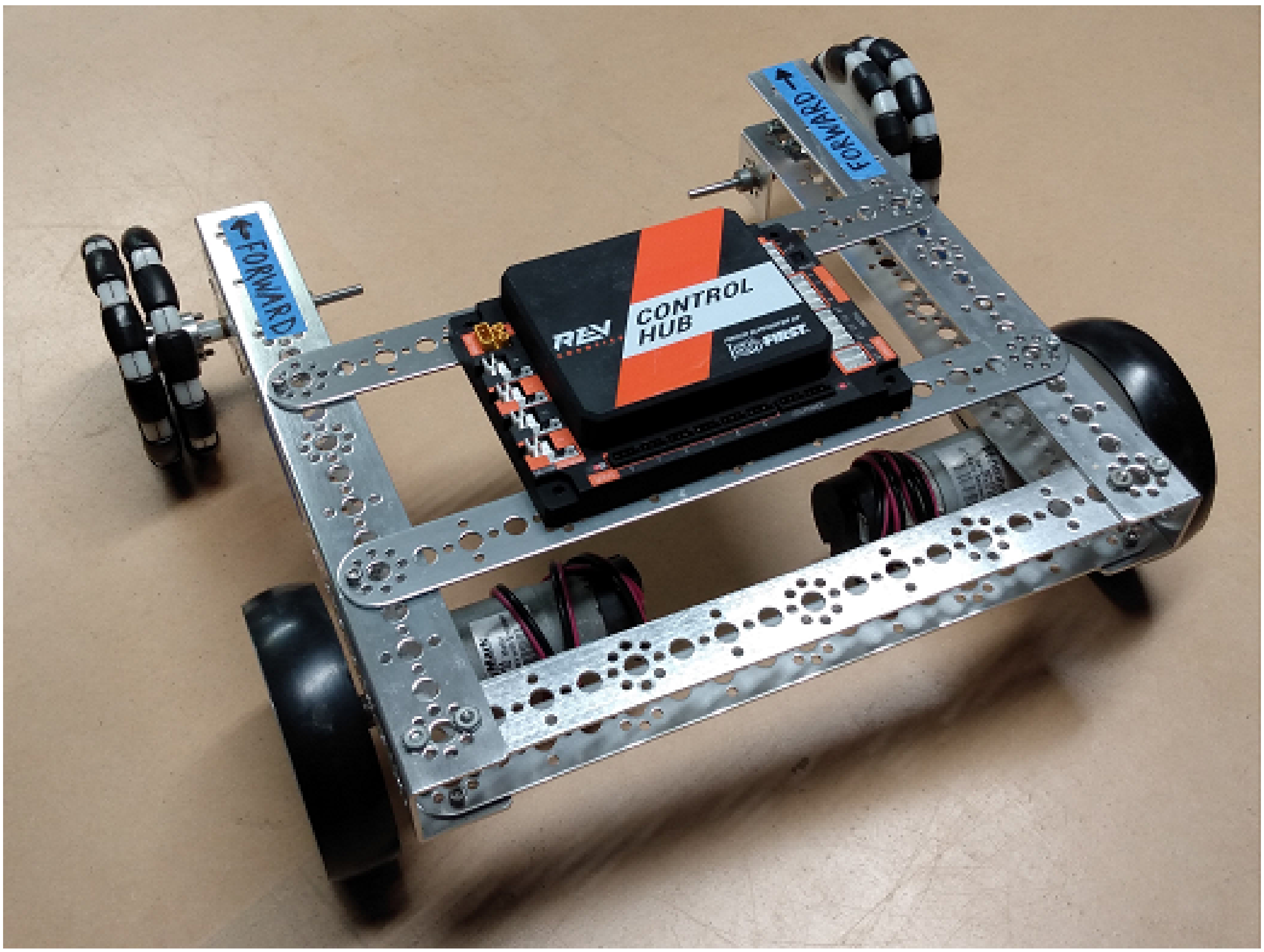
Orthogonal Mounting

Imagine a **cube** anywhere on your robot, parallel to the floor, with one flat side facing exactly towards the designated "front" of your robot. Place your Hub on one of these cube faces, with the Hub's straight edges **parallel** to the cube.

If that describes the orientation of your Hub, use the **Orthogonal** method of specifying its orientation. See the IMU Programming section below.

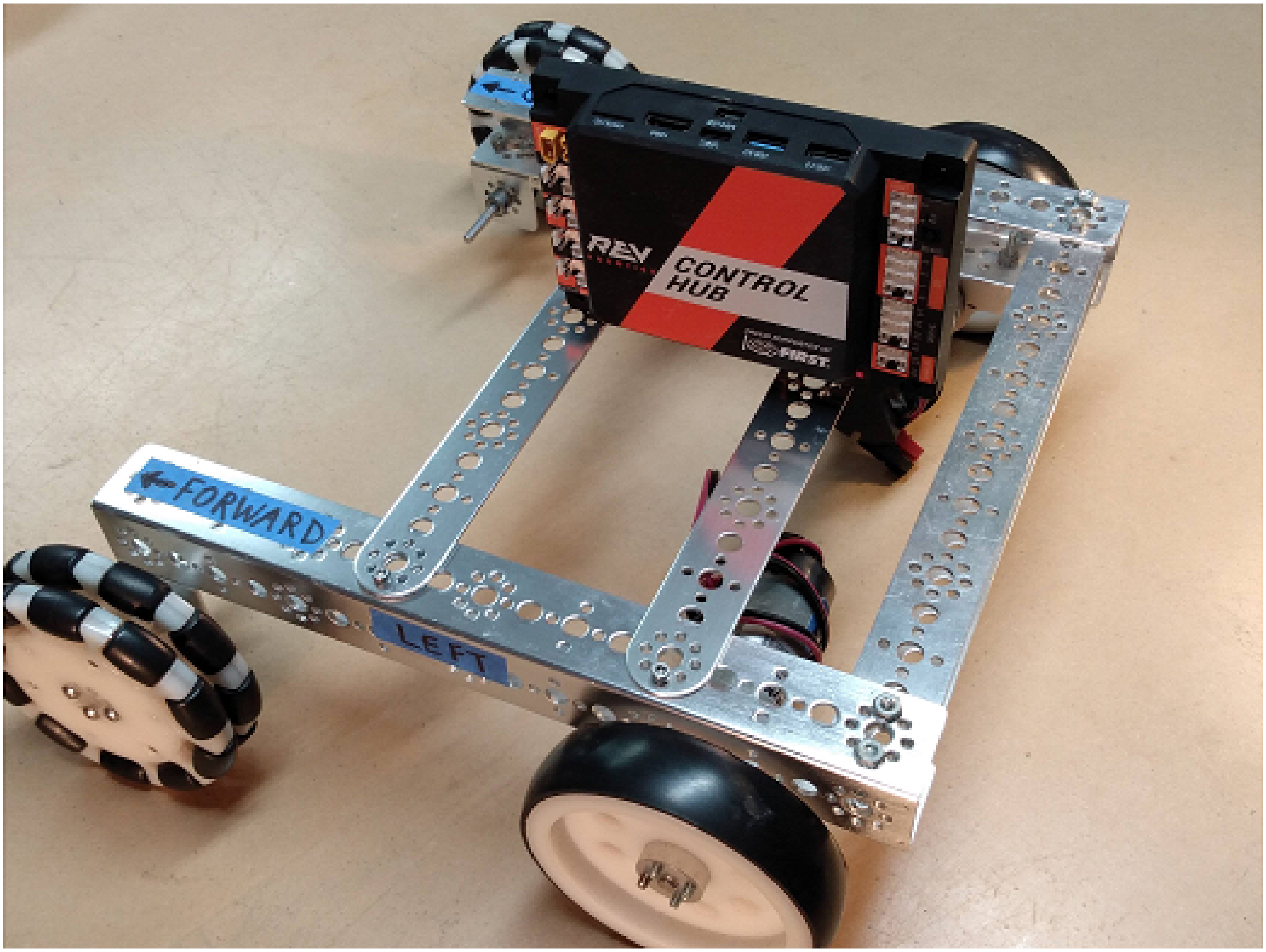
Here are some common examples:

Orthogonal #1



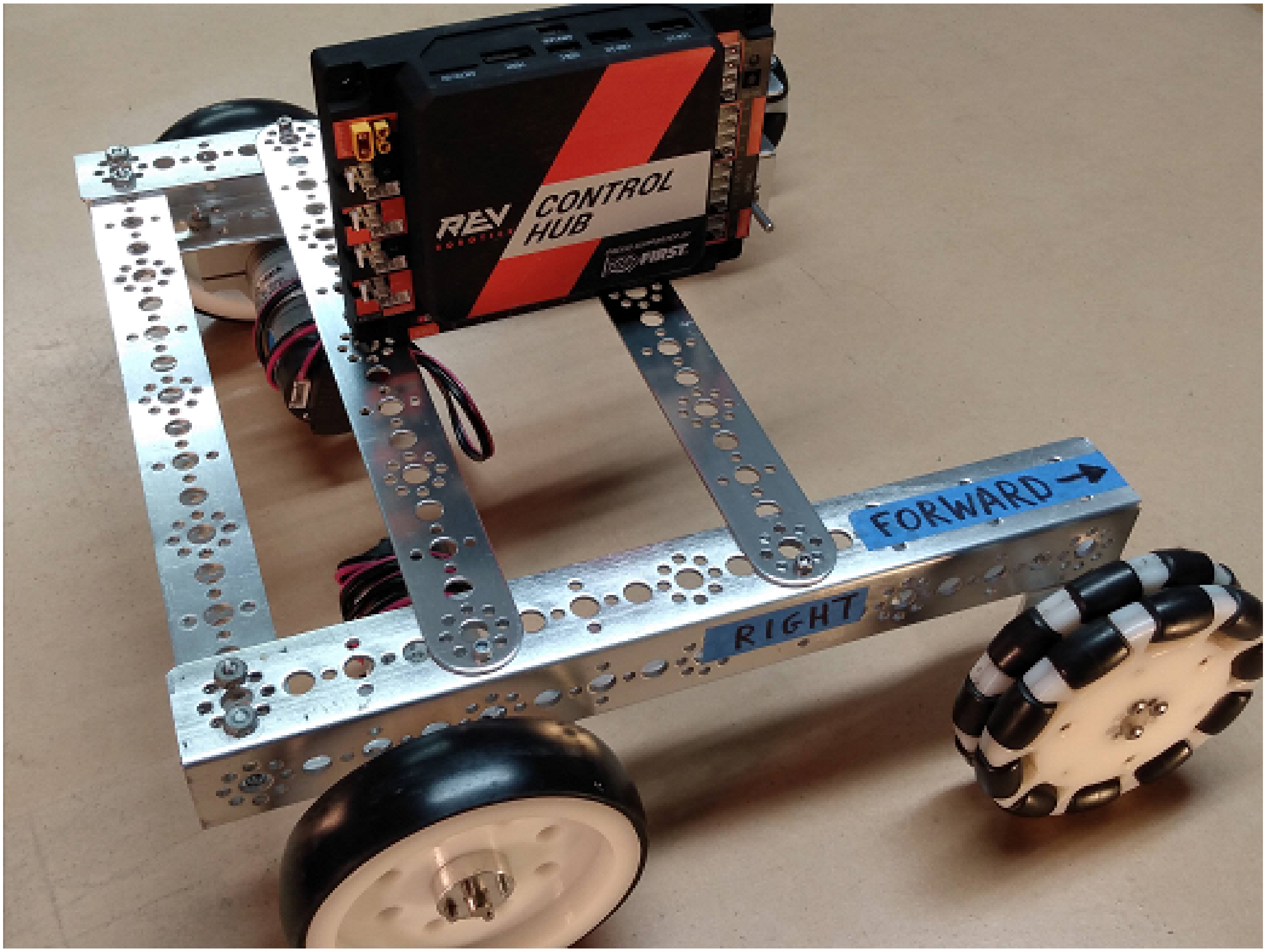
Logo UP, USB FORWARD

Orthogonal #2


REV
CONTROL HUB
FORWARD
LEFT

Logo LEFT, USB UP

Orthogonal #3



Logo RIGHT, USB UP

Orthogonal #4

Logo FORWARD, USB UP

Orthogonal #5

Logo BACKWARD, USB UP

Orthogonal #6

Logo DOWN, USB FORWARD

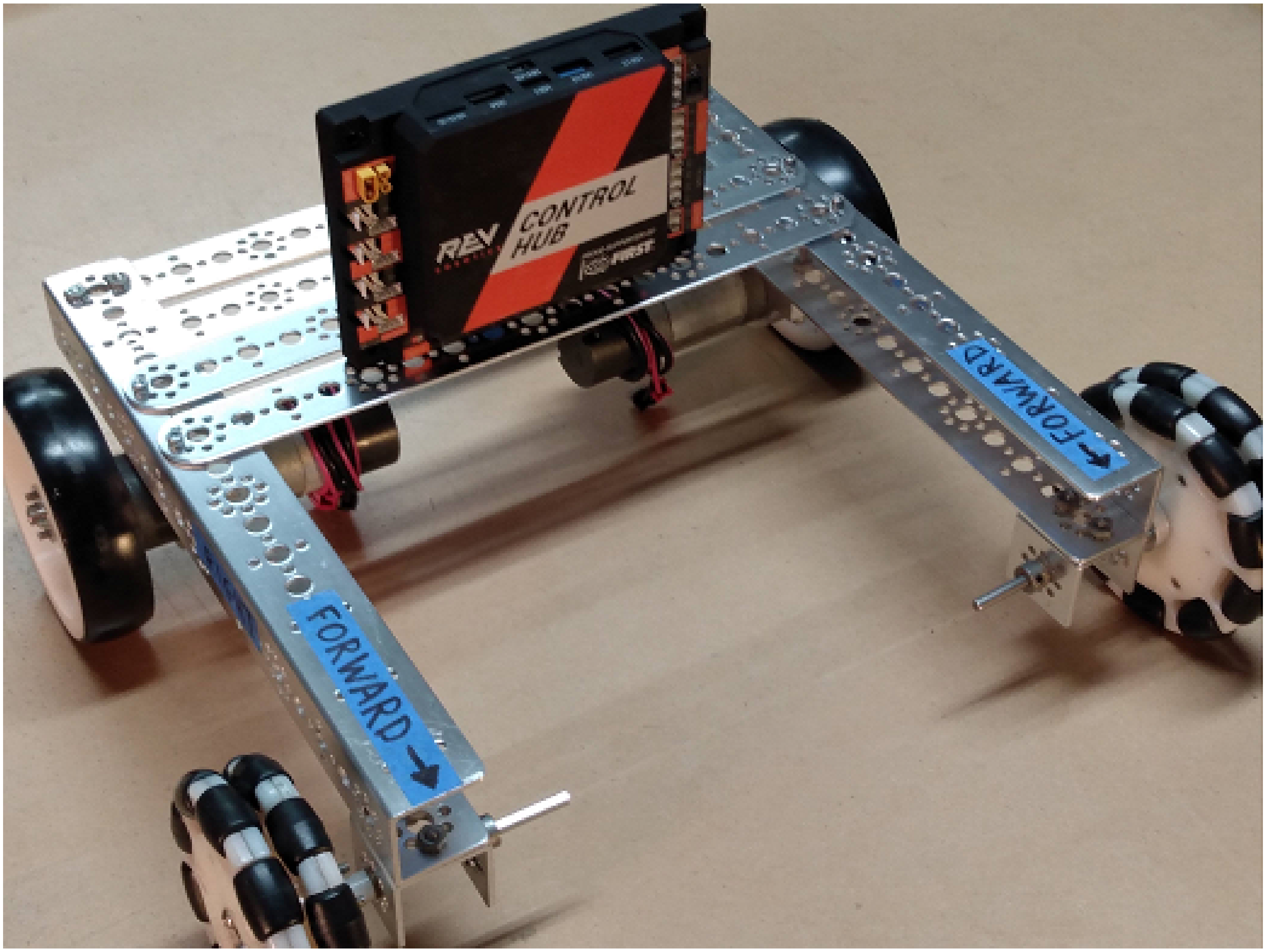
Orthogonal #7

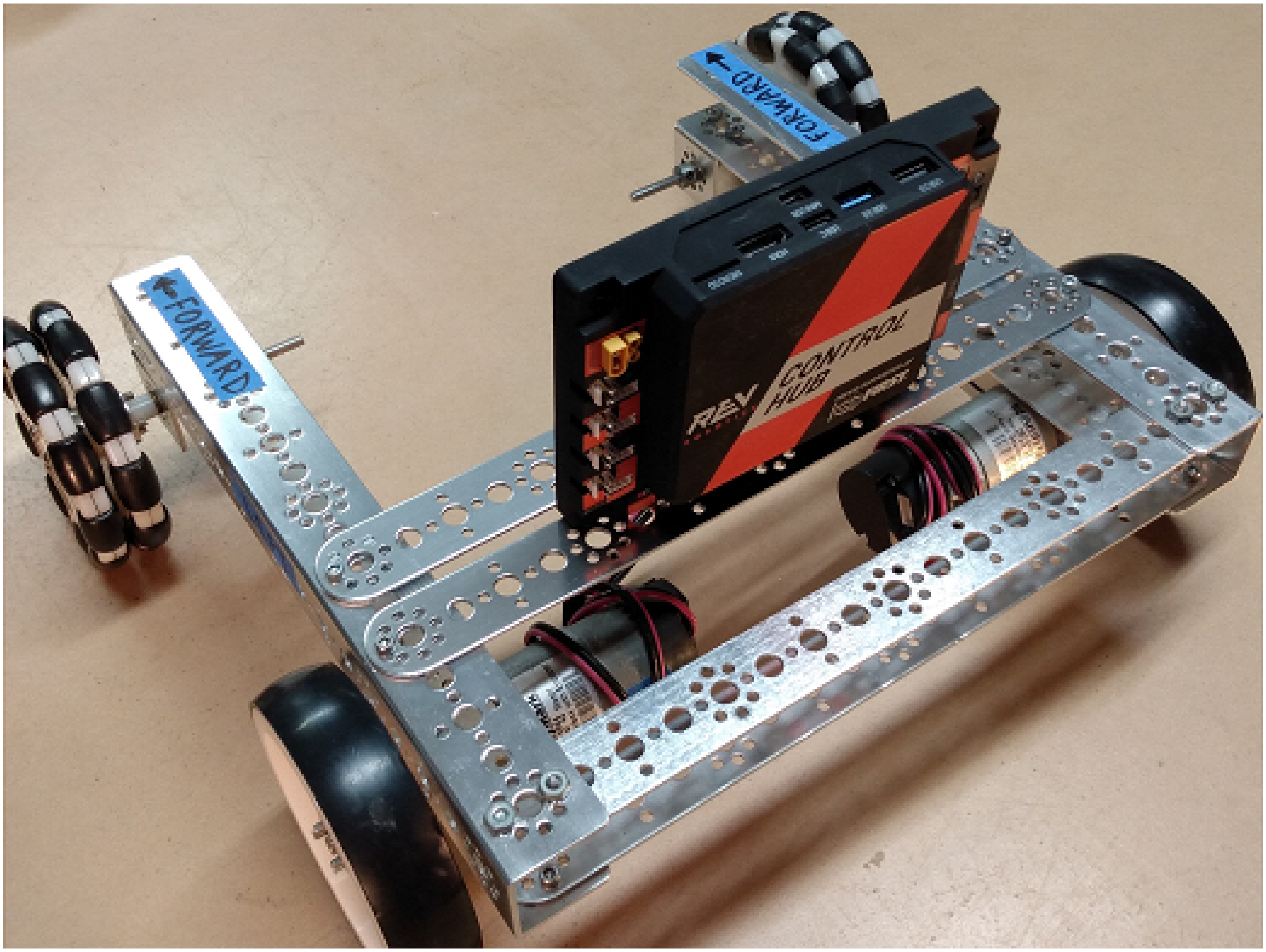
Logo FORWARD, USB LEFT

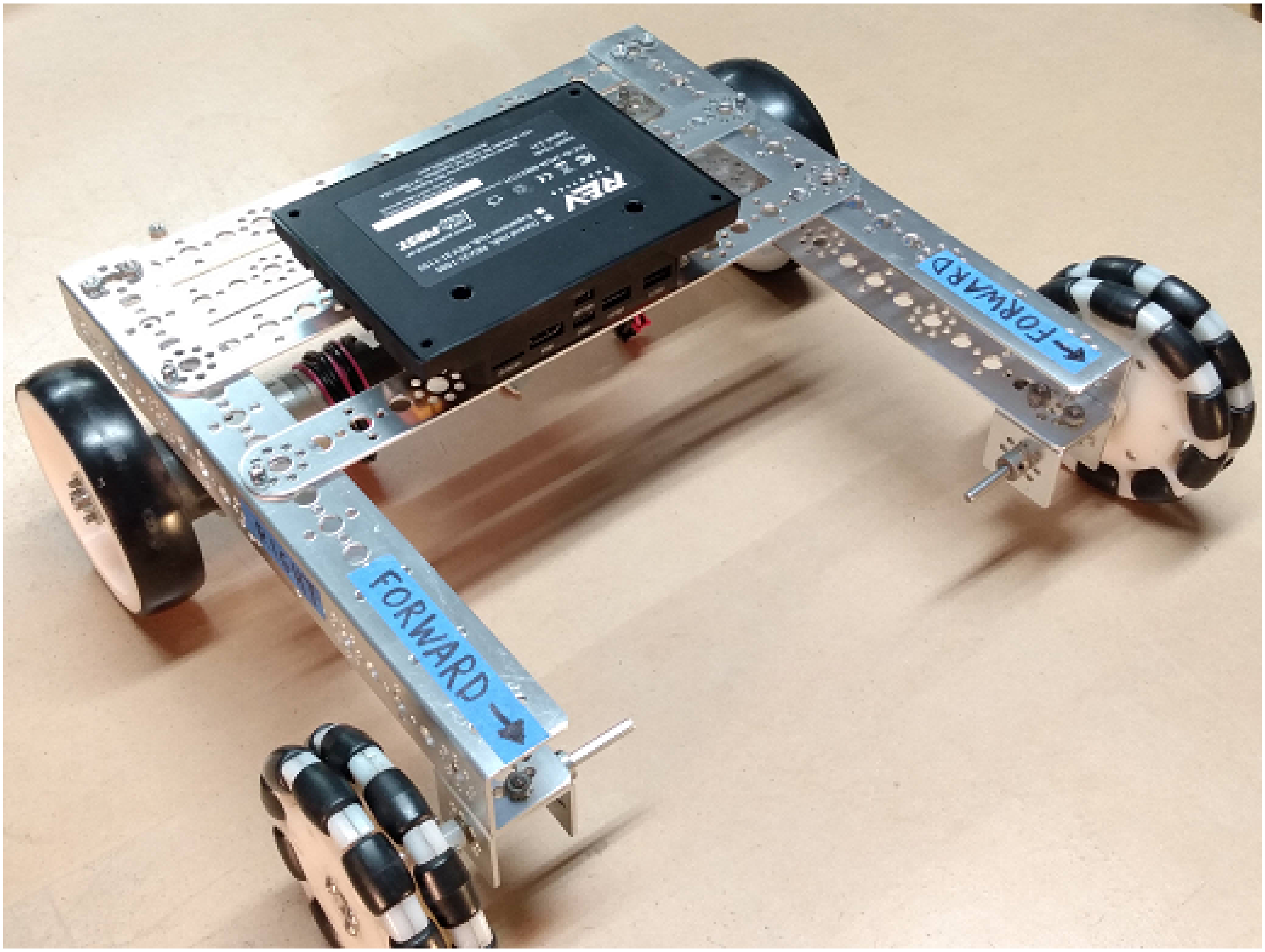
Orthogonal #8

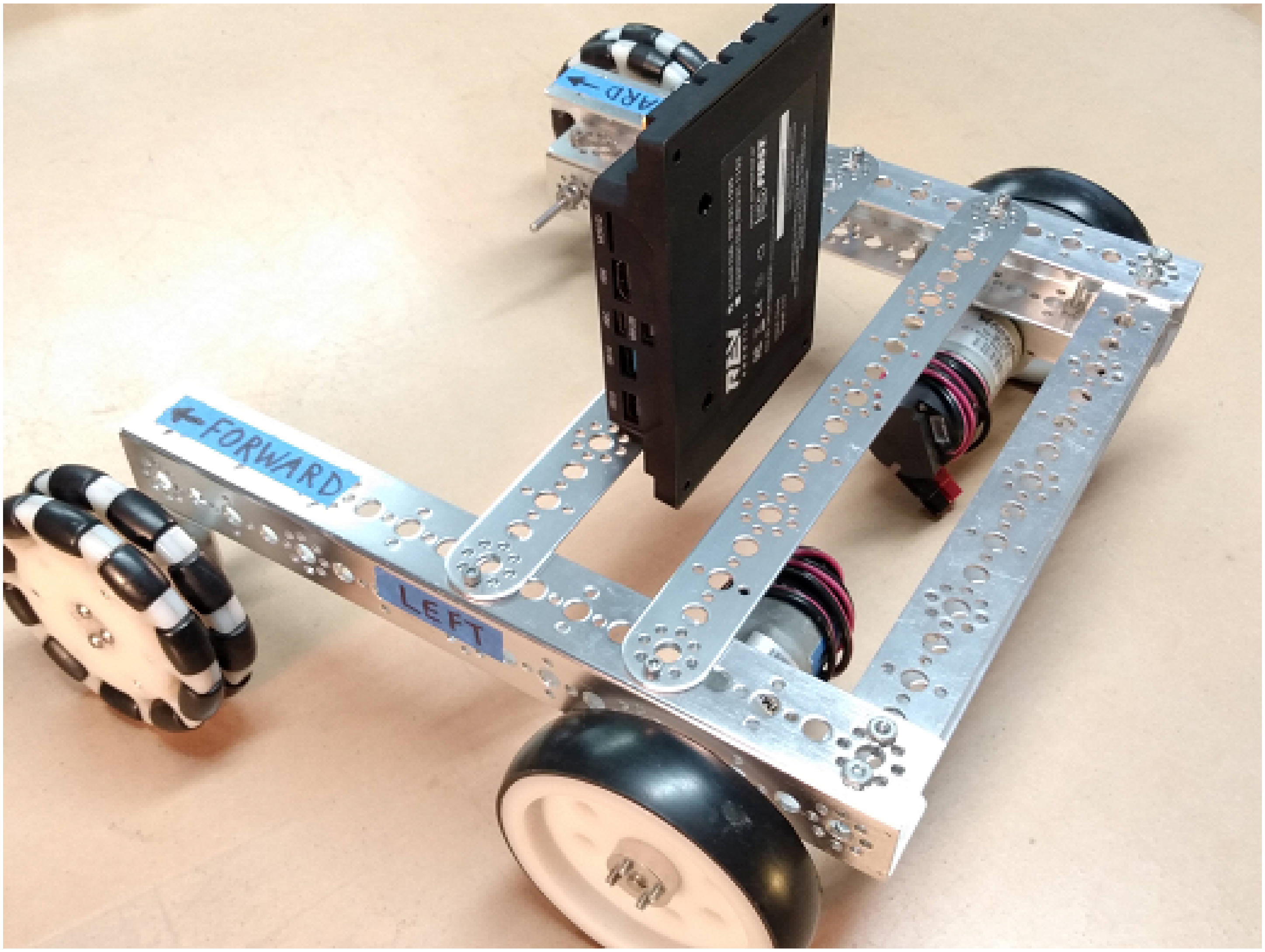
Logo FORWARD, USB RIGHT

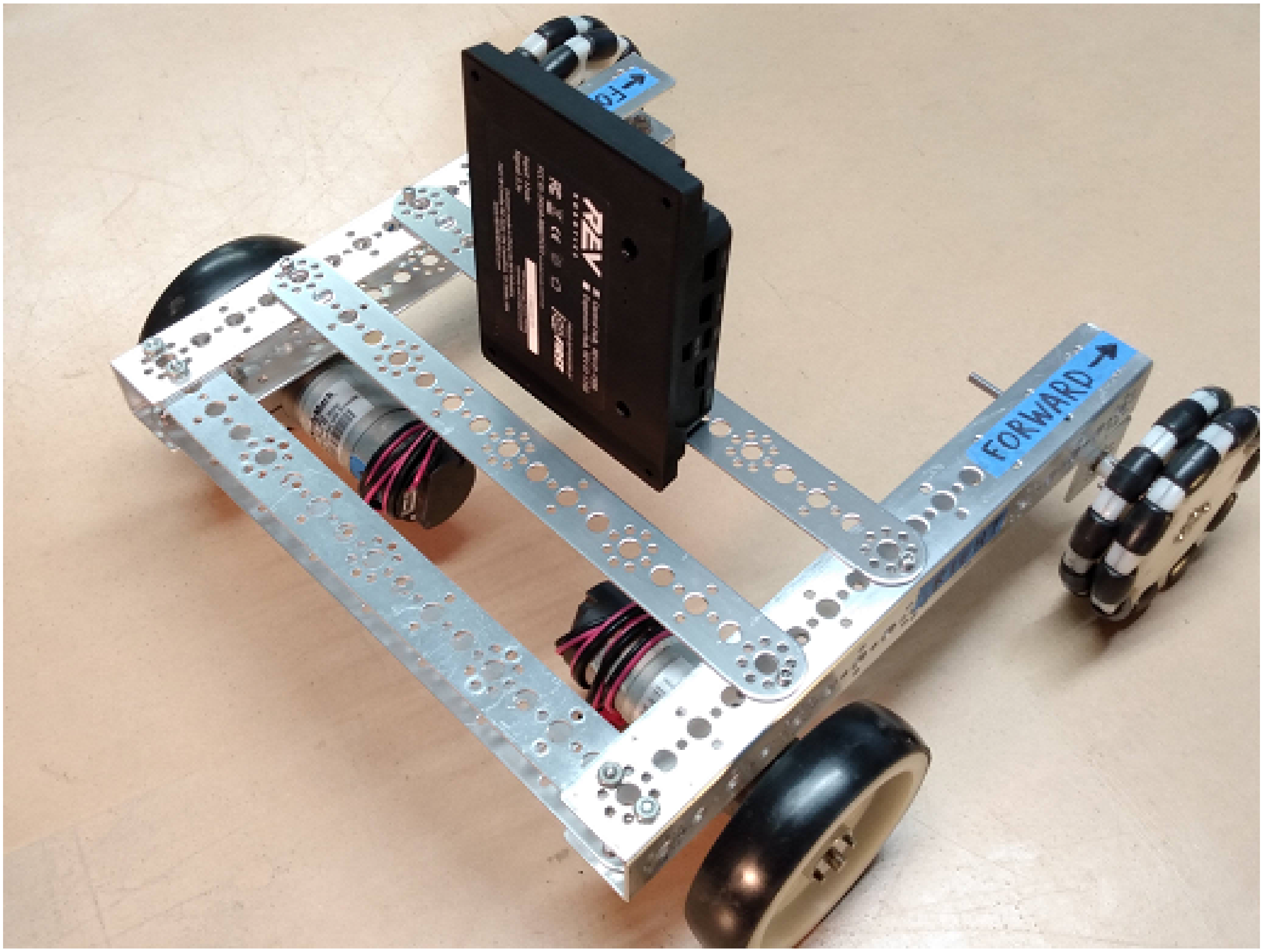
Orthogonal #9

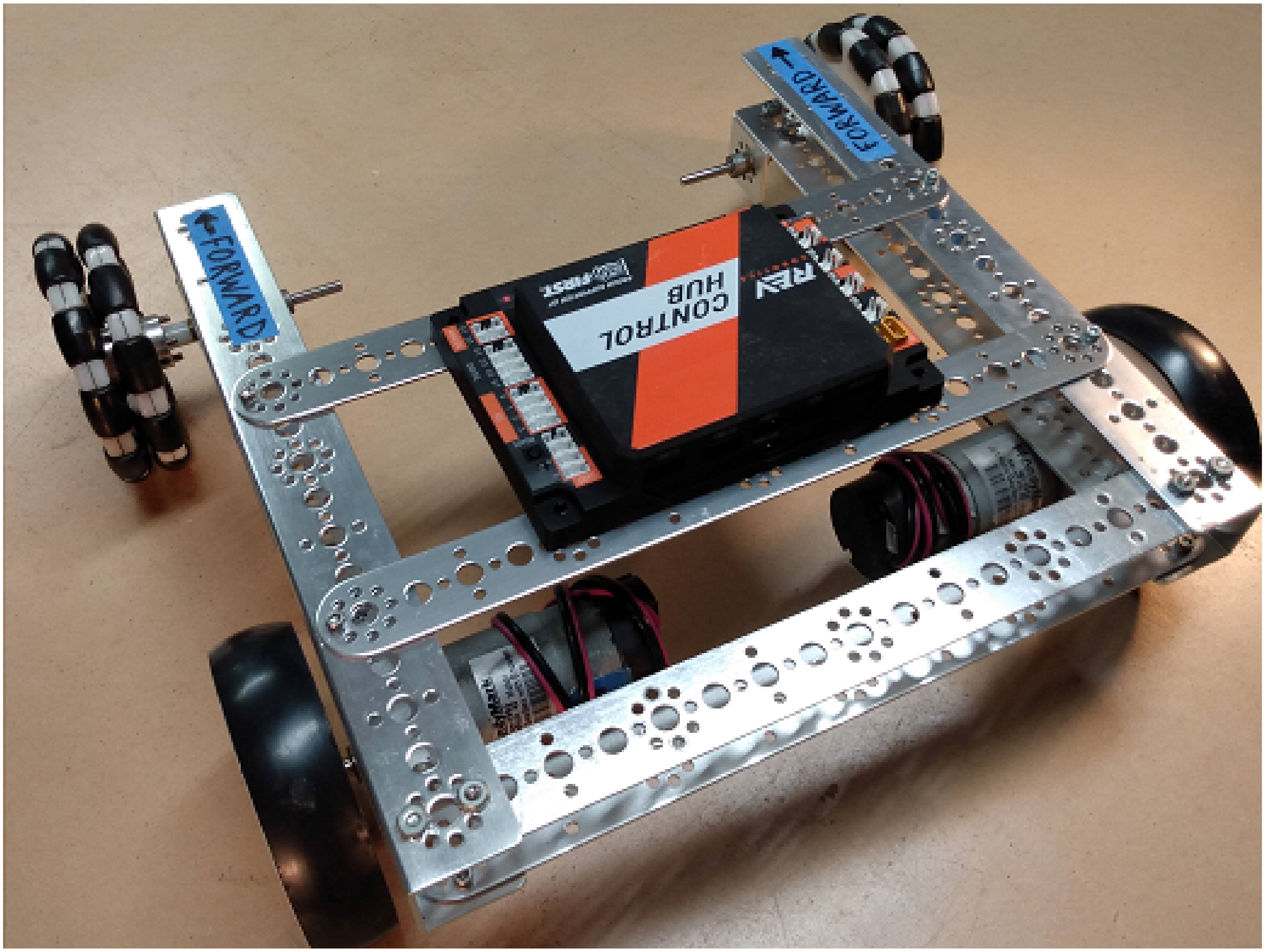

CONTROL HUB
REV
FORWARD
FORWARD


FORWARD
FORWARD
CONTROL HUB


FORWARD
FORWARD
REV


FORWARD
LEFT


FORWARD


FORWARD
FORWARD
CONTROL HUB
REV

Logo UP, USB BACKWARD

With six cube faces, and four 90-degree positions on each face, there are **24 possible Orthogonal orientations**.

Non-Orthogonal Mounting

Here are some scenarios, ranging from simple to complex:

- Imagine the same front-aligned cube, with your Hub on any face. The Hub's edges are **not parallel** to the cube. Namely, the Hub is rotated only **in-plane** (clockwise or counter-clockwise, looking at the REV logo).
- The Hub is mounted/tilted at some oblique angle from a face on the imaginary cube. At that single tilted angle, the Hub is not rotated in-plane (clockwise or counter-clockwise, looking at the logo).
- The Hub is tilted at multiple angles, with or without in-plane rotation.

For any Non-Orthogonal scenarios, SDK 8.1 provides **two ways** to describe the Hub's orientation. See below for the **Angles** method and the **Quaternion** method.

IMU Programming

SDK 8.1 offers new classes and methods that apply **universally** to both types of IMU. Once configured, the IMU type will not affect your programming. The programming steps include:

- set the IMU **parameters**, or use defaults
- **initialize** the IMU
- **read values** from the IMU, use as needed to control the robot
- optional: **reset Heading** one or more times

The following sections cover these topics in order.

Parameters

There are **three ways** to describe the Hub's orientation, using IMU parameters. One is for Orthogonal mounting, and two are for Non-Orthogonal mounting. Choose the simplest method that applies to your robot.

As an example, in the *FIRST* Tech Challenge Blocks menu under `Sensors` and `IMU`, you can find these three methods for specifying parameters:

Parameters for Method 1, Orthogonal

Method 1 consists of supplying a simple Orthogonal configuration. This requires you to determine the direction that the REV logo is facing. To do this, consider the Hub is mounted on an imaginary cube aligned to the "front" of the robot. Specify the Hub's mounting face: "Forward" means robot forward (front face), "Left" means robot left, etc.

Next, choose how the Hub is rotated on that face. Use the USB ports at the "top" of the Hub to determine this direction; assume you are at the rear of the robot, looking "forward".

Note: *Certain combinations are physically impossible. For example, if the REV logo is facing UP, the USB ports cannot also be facing UP. The OpMode will reject such combinations during IMU initialization.*

It's optional to save the parameters to a new variable called, for example, "myIMUparameters". That variable can be used in the next step (IMU initialization).

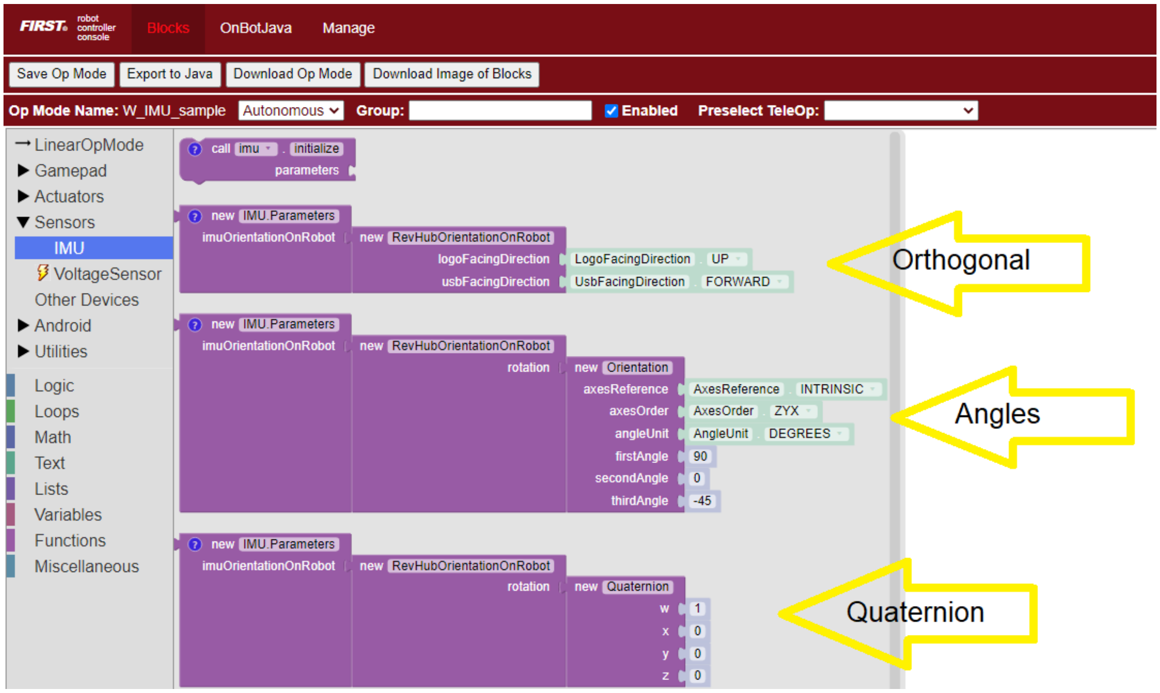


Fig. 247: Sample Blocks screenshot, demonstrating the three parameter methods

Blocks

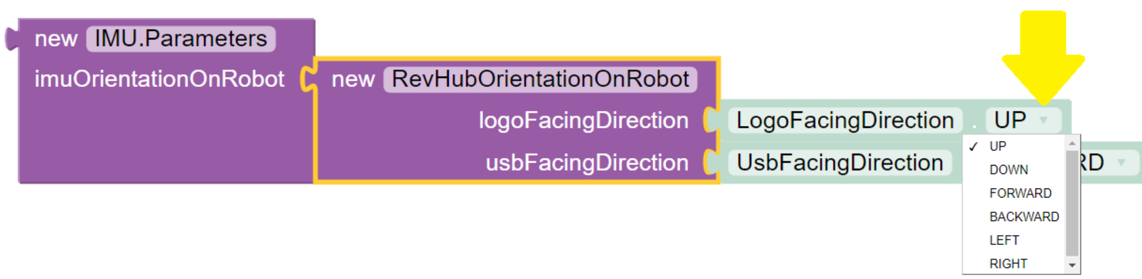


Fig. 248: Specifying Logo Facing Direction

Java

```
IMU.Parameters myIMUparameters;

myIMUparameters = new IMU.Parameters(
     new RevHubOrientationOnRobot(
          RevHubOrientationOnRobot.LogoFacingDirection.UP,
          RevHubOrientationOnRobot.UsbFacingDirection.FORWARD
     )
);
```

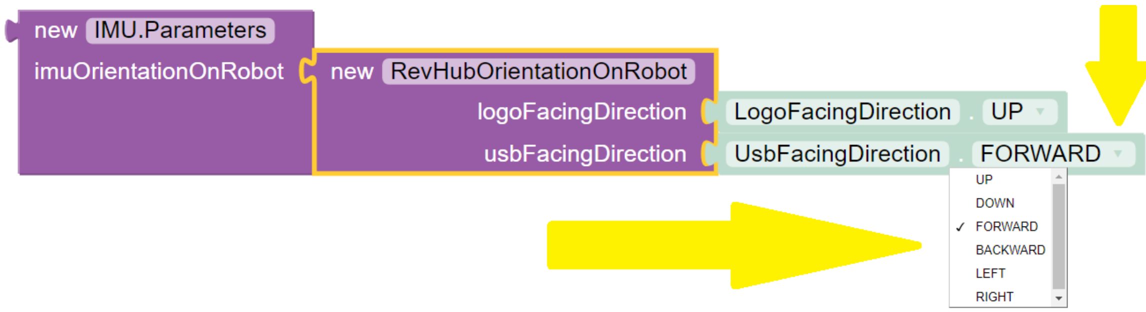


Fig. 249: Specifying USB Facing Direction

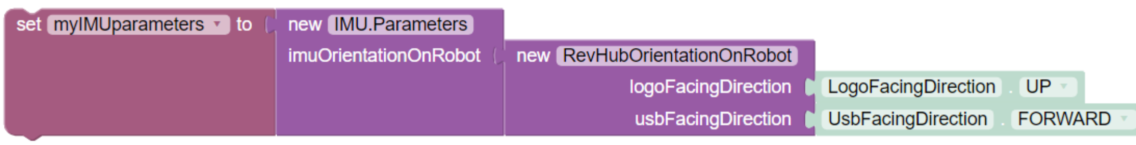


Fig. 250: Setting parameters to a Variable

Hub Axes for Setting Parameters

Only for the next two Parameters sections (Angles and Quaternion), we must temporarily use **Hub axes** instead of Robot axes. Hub axes are also at 90 degrees to each other, with origin inside the Hub.

The assumed initial Hub position is REV logo facing UP (Robot +Z), with USB ports FORWARD (Robot +Y). For the Angles and Quaternion methods, all rotations start here.

Again, "forward" is based on your team's agreed definition.

In this starting orientation, the Hub axes are **aligned with** the Robot Coordinate System:

- Heading, or Yaw, is the measure of rotation about the Z axis, which points upwards through the Hub's front plate or logo.
- Pitch is the measure of rotation about the X axis, which points toward the right-side I2C sensor ports.
- Roll is the measure about the Y axis, which points toward the top-edge USB port(s).

Hub rotations also follow the right-hand rule.

The legacy `BNO055IMU` driver uses **different Hub axes**: its X axis pointed to the USB port, and Y axis pointed to the left-side motor ports. The new SDK 8.1 universal IMU driver uses the above Hub axes for BNO055 and BHI260AP.

Parameters for Method 2, Angles

If your Hub is **not** mounted Orthogonally, you can specify the Hub's *rotation* about one or more **Hub axes** X, Y, Z. These are expressed in *degrees*, and the **order** in which the rotations are applied (it matters!).

The Blocks IMU palette contains a Block with default parameters for the Angles method of describing the Hub's orientation on the robot. Let's review this Blocks palette function now, as a good example. The Java API closely resembles the Blocks method.

The second listed default is ZYX, meaning you will provide the Hub's rotations in that order. Thus the "first angle" is the Z axis, the "second angle" is the Y axis, and the "third angle" is the X axis.

So the Hub will be rotated as follows: +90 degrees about **Z**, no rotation about **Y**, then -45 degrees about **X** (in its new direction).

For the Angles method, the assumed initial Hub position is REV Logo facing UP, with USB ports facing FORWARD. Additional rotations begin at this orientation.

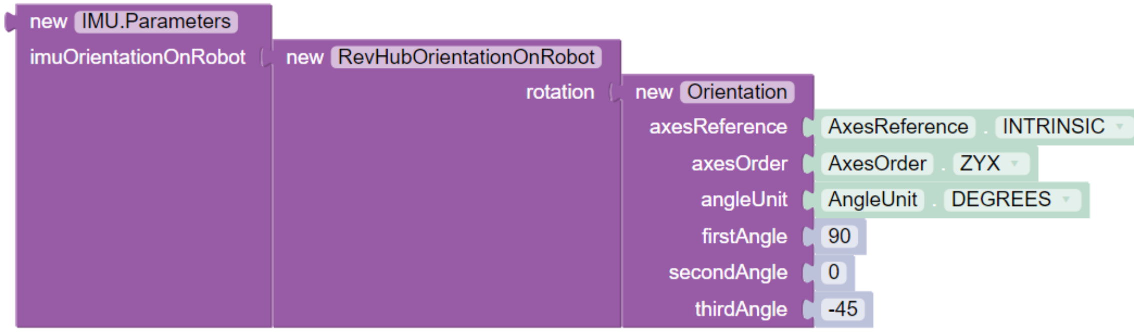


Fig. 251: Sample Block demonstrating angles method

1. From logo-up/USB-forward, this example starts with a "first angle" rotation of **+90 degrees about the Z axis**. Namely, the Hub rotates counter-clockwise (CCW), ending with the USB ports pointing to the robot's left side. Note the **X and Y axes have also rotated CCW**, since they are INTRINSIC (described below).
2. The "second angle" rotation is **0 degrees, no action**.
3. The "third angle" rotation is **-45 degrees about the Hub's X axis**, which **now points in the robot's forward direction** (after the first-angle Z rotation). So, the top edge of the Hub tilts downward, causing the USB ports to angle downward at 45 degrees, at the robot's left side.

Here's the full sequence:

Angles Rotation Step #1

Starting Position

Angles Rotation Step #2

First Angle (Z axis +90)

Angles Rotation Step #3

Third Angle (X axis -45)

The remaining default parameters don't need attention or editing. The third listed default is simply DEGREES, easy to work with. The first listed default is INTRINSIC axes reference, which means that the Hub axes move with each rotation of the Hub. (The other choice, rarely used, is EXTRINSIC for global axes that **don't move** with each Hub rotation.)

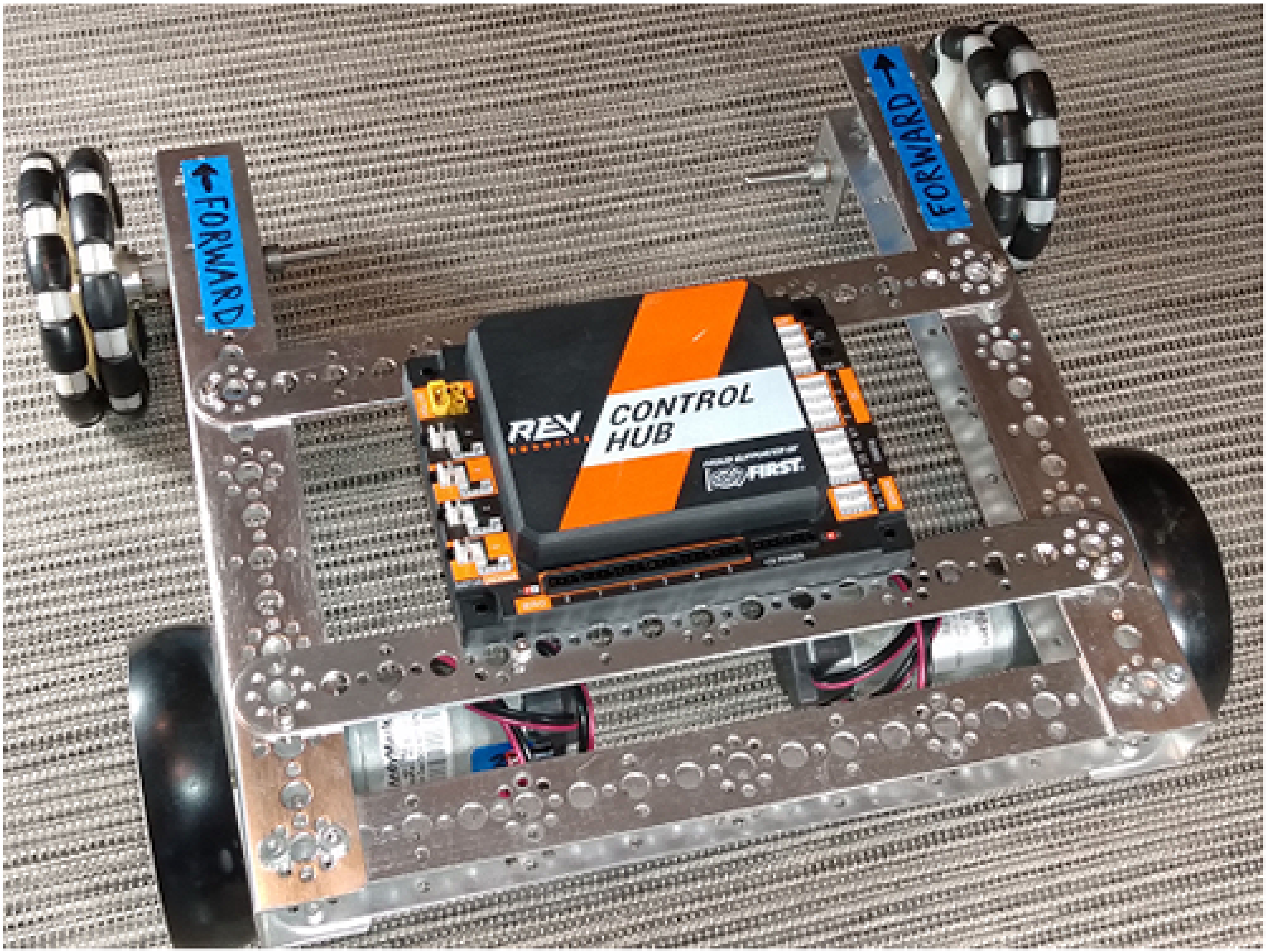
As with Orthogonal, it's optional to save the parameters to a new variable called, for example, "myIMUparameters". That variable can be used in the next step (IMU initialization).

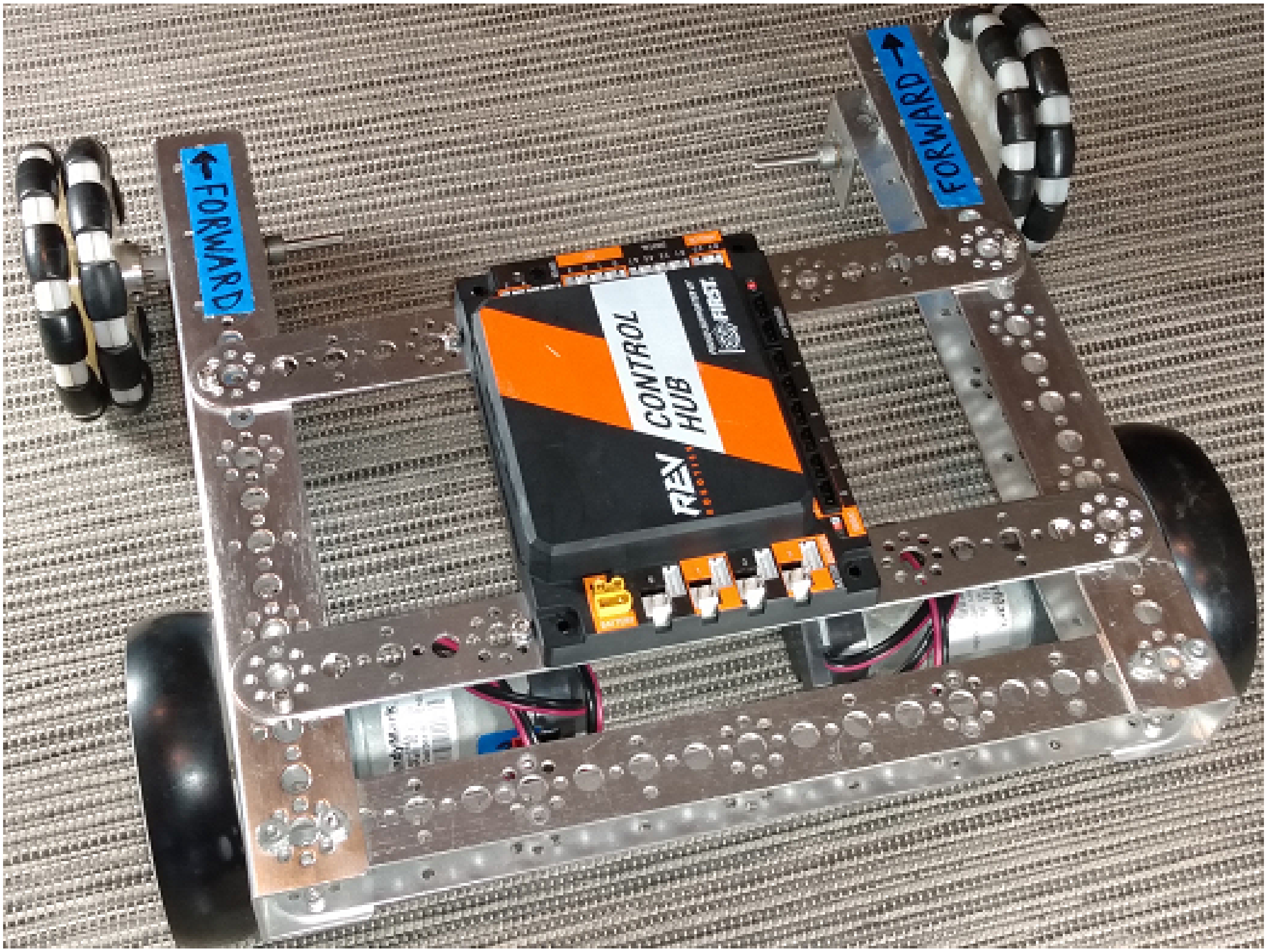
Blocks

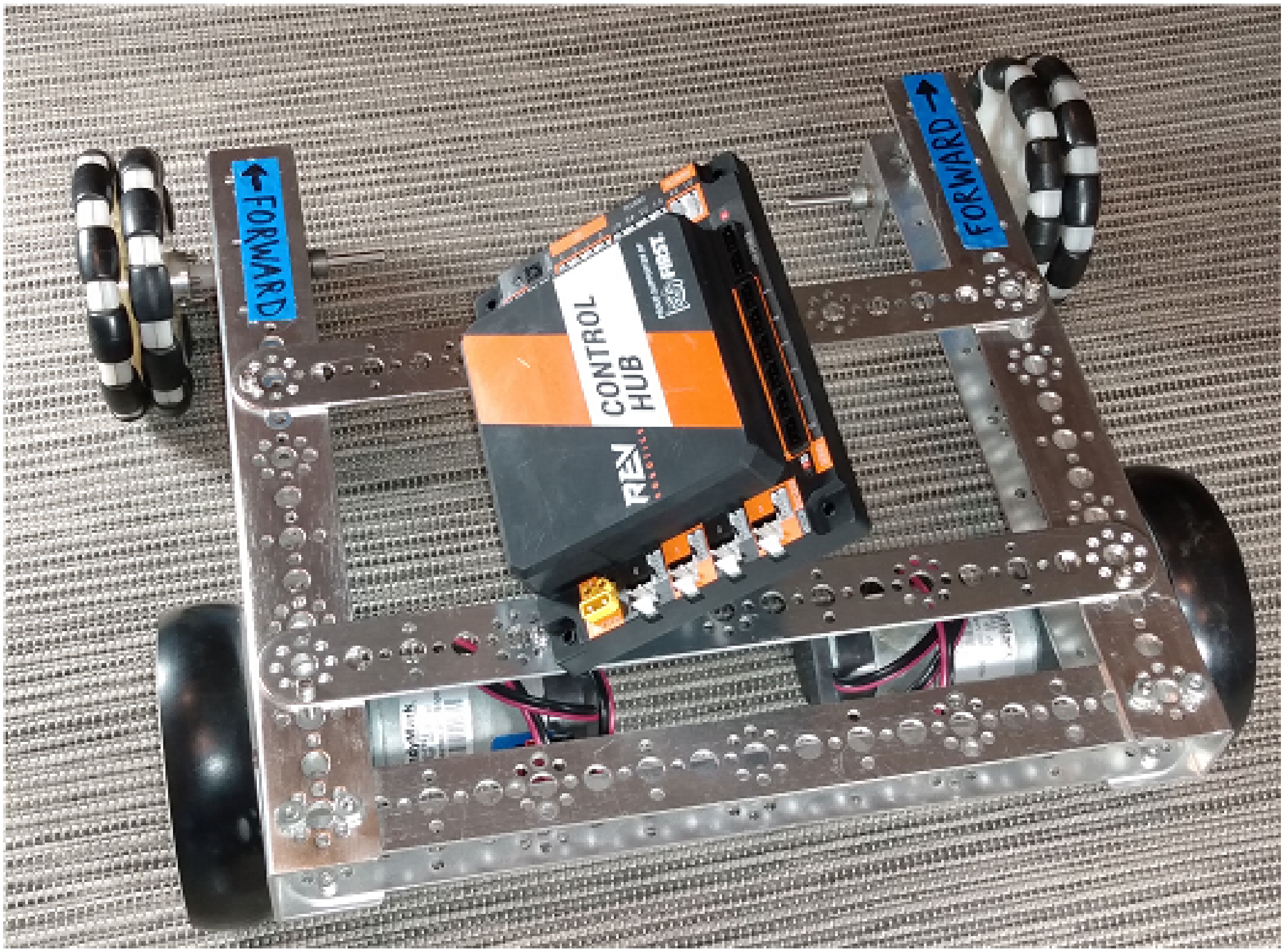
Java

```
IMU.Parameters myIMUparameters;

myIMUparameters = new IMU.Parameters(
     new RevHubOrientationOnRobot(
          new Orientation(
               AxesReference.INTRINSIC,
               AxesOrder.ZYX,
               AngleUnit.DEGREES,
               90,
               0,
               -45,
```

(continues on next page)


FORWARD
FORWARD
REV
CONTROL HUB
FIRST


FORWARD
FORWARD
CONTROL HUB
REV



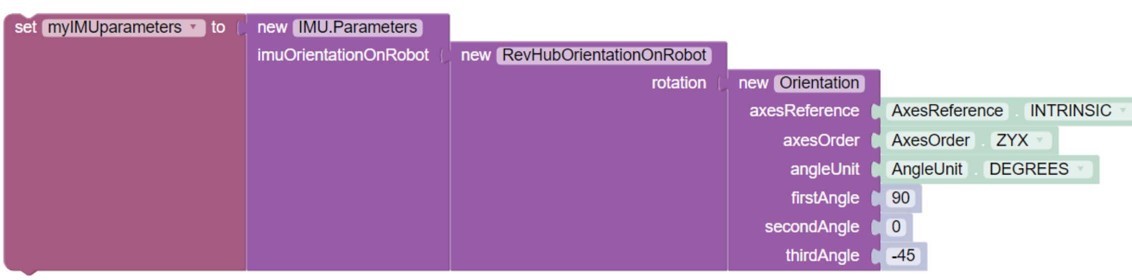


Fig. 252: Setting Angles in Blocks

(continued from previous page)

```
                0  // acquisitionTime, not used
          )
     )
);
```

Parameters for Method 3, Quaternion

As an alternative to the Angles method, the Hub's non-orthogonal orientation can be described using a Quaternion, an advanced math technique for describing **any** combination of tilting and rotating.

The following default Quaternion (w=1, x=0, y=0, z=0) describes a Hub in the assumed starting position: Logo facing UP, and USB ports FORWARD. Namely, no rotations.

Blocks

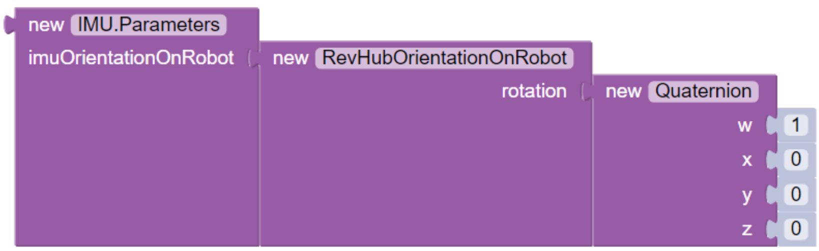


Fig. 253: Default Quaternion (no rotation)

Java

```
IMU.Parameters myIMUparameters;

// Default Quaternion
myIMUparameters = new IMU.Parameters(
     new RevHubOrientationOnRobot(
          new Quaternion(
               1.0f, // w
               0.0f, // x
               0.0f, // y
               0.0f, // z
               0     // acquisitionTime
          )
     )
);

// Or, consider a single rotation of +30 degrees
// about the X axis. Namely, the Hub's USB ports
// tilt 30 degrees upwards from the default starting
// position.
myIMUparameters = new IMU.Parameters(
     new RevHubOrientationOnRobot(
          new Quaternion(
               0.9659258f, // w
               0.258819f,  // x
```

(continues on next page)

(continued from previous page)

```
                0.0f,       // y
                0.0f,       // z
                0           // acquisitionTime
            )
        )
);
```

This basic tutorial does not cover the math behind Quaternions, an advanced substitute for Euler Angles described above. The SDK 8.1 IMU interface supports the use of Quaternions, for teams and third party libraries familiar with them.

Initialize IMU

This prepares the IMU for operation, using the parameters you defined.

In Blocks, use the first Block shown in the IMU palette, called `imu.initialize`. Most teams do this during the INIT phase of their OpMode, before `waitForStart()`.

The IMU should be motionless during its initialization process. The OpMode will continue when initialization is complete.

Note: Fun fact: Under the legacy `BNO055IMU` interface, intialization takes about 900 milliseconds. Under the new universal IMU interface, the BNO055 takes about 100 milliseconds, while the BHI260AP takes about 50 milliseconds.

For **any of the three methods** (Orthogonal, Angles, Quaternion), initialize with the IMU parameters from the new Block, or from your optional Variable.

Blocks

Two methods for Initializing the IMU:

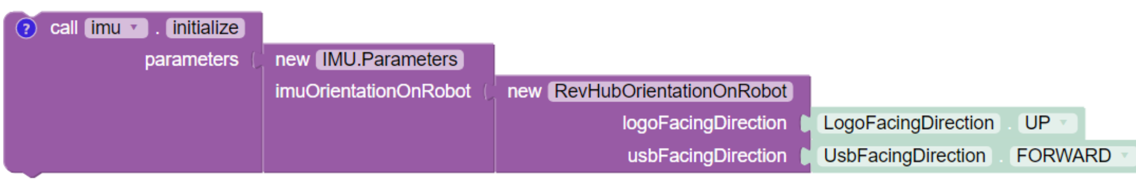


Fig. 254: Initializing the IMU directly

Fig. 255: Initializing the IMU using Parameters

Java

```java
// Two methods for Initializing the IMU:

// Initialize IMU directly
imu.initialize(
    new IMU.Parameters(
        new RevHubOrientationOnRobot(
            RevHubOrientationOnRobot.LogoFacingDirection.UP,
            RevHubOrientationOnRobot.UsbFacingDirection.FORWARD
        )
    )
);

// Initialize IMU using Parameters
imu.initialize(myIMUparameters);
```

Read IMU Angles - Basic

Now you can read the IMU values for **robot orientation**, expressed as Heading (Yaw or Z-angle), Pitch (X-angle) and Roll (Y-angle). You have no concern now about the Hub's orientation or mounting – that has been defined with parameters, and the SDK is ready to provide actual data about the robot, using the robot's axes.

Note: *Reminder: Robot Z points upwards to the ceiling. Robot Y points forward – whatever you decide is "forward" on your robot (which could be round!). Robot X points to the right side of the robot. Robot rotations follow the right-hand rule.*

For all axes, IMU angles are provided in the range of **-180 to +180 degrees** (or from `-Pi` to `+Pi` radians). If you are working with values that might cross the +/- 180-degree transition, handle this with your programming. That's beyond the scope of this IMU tutorial.

Here's an example of reading IMU Angles:

Blocks

In Blocks, create a new Variable to receive data from this green Block in the **IMU** palette:

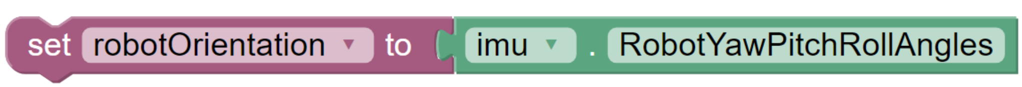


Fig. 256: Get Yaw-Pitch-Roll Angles

From the **YawPitchRollAngles** palette under **Utilities**, use the green Blocks to read each angle from the Variable you just created.

These Blocks are used here in a Repeat Loop, to display the angles on the Driver Station:

These Blocks are shown in the Sample OpMode called `SensorIMU`.

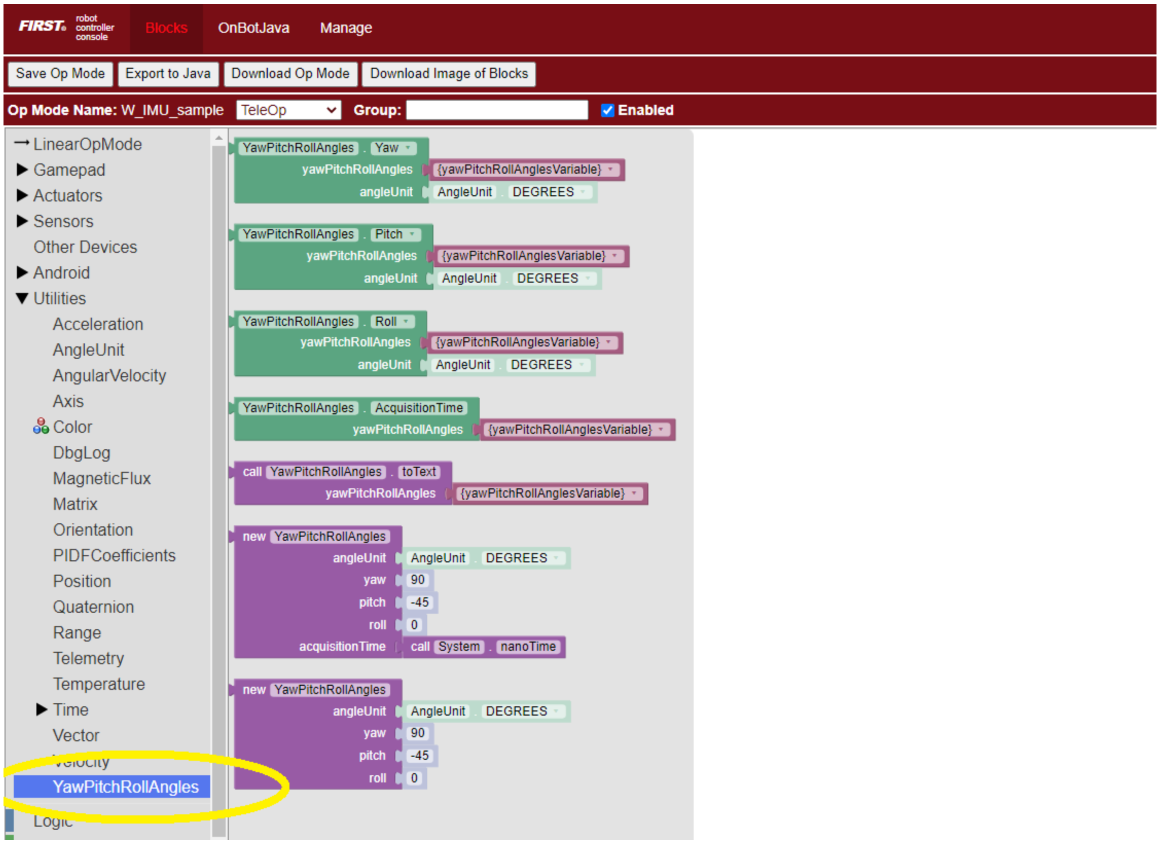


Fig. 257: Extract Angles

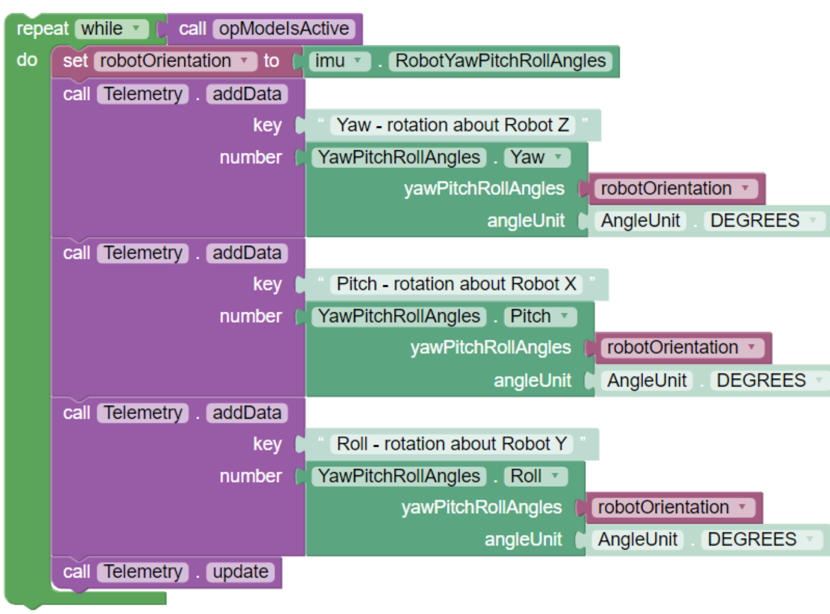


Fig. 258: Displaying Yaw-Pitch-Roll using Telemetry

Java

```
// Create an object to receive the IMU angles
YawPitchRollAngles robotOrientation;
robotOrientation = imu.getRobotYawPitchRollAngles();

// Now use these simple methods to extract each angle
// (Java type double) from the object you just created:
double Yaw   = robotOrientation.getYaw(AngleUnit.DEGREES);
double Pitch = robotOrientation.getPitch(AngleUnit.DEGREES);
double Roll  = robotOrientation.getRoll(AngleUnit.DEGREES);
```

Note that the robot's orientation is described here **intrinsically**; the axes move with each rotation. Here's an example from the Javadocs:

> As an example, if the yaw is 30 degrees, the pitch is 40 degrees, and the roll is 10 degrees, that means that you would reach the described orientation by first rotating a robot 30 degrees counter-clockwise from the starting point, with all wheels continuing to touch the ground (rotation around the Z axis). Then, you make your robot point 40 degrees upward (rotate it 40 degrees around the X axis). Because the X axis moved with the robot, the pitch is not affected by the yaw value. Then from that position, the robot is tilted 10 degrees to the right, around the newly positioned Y axis, to produce the actual position of the robot.

Again, the IMU ***output*** *results are given in the* ***Robot Coordinate System****, or Robot axes. Only for a non-Orthogonal orientation,* ***Hub axes*** *were used temporarily for* ***input*** *parameters, describing the Hub's rotation to achieve its mounted orientation.*

Read IMU Angles - Flexible

As an alternative to the `YawPitchRollAngles` class, the SDK also provides a more flexible `Orientation` class. This allows you to specify a **custom order** of axis rotations, and a choice of intrinsic or extrinsic axes.

Again, IMU angles are provided in the range of -180 to +180 degrees (or from `-Pi` *to* `+Pi` *radians).*

Here is an example use of these functions:

Blocks

As before, first create an object (Blocks Variable) containing the array of orientation values (from the Blocks `Sensors / IMU` palette):

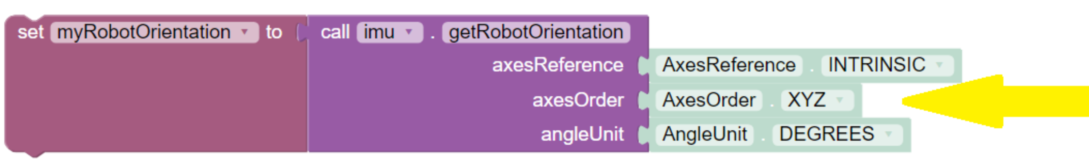


Fig. 259: Get Robot Orientation

Notice the **axes order of XYZ**, different than the ZXY order used by the `YawPitchRollAngles` class.

Then extract the specific axis rotations you want, from the Blocks `Utilities / Orientation` palette:

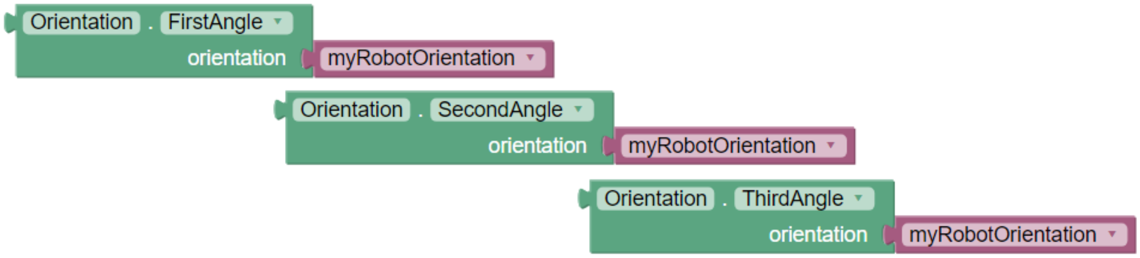


Fig. 260: Extract Orientation Angles

Java

```
// Create Orientation variable
Orientation myRobotOrientation;

// Get Robot Orientation
myRobotOrientation = imu.getRobotOrientation(
     AxesReference.INTRINSIC,
     AxesOrder.XYZ,
     AngleUnit.DEGREES
);

// Then read or display the desired values (Java type float):
float X_axis = myRobotOrientation.firstAngle;
float Y_axis = myRobotOrientation.secondAngle;
float Z_axis = myRobotOrientation.thirdAngle;
```

Note: *Pay close attention to the selection of* **axes order***, which greatly affects the IMU results. If you care mostly about Heading (Yaw), choose an axes order that starts with Z.*

Read Angular Velocity

The SDK also provides values for **angular velocity**, the rate of change (degrees or radians per second) for Roll, Pitch or Yaw.

Here is an example for reading Angular Velocity:

Blocks

As before, first create an object (Blocks Variable) containing the array of angular velocity values (from the Blocks `Sensors / IMU` palette):

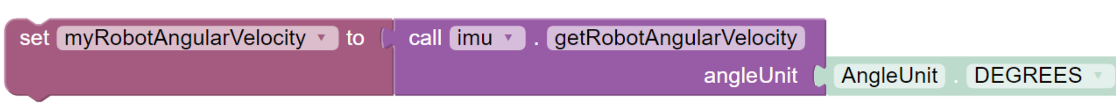


Fig. 261: Get Robot Angular Velocity

Then extract the specific axis rotations you want, from the Blocks `Utilities / AngularVelocity` palette:

These Blocks are shown in the Sample OpMode called `SensorIMU`.

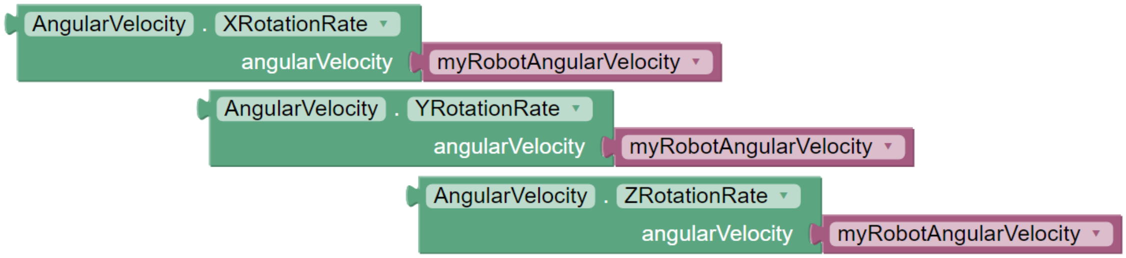


Fig. 262: Extract Rotation Rates

Java

```
// Create angular velocity array variable
AngularVelocity myRobotAngularVelocity;

// Read Angular Velocities
myRobotAngularVelocity = imu.getRobotAngularVelocity(AngleUnit.DEGREES);

// Then read or display these values (Java type float)
// from the object you just created:
float zRotationRate = myRobotAngularVelocity.zRotationRate;
float xRotationRate = myRobotAngularVelocity.xRotationRate;
float yRotationRate = myRobotAngularVelocity.yRotationRate;
```

These are also shown in each of the Java **Sample OpModes** listed in a section below.

Reset Heading

It can be useful to reset the Heading (or Yaw or Z-angle) to zero, at one or more places in your OpMode.

Here is an example for resetting the Yaw axis:

Blocks

In Blocks, this optional command is simple:

Fig. 263: Reset Yaw

Java

```java
// Reset Yaw
imu.resetYaw();
```

It's safest to reset Yaw only when the robot has not significantly deviated from a flat/horizontal orientation.

This command assumes the Hub's actual orientation was **correctly described** with Orthogonal, Angles or Quaternion parameters.

In other words, a non-Orthogonal Hub moved away from its parameter-defined orientation, may not give reliable results for Heading/Yaw or `resetYaw()`, even after the robot has returned to its original defined orientation.

An exception, or loophole, is that "reset" Heading/Yaw values might still be valid if the Hub is actually mounted in an incorrectly described Orthogonal orientation, and the robot remains level. This may benefit a rookie team that overlooked the IMU Parameters or moved the Hub to a different Orthogonal position, still relying only on Heading. This `resetYaw()` *exception does* **not** *apply to angular velocity for Yaw (Z-axis).*

Here's the official Javadocs description for `resetYaw()`:

> Resets the robot's yaw angle to 0. After calling this method, the reported orientation will be relative to the robot's position when this method is called, as if the robot was perfectly level right now. That is to say, the pitch and yaw will be ignored when this method is called.

The Javadocs' statement 'resets to 0' should be read in the context of the previous discussion. In certain off-axis Hub orientations, a reset Yaw value might not actually display as zero.

If `resetYaw()` does not meet your needs, other code-based choices (possibly less effective) include:

- 'Save & Subtract' to establish the current Heading as a new "zero" baseline for further navigation
- use the original Heading for the entire match, using only absolute (global) targets

Important: *For all choices, be aware of "gyro drift". Most electronic IMUs give slowly shifting Z-angle results over time, for various reasons. Although the Pitch and Roll axes can use* **gravity's direction** *to correct for drift, Yaw (Heading or Z-angle) cannot.*

Sample OpModes

SDK 8.1 and newer contains Sample OpModes demonstrating the above.

Blocks

In Blocks, a simple example is called `SensorIMU`.

Here's an `image` and the `Blocks file` of this Sample OpMode.

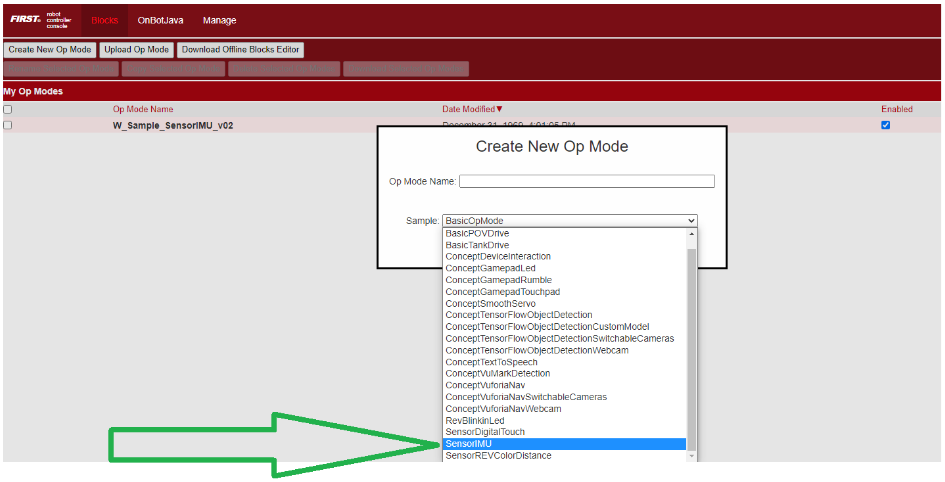


Fig. 264: Blocks IMU Sample

Java

In Java, three Sample OpModes demonstrate the new universal IMU interface:

ConceptExploringIMUOrientation.java

Provides a tool to experiment with setting your Hub orientation on the robot

`ConceptExploringIMUOrientation.java`

```
/*
Copyright (c) 2022 REV Robotics, FIRST
All rights reserved.
Redistribution and use in source and binary forms, with or without modification,
are permitted (subject to the limitations in the disclaimer below) provided that
the following conditions are met:
Redistributions of source code must retain the above copyright notice, this list
of conditions and the following disclaimer.
Redistributions in binary form must reproduce the above copyright notice, this
list of conditions and the following disclaimer in the documentation and/or
other materials provided with the distribution.
Neither the name of REV Robotics nor the names of its contributors may be used to
endorse or promote products derived from this software without specific prior
written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED BY THIS
LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE
FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR
TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF
THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/
package org.firstinspires.ftc.robotcontroller.external.samples;

import com.qualcomm.hardware.rev.RevHubOrientationOnRobot;
```

(continues on next page)

(continued from previous page)

```
import com.qualcomm.robotcore.eventloop.opmode.Disabled;
import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.eventloop.opmode.TeleOp;
import com.qualcomm.robotcore.hardware.IMU;

import org.firstinspires.ftc.robotcore.external.navigation.AngleUnit;
import org.firstinspires.ftc.robotcore.external.navigation.AngularVelocity;
import org.firstinspires.ftc.robotcore.external.navigation.YawPitchRollAngles;

/**
 * This file demonstrates the impact of setting the IMU orientation correctly or incorrectly. This
 * code assumes there is an IMU configured with the name "imu".
 * <p>
 * Note: This OpMode is more of a tool than a code sample. The User Interface portion of this code
 *       goes beyond simply showing how to interface to the IMU.<br>
 *       For a minimal example of interfacing to an IMU, please see the SensorIMUOrthogonal or␣
↪SensorIMUNonOrthogonal sample OpModes.
 * <p>
 * This sample enables you to re-specify the Hub Mounting orientation dynamically by using gamepad␣
↪controls.
 * While doing so, the sample will display how Pitch, Roll and Yaw angles change as the hub is␣
↪moved.
 * <p>
 * The gamepad controls let you change the two parameters that specify how the Control/Expansion␣
↪Hub is mounted. <br>
 * The first parameter specifies which direction the printed logo on the Hub is pointing. <br>
 * The second parameter specifies which direction the USB connector on the Hub is pointing. <br>
 * All directions are relative to the robot, and left/right is as viewed from behind the robot.
 * <p>
 * How will you know if you have chosen the correct Orientation? With the correct orientation
 * parameters selected, pitch/roll/yaw should act as follows:
 * <p>
 *   Pitch value should INCREASE as the robot is tipped UP at the front. (Rotation about X) <br>
 *   Roll value should INCREASE as the robot is tipped UP at the left side. (Rotation about Y) <br>
 *   Yaw value should INCREASE as the robot is rotated Counter Clockwise. (Rotation about Z) <br>
 * <p>
 * The Yaw can be reset (to zero) by pressing the Y button on the gamepad (Triangle on a PS4␣
↪controller)
 * <p>
 * The rotational velocities should follow the change in corresponding axes.
 */

@TeleOp(name="Concept: IMU Orientation", group="Concept")
@Disabled
public class ConceptExploringIMUOrientation extends LinearOpMode {
    static RevHubOrientationOnRobot.LogoFacingDirection[] logoFacingDirections
            = RevHubOrientationOnRobot.LogoFacingDirection.values();
    static RevHubOrientationOnRobot.UsbFacingDirection[] usbFacingDirections
            = RevHubOrientationOnRobot.UsbFacingDirection.values();
    static int LAST_DIRECTION = logoFacingDirections.length - 1;
    static float TRIGGER_THRESHOLD = 0.2f;

    IMU imu;
    int logoFacingDirectionPosition;
    int usbFacingDirectionPosition;
    boolean orientationIsValid = true;

    @Override public void runOpMode() throws InterruptedException {
        imu = hardwareMap.get(IMU.class, "imu");
```

(continues on next page)

(continued from previous page)

```
        logoFacingDirectionPosition = 0; // Up
        usbFacingDirectionPosition = 2; // Forward

        updateOrientation();

        boolean justChangedLogoDirection = false;
        boolean justChangedUsbDirection = false;

        // Loop until stop requested
        while (!isStopRequested()) {

            // Check to see if Yaw reset is requested (Y button)
            if (gamepad1.y) {
                telemetry.addData("Yaw", "Resetting\n");
                imu.resetYaw();
            } else {
                telemetry.addData("Yaw", "Press Y (triangle) on Gamepad to reset.\n");
            }

            // Check to see if new Logo Direction is requested
            if (gamepad1.left_bumper || gamepad1.right_bumper) {
                if (!justChangedLogoDirection) {
                    justChangedLogoDirection = true;
                    if (gamepad1.left_bumper) {
                        logoFacingDirectionPosition--;
                        if (logoFacingDirectionPosition < 0) {
                            logoFacingDirectionPosition = LAST_DIRECTION;
                        }
                    } else {
                        logoFacingDirectionPosition++;
                        if (logoFacingDirectionPosition > LAST_DIRECTION) {
                            logoFacingDirectionPosition = 0;
                        }
                    }
                    updateOrientation();
                }
            } else {
                justChangedLogoDirection = false;
            }

            // Check to see if new USB Direction is requested
            if (gamepad1.left_trigger > TRIGGER_THRESHOLD || gamepad1.right_trigger > TRIGGER_
↪THRESHOLD) {
                if (!justChangedUsbDirection) {
                    justChangedUsbDirection = true;
                    if (gamepad1.left_trigger > TRIGGER_THRESHOLD) {
                        usbFacingDirectionPosition--;
                        if (usbFacingDirectionPosition < 0) {
                            usbFacingDirectionPosition = LAST_DIRECTION;
                        }
                    } else {
                        usbFacingDirectionPosition++;
                        if (usbFacingDirectionPosition > LAST_DIRECTION) {
                            usbFacingDirectionPosition = 0;
                        }
                    }
                    updateOrientation();
                }
            } else {
```

(continues on next page)

(continued from previous page)

```
                justChangedUsbDirection = false;
            }

            // Display User instructions and IMU data
            telemetry.addData("logo Direction (set with bumpers)",␣
↪logoFacingDirections[logoFacingDirectionPosition]);
            telemetry.addData("usb Direction (set with triggers)",␣
↪usbFacingDirections[usbFacingDirectionPosition] + "\n");

            if (orientationIsValid) {
                YawPitchRollAngles orientation = imu.getRobotYawPitchRollAngles();
                AngularVelocity angularVelocity = imu.getRobotAngularVelocity(AngleUnit.DEGREES);

                telemetry.addData("Yaw (Z)", "%.2f Deg. (Heading)", orientation.getYaw(AngleUnit.
↪DEGREES));
                telemetry.addData("Pitch (X)", "%.2f Deg.", orientation.getPitch(AngleUnit.
↪DEGREES));
                telemetry.addData("Roll (Y)", "%.2f Deg.\n", orientation.getRoll(AngleUnit.
↪DEGREES));
                telemetry.addData("Yaw (Z) velocity", "%.2f Deg/Sec", angularVelocity.
↪zRotationRate);
                telemetry.addData("Pitch (X) velocity", "%.2f Deg/Sec", angularVelocity.
↪xRotationRate);
                telemetry.addData("Roll (Y) velocity", "%.2f Deg/Sec", angularVelocity.
↪yRotationRate);
            } else {
                telemetry.addData("Error", "Selected orientation on robot is invalid");
            }

            telemetry.update();
        }
    }

    // apply any requested orientation changes.
    void updateOrientation() {
        RevHubOrientationOnRobot.LogoFacingDirection logo =␣
↪logoFacingDirections[logoFacingDirectionPosition];
        RevHubOrientationOnRobot.UsbFacingDirection usb =␣
↪usbFacingDirections[usbFacingDirectionPosition];
        try {
            RevHubOrientationOnRobot orientationOnRobot = new RevHubOrientationOnRobot(logo, usb);
            imu.initialize(new IMU.Parameters(orientationOnRobot));
            orientationIsValid = true;
        } catch (IllegalArgumentException e) {
            orientationIsValid = false;
        }
    }
}
```

SensorIMUOrthogonal.java

Shows how to define your Hub orientation on the robot, for simple orthogonal (90 degree) mounting

SensorIMUOrthogonal.java

```
/* Copyright (c) 2022 FIRST. All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without modification,
 * are permitted (subject to the limitations in the disclaimer below) provided that
 * the following conditions are met:
 *
 * Redistributions of source code must retain the above copyright notice, this list
 * of conditions and the following disclaimer.
 *
 * Redistributions in binary form must reproduce the above copyright notice, this
 * list of conditions and the following disclaimer in the documentation and/or
 * other materials provided with the distribution.
 *
 * Neither the name of FIRST nor the names of its contributors may be used to endorse or
 * promote products derived from this software without specific prior written permission.
 *
 * NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED BY THIS
 * LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
 * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
 * SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
 * CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

package org.firstinspires.ftc.robotcontroller.external.samples;

import com.qualcomm.hardware.rev.RevHubOrientationOnRobot;
import com.qualcomm.robotcore.eventloop.opmode.Disabled;
import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.eventloop.opmode.TeleOp;
import com.qualcomm.robotcore.hardware.IMU;

import org.firstinspires.ftc.robotcore.external.navigation.AngleUnit;
import org.firstinspires.ftc.robotcore.external.navigation.AngularVelocity;
import org.firstinspires.ftc.robotcore.external.navigation.YawPitchRollAngles;

/**
 * {@link SensorIMUOrthogonal} shows how to use the new universal {@link IMU} interface. This
 * interface may be used with the BNO055 IMU or the BHI260 IMU. It assumes that an IMU is configured
 * on the robot with the name "imu".
 * <p>
 * The sample will display the current Yaw, Pitch and Roll of the robot.<br>
 * With the correct orientation parameters selected, pitch/roll/yaw should act as follows:
 * <p>
 *   Pitch value should INCREASE as the robot is tipped UP at the front. (Rotation about X) <br>
 *   Roll value should INCREASE as the robot is tipped UP at the left side. (Rotation about Y) <br>
 *   Yaw value should INCREASE as the robot is rotated Counter Clockwise. (Rotation about Z) <br>
 * <p>
 * The yaw can be reset (to zero) by pressing the Y button on the gamepad (Triangle on a PS4␣
↪controller)
```

(continues on next page)

(continued from previous page)

```
 * <p>
 * This specific sample assumes that the Hub is mounted on one of the three orthogonal planes
 * (X/Y, X/Z or Y/Z) and that the Hub has only been rotated in a range of 90 degree increments.
 * <p>
 * Note: if your Hub is mounted on a surface angled at some non-90 Degree multiple (like 30) look at
 *       the alternative SensorImuNonOrthogonal sample in this folder.
 * <p>
 * This "Orthogonal" requirement means that:
 * <p>
 * 1) The Logo printed on the top of the Hub can ONLY be pointing in one of six directions:
 *    FORWARD, BACKWARD, UP, DOWN, LEFT and RIGHT.
 * <p>
 * 2) The USB ports can only be pointing in one of the same six directions:<br>
 *    FORWARD, BACKWARD, UP, DOWN, LEFT and RIGHT.
 * <p>
 * So, To fully define how your Hub is mounted to the robot, you must simply specify:<br>
 *    logoFacingDirection<br>
 *    usbFacingDirection
 * <p>
 * Use Android Studio to Copy this Class, and Paste it into your team's code folder with a new name.
 * Remove or comment out the @Disabled line to add this OpMode to the Driver Station OpMode list.
 * <p>
 * Finally, choose the two correct parameters to define how your Hub is mounted and edit this OpMode
 * to use those parameters.
 */
@TeleOp(name = "Sensor: IMU Orthogonal", group = "Sensor")
@Disabled   // Comment this out to add to the OpMode list
public class SensorIMUOrthogonal extends LinearOpMode
{
    // The IMU sensor object
    IMU imu;

    //----------------------------------------------------------------------------------------------
    // Main logic
    //----------------------------------------------------------------------------------------------

    @Override public void runOpMode() throws InterruptedException {

        // Retrieve and initialize the IMU.
        // This sample expects the IMU to be in a REV Hub and named "imu".
        imu = hardwareMap.get(IMU.class, "imu");

        /* Define how the hub is mounted on the robot to get the correct Yaw, Pitch and Roll values.
         *
         * Two input parameters are required to fully specify the Orientation.
         * The first parameter specifies the direction the printed logo on the Hub is pointing.
         * The second parameter specifies the direction the USB connector on the Hub is pointing.
         * All directions are relative to the robot, and left/right is as-viewed from behind the robot.
         */

        /* The next two lines define Hub orientation.
         * The Default Orientation (shown) is when a hub is mounted horizontally with the printed logo pointing UP and the USB port pointing FORWARD.
         *
         * To Do:  EDIT these two lines to match YOUR mounting configuration.
         */
        RevHubOrientationOnRobot.LogoFacingDirection logoDirection = RevHubOrientationOnRobot.LogoFacingDirection.UP;
```

(continues on next page)

(continued from previous page)

```
        RevHubOrientationOnRobot.UsbFacingDirection  usbDirection  = RevHubOrientationOnRobot.
↪UsbFacingDirection.FORWARD;

        RevHubOrientationOnRobot orientationOnRobot = new RevHubOrientationOnRobot(logoDirection,␣
↪usbDirection);

        // Now initialize the IMU with this mounting orientation
        // Note: if you choose two conflicting directions, this initialization will cause a code␣
↪exception.
        imu.initialize(new IMU.Parameters(orientationOnRobot));

        // Loop and update the dashboard
        while (!isStopRequested()) {

            telemetry.addData("Hub orientation", "Logo=%s   USB=%s\n ", logoDirection,␣
↪usbDirection);

            // Check to see if heading reset is requested
            if (gamepad1.y) {
                telemetry.addData("Yaw", "Resetting\n");
                imu.resetYaw();
            } else {
                telemetry.addData("Yaw", "Press Y (triangle) on Gamepad to reset\n");
            }

            // Retrieve Rotational Angles and Velocities
            YawPitchRollAngles orientation = imu.getRobotYawPitchRollAngles();
            AngularVelocity angularVelocity = imu.getRobotAngularVelocity(AngleUnit.DEGREES);

            telemetry.addData("Yaw (Z)", "%.2f Deg. (Heading)", orientation.getYaw(AngleUnit.
↪DEGREES));
            telemetry.addData("Pitch (X)", "%.2f Deg.", orientation.getPitch(AngleUnit.DEGREES));
            telemetry.addData("Roll (Y)", "%.2f Deg.\n", orientation.getRoll(AngleUnit.DEGREES));
            telemetry.addData("Yaw (Z) velocity", "%.2f Deg/Sec", angularVelocity.zRotationRate);
            telemetry.addData("Pitch (X) velocity", "%.2f Deg/Sec", angularVelocity.xRotationRate);
            telemetry.addData("Roll (Y) velocity", "%.2f Deg/Sec", angularVelocity.yRotationRate);
            telemetry.update();
        }
    }
}
```

SensorIMUNonOrthogonal.java

Shows how to define (with the Angles method) your Hub orientation on the robot for a non-orthogonal orientation

`SensorIMUNonOrthogonal.java`

```
/* Copyright (c) 2022 FIRST. All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without modification,
 * are permitted (subject to the limitations in the disclaimer below) provided that
 * the following conditions are met:
 *
 * Redistributions of source code must retain the above copyright notice, this list
 * of conditions and the following disclaimer.
 *
 * Redistributions in binary form must reproduce the above copyright notice, this
```

(continues on next page)

(continued from previous page)

```
 * list of conditions and the following disclaimer in the documentation and/or
 * other materials provided with the distribution.
 *
 * Neither the name of FIRST nor the names of its contributors may be used to endorse or
 * promote products derived from this software without specific prior written permission.
 *
 * NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED BY THIS
 * LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
 * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
 * SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
 * CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

package org.firstinspires.ftc.teamcode;

import static com.qualcomm.hardware.rev.RevHubOrientationOnRobot.xyzOrientation;

import com.qualcomm.hardware.rev.RevHubOrientationOnRobot;
import com.qualcomm.robotcore.eventloop.opmode.Disabled;
import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.eventloop.opmode.TeleOp;
import com.qualcomm.robotcore.hardware.IMU;

import org.firstinspires.ftc.robotcore.external.navigation.AngleUnit;
import org.firstinspires.ftc.robotcore.external.navigation.AngularVelocity;
import org.firstinspires.ftc.robotcore.external.navigation.Orientation;
import org.firstinspires.ftc.robotcore.external.navigation.YawPitchRollAngles;

/**
 * {@link SensorIMUNonOrthogonal} shows how to use the new universal {@link IMU} interface. This
 * interface may be used with the BNO055 IMU or the BHI260 IMU. It assumes that an IMU is configured
 * on the robot with the name "imu".
 * <p>
 * The sample will display the current Yaw, Pitch and Roll of the robot.<br>
 * With the correct orientation parameters selected, pitch/roll/yaw should act as follows:
 * <p>
 *   Pitch value should INCREASE as the robot is tipped UP at the front. (Rotation about X) <br>
 *   Roll value should INCREASE as the robot is tipped UP at the left side. (Rotation about Y) <br>
 *   Yaw value should INCREASE as the robot is rotated Counter Clockwise. (Rotation about Z) <br>
 * <p>
 * The yaw can be reset (to zero) by pressing the Y button on the gamepad (Triangle on a PS4␣
↪controller)
 * <p>
 * This specific sample DOES NOT assume that the Hub is mounted on one of the three orthogonal
 * planes (X/Y, X/Z or Y/Z) OR that the Hub has only been rotated in a range of 90 degree␣
↪increments.
 * <p>
 * Note: if your Hub is mounted Orthogonally (on a orthogonal surface, angled at some multiple of
 * 90 Degrees) then you should use the simpler SensorImuOrthogonal sample in this folder.
 * <p>
 * But... If your Hub is mounted Non-Orthogonally, you must specify one or more rotational angles
 * that transform a "Default" Hub orientation into your desired orientation.  That is what is
 * illustrated here.
```

(continues on next page)

(continued from previous page)

```
 * <p>
 * Use Android Studio to Copy this Class, and Paste it into your team's code folder with a new name.
 * Remove or comment out the @Disabled line to add this OpMode to the Driver Station OpMode list.
 * <p>
 * Finally, edit this OpMode to use at least one angle around an axis to orient your Hub.
 */
@TeleOp(name = "Sensor: IMU Non-Orthogonal", group = "Sensor")
@Disabled     // Comment this out to add to the OpMode list
public class W_nonortho extends LinearOpMode
{
    // The IMU sensor object
    IMU imu;

    //----------------------------------------------------------------------------------------------
    // Main logic
    //----------------------------------------------------------------------------------------------

    @Override public void runOpMode() throws InterruptedException {

        // Retrieve and initialize the IMU.
        // This sample expects the IMU to be in a REV Hub and named "imu".
        imu = hardwareMap.get(IMU.class, "imu");

        /* Define how the hub is mounted to the robot to get the correct Yaw, Pitch and Roll values.
         *
         * You can apply up to three axis rotations to orient your Hub according to how it's␣
↪mounted on the robot.
         *
         * The starting point for these rotations is the "Default" Hub orientation, which is:
         * 1) Hub laying flat on a horizontal surface, with the Printed Logo facing UP
         * 2) Rotated such that the USB ports are facing forward on the robot.
         *
         * The order that the rotations are performed matters, so this sample shows doing them in␣
↪the order X, Y, then Z.
         * For specifying non-orthogonal hub mounting orientations, we must temporarily use axes
         * defined relative to the Hub itself, instead of the usual Robot Coordinate System axes
         * used for the results the IMU gives us. In the starting orientation, the Hub axes are
         * aligned with the Robot Coordinate System:
         *
         * X Axis:  Starting at Center of Hub, pointing out towards I2C connectors
         * Y Axis:  Starting at Center of Hub, pointing out towards USB connectors
         * Z Axis:  Starting at Center of Hub, pointing Up through LOGO
         *
         * Positive rotation is defined by right-hand rule with thumb pointing in +ve direction on␣
↪axis.
         *
         * Some examples.
         *
         * ----------------------------------------------------------------------------------------
↪-----------------------------------------
         * Example A) Assume that the hub is mounted on a sloped plate at the back of the robot,␣
↪with the USB ports coming out the top of the hub.
         *  The plate is tilted UP 60 degrees from horizontal.
         *
         *  To get the "Default" hub into this configuration you would just need a single rotation.
         *  1) Rotate the Hub +60 degrees around the X axis to tilt up the front edge.
         *  2) No rotation around the Y or Z axes.
         *
         *  So the X,Y,Z rotations would be 60,0,0
```

(continues on next page)

(continued from previous page)

```
         *
         * ---------------------------------------------------------------------------------
↪---------------------------------------
         * Example B) Assume that the hub is laying flat on the chassis, but it has been twisted 30␣
↪degrees towards the right front wheel to make
         *  the USB cable accessible.
         *
         *  To get the "Default" hub into this configuration you would just need a single rotation,␣
↪but around a different axis.
         *  1) No rotation around the X or Y axes.
         *  1) Rotate the Hub -30 degrees (Clockwise) around the Z axis, since a positive angle␣
↪would be Counter Clockwise.
         *
         *  So the X,Y,Z rotations would be 0,0,-30
         *
         * ---------------------------------------------------------------------------------
↪---------------------------------------
         *  Example C) Assume that the hub is mounted on a vertical plate on the right side of the␣
↪robot, with the Logo facing out, and the
         *  Hub rotated so that the USB ports are facing down 30 degrees towards the back wheels of␣
↪the robot.
         *
         *  To get the "Default" hub into this configuration will require several rotations.
         *  1) Rotate the hub +90 degrees around the X axis to get it standing upright with the␣
↪logo pointing backwards on the robot
         *  2) Next, rotate the hub +90 around the Y axis to get it facing to the right.
         *  3) Finally rotate the hub +120 degrees around the Z axis to take the USB ports from␣
↪vertical to sloping down 30 degrees and
         *     facing towards the back of the robot.
         *
         *  So the X,Y,Z rotations would be 90,90,120
         */

        // The next three lines define the desired axis rotations.
        // To Do: EDIT these values to match YOUR mounting configuration.
        double xRotation = 0;  // enter the desired X rotation angle here.
        double yRotation = 0;  // enter the desired Y rotation angle here.
        double zRotation = 0;  // enter the desired Z rotation angle here.

        Orientation hubRotation = xyzOrientation(xRotation, yRotation, zRotation);

        // Now initialize the IMU with this mounting orientation
        RevHubOrientationOnRobot orientationOnRobot = new RevHubOrientationOnRobot(hubRotation);
        imu.initialize(new IMU.Parameters(orientationOnRobot));

        // Loop and update the dashboard
        while (!isStopRequested()) {
            telemetry.addData("Hub orientation", "X=%.1f,  Y=%.1f,  Z=%.1f \n", xRotation,␣
↪yRotation, zRotation);

            // Check to see if heading reset is requested
            if (gamepad1.y) {
                telemetry.addData("Yaw", "Resetting\n");
                imu.resetYaw();
            } else {
                telemetry.addData("Yaw", "Press Y (triangle) on Gamepad to reset\n");
            }

            // Retrieve Rotational Angles and Velocities
```

(continues on next page)

(continued from previous page)

```
            YawPitchRollAngles orientation = imu.getRobotYawPitchRollAngles();
            AngularVelocity angularVelocity = imu.getRobotAngularVelocity(AngleUnit.DEGREES);

            telemetry.addData("Yaw (Z)", "%.2f Deg. (Heading)", orientation.getYaw(AngleUnit.
→DEGREES));
            telemetry.addData("Pitch (X)", "%.2f Deg.", orientation.getPitch(AngleUnit.DEGREES));
            telemetry.addData("Roll (Y)", "%.2f Deg.\n", orientation.getRoll(AngleUnit.DEGREES));
            telemetry.addData("Yaw (Z) velocity", "%.2f Deg/Sec", angularVelocity.zRotationRate);
            telemetry.addData("Pitch (X) velocity", "%.2f Deg/Sec", angularVelocity.xRotationRate);
            telemetry.addData("Roll (Y) velocity", "%.2f Deg/Sec", angularVelocity.yRotationRate);
            telemetry.update();
        }
    }
}
```

These three Java samples include extensive comments describing the IMU interface, consistent with this tutorial. In particular, `SensorIMUNonOrthogonal.java` describes three helpful examples.

SDK Resources

Advanced programmers are invited to browse the Javadocs documentation (API), particularly in:

- `com.qualcomm.robotcore.hardware`
- `org.firstinspires.ftc.robotcore.external.navigation`

The new universal IMU classes for SDK 8.1 are:

- `IMU`
- `ImuOrientationOnRobot`
- `YawPitchRollAngles`
- `RevHubOrientationOnRobot`

The Javadocs describe other IMU methods and variables not covered in this basic tutorial.

Summary

The SDK 8.1 provides a universal interface that supports both the BHI260AP and BNO055 IMU. This basic tutorial introduced some new features:

- robot configuration allows selection of IMU type
- three ways to specify Hub mounting orientation on the robot
- new Blocks and Java methods to read data from both IMU types

Teams using the new Control Hub IMU must use at least SDK 8.1 AND must update to at least Control Hub OS 1.1.3.

However **all teams** are encouraged to begin using the universal IMU classes and methods for **new** Blocks and Java code, and consider migrating **existing** code.

Questions, comments and corrections to westsiderobotics@verizon.net

21.7.7 Using the Kotlin Programming Language

What Is Kotlin?

The Kotlin programming language is a a modern alternative to the Java programing language that compiles and runs on the Java Virtual Machine (JVM) and can be used to develop Android applications. It was developed by JetBrains, the same company that developed the IntelliJ IDE (the basis for Android Studio).

- https://kotlinlang.org/

Being based on Java, Kotlin shares many of the same features and syntax. However, it also adds many new features and syntax that can make it easier to write code and less prone to errors. Some of the features of Kotlin include:

- Full interoperability with Java; you can use Java classes and libraries from Kotlin and vice versa.
- Type inference; Kotlin allows you to use type inference when needed, that is, you don't need to specify the type of a variable when it can be inferred from the context (`var myString = "Hi!"`).
- No semicolons; Kotlin does not require semicolons to end statements.
- Data classes; Kotlin has a concise syntax for creating classes that are used to store data.
- Extension functions; Kotlin allows you to add functions to existing classes without having to modify the original class.
- Null safety; Kotlin has a type system that helps eliminate null pointer exceptions.
- Operator overloading; Kotlin allows you to define how operators such as + and * work with your own classes.
- Many more!

In addition, if you don't want to learn how to code in Kotlin from scratch, the Android Studio IDE has a tool to convert sections of code or an entire Java file to a Kotlin file. This is extremely useful to learn how certain Java code is written in Kotlin.

Because Kotlin is fully interoperable, you can also use all your existing Java code in a Kotlin project without having to convert it.

Using Kotlin in *FIRST* Tech Challenge

While there is no rule (as of the writing of this document) prohibiting Kotlin as a programming option in *FIRST* Tech Challenge, it is not one of the recommended tools as listed in **<RS02>** "Recommended Programming Tools" portion of the *FIRST* Tech Challenge *Game Manual Part 1*. Teams that use Kotlin do so at their own risk and should expect that there will not be technical help/support available at events in the case of software issues.

Installing Kotlin In Your Project

To use Kotlin in your Android project, you need to add the Kotlin plugin to your project. This is done by adding the following lines to the root `build.gradle` file in the `buildscript` section:

```
buildscript {

    ext.kotlin_version = '1.9.22' <----- ADD THIS LINE, UPDATE VERSION TO LATEST IF NEEDED

    repositories {
        mavenCentral()
        google()
    }

    dependencies {
        classpath 'com.android.tools.build:gradle:8.7.0'
        classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:$kotlin_version" <----- ADD THIS LINE
```

(continues on next page)

(continued from previous page)

```
    }
}
```

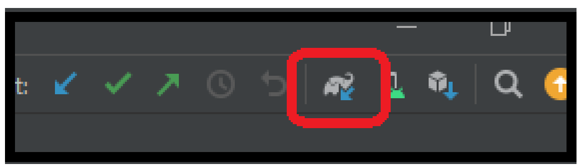
Note: This file is located in the base folder of your project, not the one in the TeamCode module nor the one in the FtcRobot-Controller module.

Note: The exact kotlin version can be changed/updated if desired per new releases. The latest version as of this writing is 1.8.20 but you should check the Kotlin website to see if a newer version (one that is compatible with the current Gradle version) is available (see table here).

Next you need to add the Kotlin plugin to the `build.gradle` file in the TeamCode module. Open the file and find the following section near the top of the file. Change it to look like this:

```
// Include common definitions from above.
apply from: '../build.common.gradle'
apply from: '../build.dependencies.gradle'
apply plugin: 'kotlin-android' <----- ADD THIS LINE
```

Finally, you need to run a Gradle sync to download the Kotlin plugin and any other dependencies. This is done by clicking on the `Sync Now` link in the upper right corner of the Android Studio window.



Make sure you are on a reliable internet connection when you do this!

Note: If you get an error that says "Kotlin not configured" when you try to run a Gradle sync, you may need to install the Kotlin plugin. To do this, go to `File -> Settings -> Plugins` and search for "Kotlin". Click on the `Install` button to install the plugin.

21.7.8 HuskyLens Intro for *FIRST* Tech Challenge

Introduction

This is a simple tutorial to introduce the use of HuskyLens in *FIRST* Tech Challenge (FTC), for teams that **already decided** to explore its potential.

Basic support for this **vision sensor** was added to the FTC SDK version 9.0 in September 2023 with the CENTERSTAGE robot game kickoff.

HuskyLens uses **on-board programming** to perform AI-assisted learning, vision processing and recognition. It plugs into an **I2C sensor port** of a REV Control Hub or REV Expansion Hub.

HuskyLens is **not a USB webcam**, and **does not use** the FTC *VisionPortal* software.

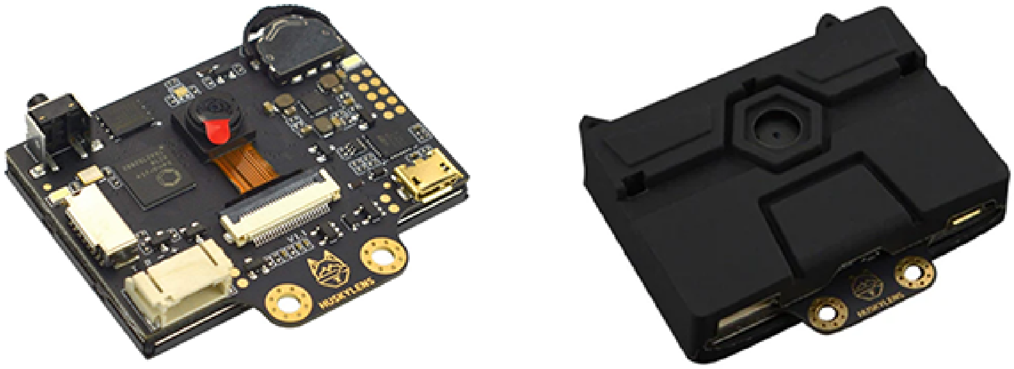


Fig. 265: DFRobot HuskyLens

Electrical Connection

You will need a **custom adapter cable** to connect the HuskyLens to an I2C port on a REV Control Hub or Expansion Hub. The 4 wires/pins of the HuskyLens connector are not in the same order/position as the 4 pins on the REV Hub.

Three of the wires have **the same color** as wires in the REV sensor cable. Your custom cable should connect **red to red**, **black to black**, and **blue to blue**. This leaves only the HuskyLens **green wire**; connect it to the REV **white wire**. Simple!

This tutorial does **not** cover the (many) ways to:

- modify an existing cable (change pin order in one connector), **OR**
- fabricate a custom cable, with:
  - soldering
  - crimped connectors
  - lever nuts (example below)

FTC Game Manual 1 allows this work, but teams must ensure high quality for robot competition all season.

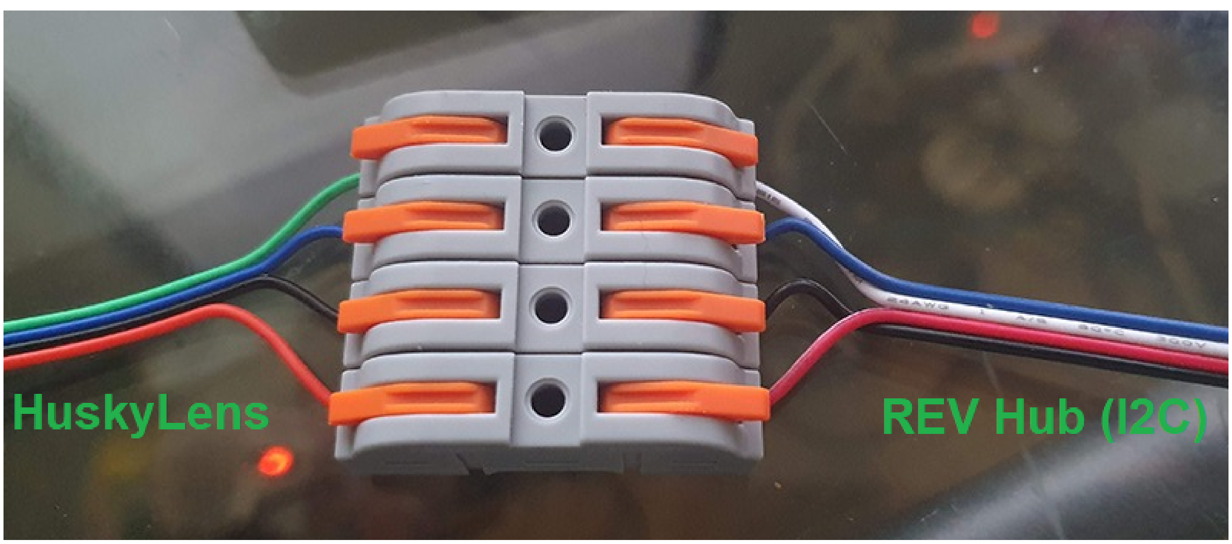


Fig. 266: image credit: @texasdiaz

To confirm these wiring instructions are correct, you could study the HuskyLens documentation and the REV Hub documentation. You will see the following "pinout" info:

- HuskyLens **green** wire 1 (“T”) SDA or data == REV Hub **white** wire 3 “SDA” or data
- HuskyLens **blue** wire 2 (“R”) SCL or clock == REV Hub **blue** wire 4 “SCL” or clock
- HuskyLens **black** wire 3 (“-”) GND or ground == REV Hub **black** wire 1 “GND” or ground
- HuskyLens **red** wire 4 (“+”) VCC or +3.3-5VDC == REV Hub **red** wire 2 “3.3V” or Vcc

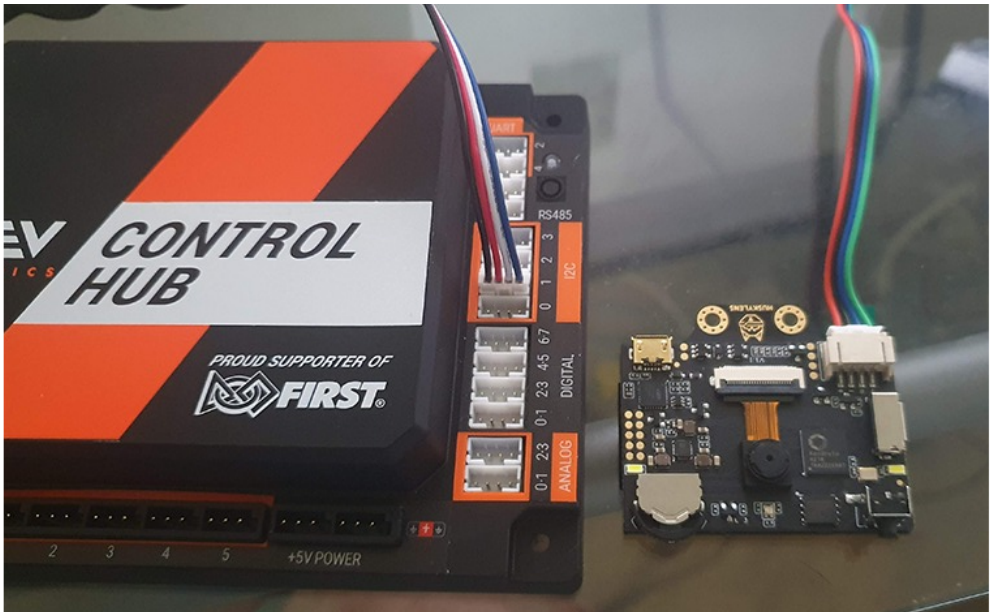


Fig. 267: image credit: @texasdiaz

Configuration

Plug the HuskyLens into a REV Hub I2C port, using your new adapter cable. The I2C connections labeled **Bus 1, 2 or 3** are suggested, to avoid (unlikely) overload of data traffic.

The label 0 (zero) is I2C Bus 0, which likely has a **built-in IMU** on its Port 0. An I2C Bus can contain multiple I2C Ports, sharing traffic.

On the Driver Station, touch the three-dots menu, and `Configure Robot`.

Edit an existing (correct) configuration, or touch `New`. Touch `Scan`, then navigate (through the Portal level) to the specific `Expansion Hub` or `Control Hub` with the HuskyLens plugged in.

Select `I2C Bus 3` or whichever Bus number has the HuskyLens plugged in.

Touch `Add`, and select device “HuskyLens” from the drop-down list for Port 0 (or first available port). Type the device name “huskylens”, as expected by the Sample OpMode.

Touch `Done` several times, then `Save`, to save and name/rename this updated robot configuration. Touch the DS “Back” arrow, returning to the DS app’s home screen.

Confirm that your new configuration is shown on-screen as the active configuration.

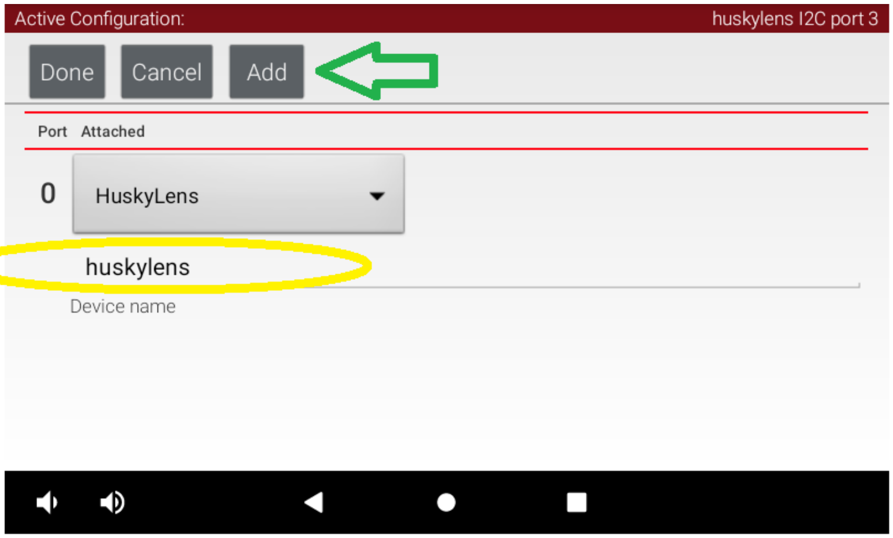


Fig. 268: Driver Station Config

Sample OpMode

Connect your programming computer to the Robot Controller, and open the programming software. This tutorial uses **FTC Blocks**.

Note: OnBot Java and **Android Studio** users can easily follow along, since the Java Sample OpMode uses the same programming logic and is well commented.

In FTC Blocks, create a new OpMode using the sample called "SensorHuskyLens":

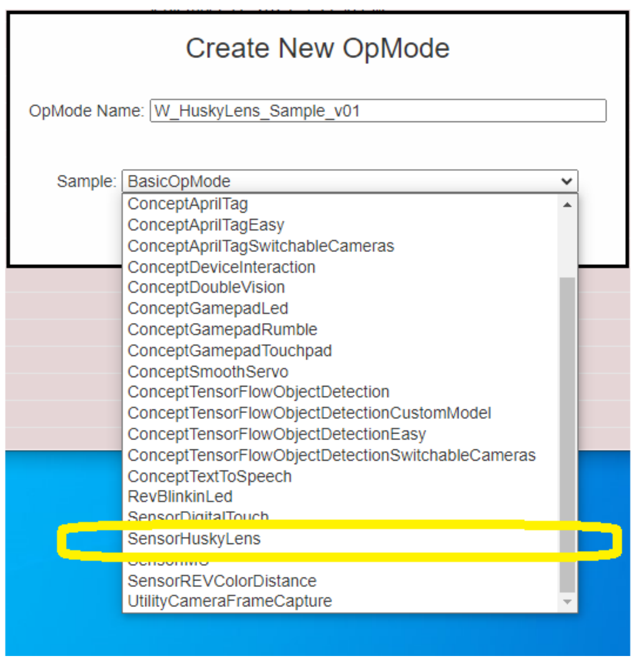


Fig. 269: HuskyLens Blocks Sample

Change the OpMode type from `TeleOp` to `Autonomous`, since this sample does not use the gamepads.

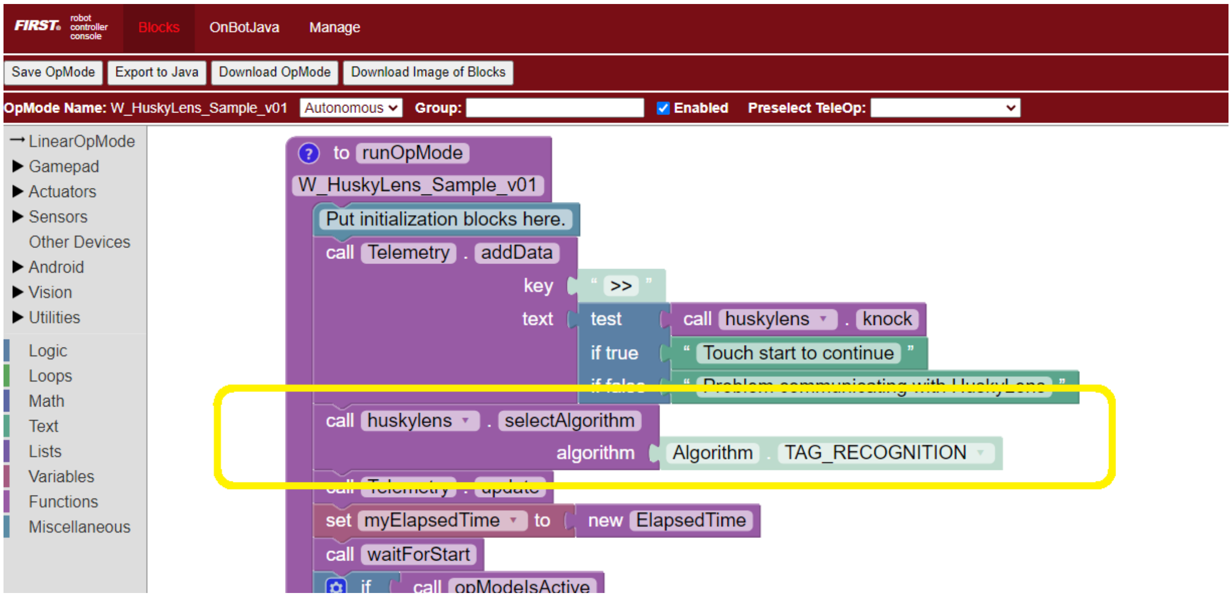


Fig. 270: HuskyLens Blocks Algorithm

Notice the default algorithm here is `TAG_RECOGNITION`, which simply detects any (common) AprilTags in the sensor's field of view. This recognition is unrelated to the FTC game CENTERSTAGE and its 10 AprilTags with metadata. Instead, this is a simple built-in, generic function of HuskyLens, used here only to validate the sensor's operation.

For AprilTag recognition and navigation, FTC teams may find much more value from a UVC webcam and the FTC *VisionPortal* software. An FTC robot may use HuskyLens **and** USB webcams.

Click `Save OpMode`, then select and run this OpMode from the Driver Station. After touching the Start arrow, point the HuskyLens at any AprilTag from the common 36h11 family:

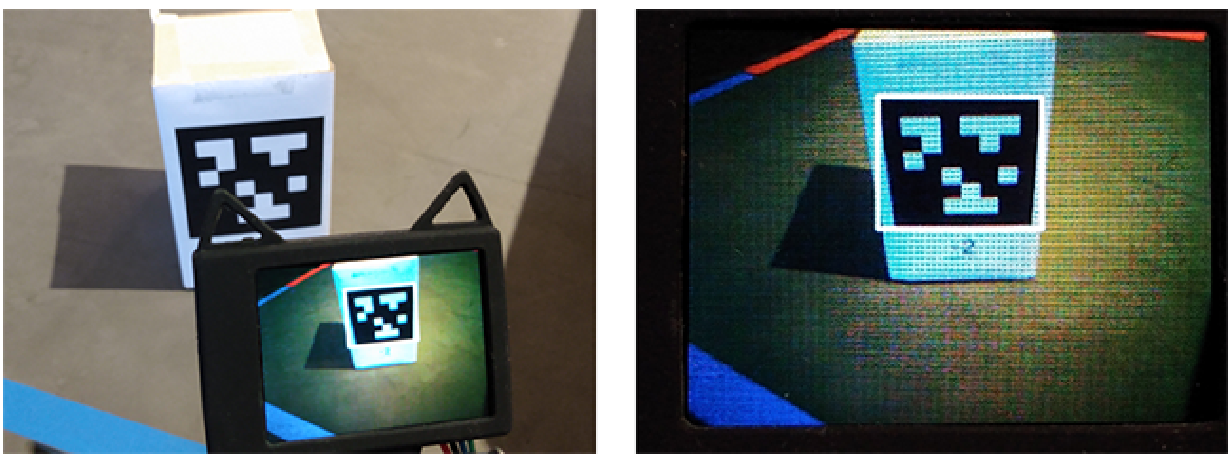


Fig. 271: Uncategorized AprilTag Detected

The HuskyLens' small screen will show the recognized AprilTag, surrounded by a thin white Bounding Box.

Here's the corresponding DS Telemetry:

The data includes:

- number of objects (called "blocks") detected
- ID code of object (might not be correct or meaningful)
- size of Bounding Box, in pixels

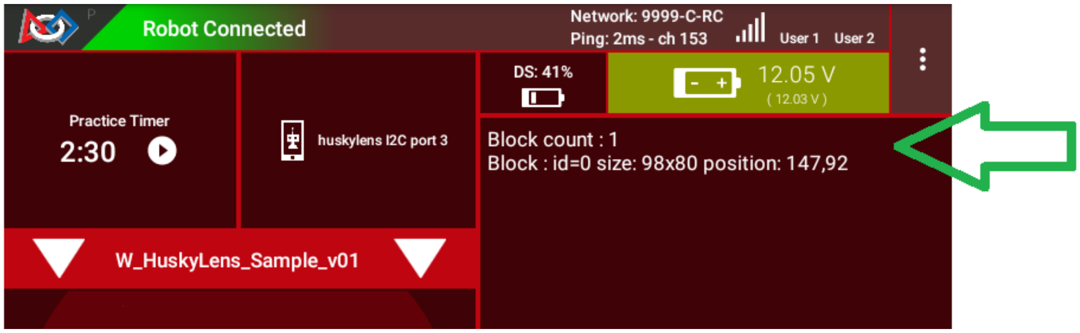


Fig. 272: AprilTag Telemetry

- center position of Bounding Box, in pixels, with (X, Y) origin at the top left

The HuskyLens device screen is 320 x 240 pixels, with center at position (160, 120).

Congratulations! At this point, you have validated the HuskyLens device, its connection to the REV Hub, and the Sample OpMode program.

AprilTag Detection

Now you can test whether the HuskyLens can detect the AprilTag's position on the CENTERSTAGE Spike Marks. This is not a real game scenario, since a Team Prop (Team Game Element) cannot use an AprilTag. This simply verifies whether your robot could aim the HuskyLens to "see" 2 or 3 Spike Marks in a single view.

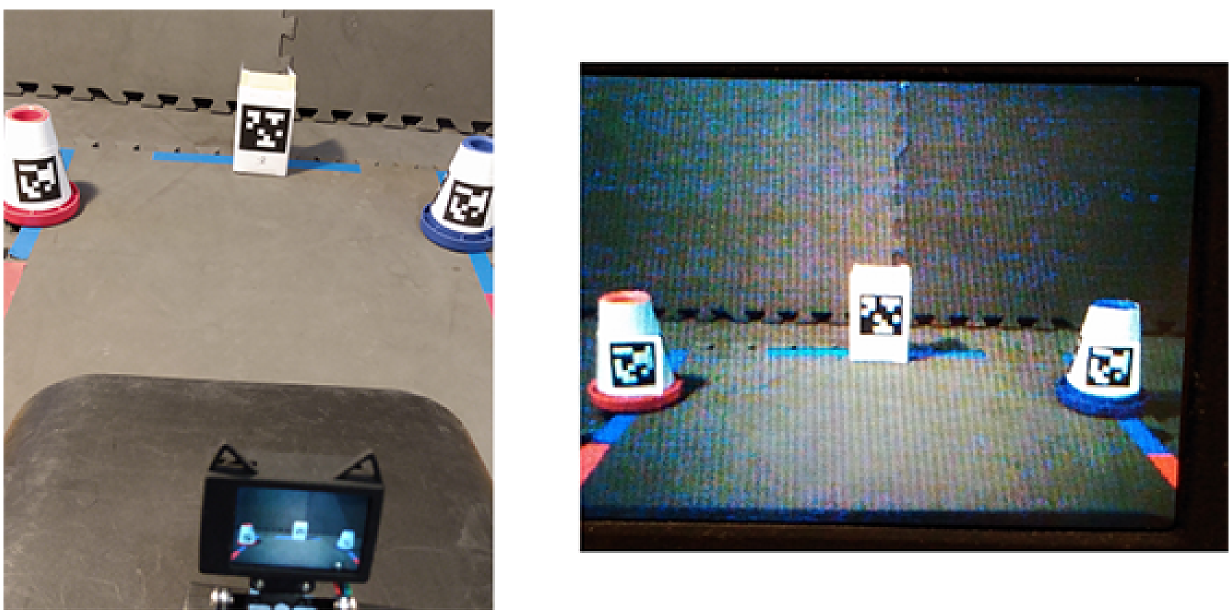


Fig. 273: HuskyLens Viewing 3 Uncategorized Tags

Here the HuskyLens was placed in a feasible position, about 10 inches from the mat, near the middle of the foam tile before the Spike-Mark tile. The view **does include** the middle of all three Spike Marks.

All three AprilTags were recognized:

This validates the possibility that HuskyLens could recognize a trained object in one of various known positions – useful for the Autonomous phase of the CENTERSTAGE game.

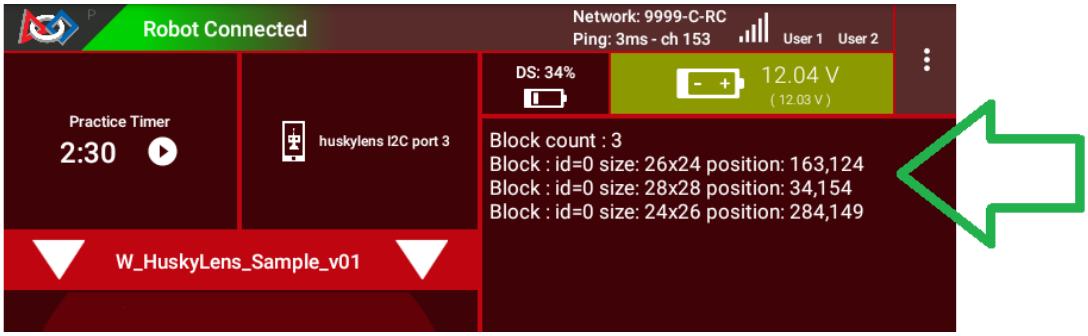


Fig. 274: Telemetry Showing 3 Blocks

Single Color Training

Soon you will try a different algorithm called `COLOR_RECOGNITION`. But first you need the HuskyLens to "learn" a single color, using its built-in AI feature.

Choose any object, about 3 to 4 inches in size, that's completely one color – any color. Here we use a flat square beverage coaster (LEGO!), with a uniform **red color**.

Place this object in the position and lighting that you expect to use for detection. This could be on a CENTERSTAGE Spike Mark, if available.

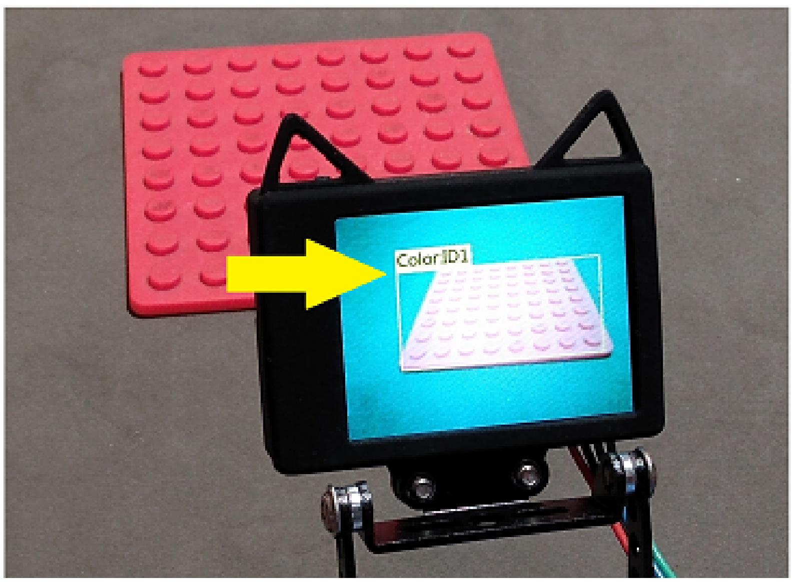


Fig. 275: Red Color ID

In the above image, the trained color is shown as ``**Color:ID1**`` with a rectangular Bounding Box. The following steps describe how to do this training.

The **HuskyLens instructions** for learning a color are posted online. You could try to follow those, or use the equivalent description here. Some practice may be required!

On the top of the HuskyLens, the wheel at the left side is called the **Function button** (actually a dial and button). At the right side is the small **Learning button**.

Dial the Function button to the right or left until **"Color Recognition"** is displayed at the bottom of the screen.

This is Step 1 only, under `Operation and Setting` of the HuskyLens instructions. For now, do not try to "learn" more than one color with Steps 2-4.

Point the plus-sign "+" icon in the center of the HuskyLens screen at your object's main color area. A white frame appears on the screen, targeting the main color. Aim the HuskyLens so the white frame includes only the target color.

This is Step 1 of `Learning and Detection`. Next comes Step 2, Color Learning.

With the main color framed, **long press** (press and hold) the small **Learning button** (right side). A yellow frame is displayed on the screen, indicating that HuskyLens is learning the color. During this long press, move the HuskyLens while pointing at the color area, to let HuskyLens learn the color from various distances and angles. Then, release the Learning button to complete learning that color. Do not press the button again (ignore the prompt); allow the 5-second time-out to finish.

The long-press learning period can last for just a few seconds. After releasing the Learning button, you allowed the training to time-out – no more colors to learn. Training is done!

As shown above, the trained color will be shown on-screen as ``**Color:ID1**`` with a rectangular Bounding Box. This "block" (of color) will be reported in the Sample OpMode (next step).

If you want to do this over again, short-press the Learning button, then short-press again to **Forget** the learned color(s). This will make the plus-sign "+" icon appear again. Aim the plus-sign at the center of the color area, and repeat the learning (long-press the Learning button). Release and let the time-out finish.

This section showed how to train a single color. After completing this tutorial, you may wish to train **two colors** (e.g. a Red shade and a Blue shade). This is described near the end of this tutorial.

HuskyLens documentation refers to the color zone as a "block" of color. This is not the same as a physical block or cube. HuskyLens uses the same word "block" for recognitions.

Note the official warning:

> **Warning:** "Color recognition is greatly affected by ambient light. Sometimes HuskyLens may misidentify similar colors. Please try to keep the ambient light unchanged."

Single Color Detection

Aim the HuskyLens at one or more of your color-trained objects.

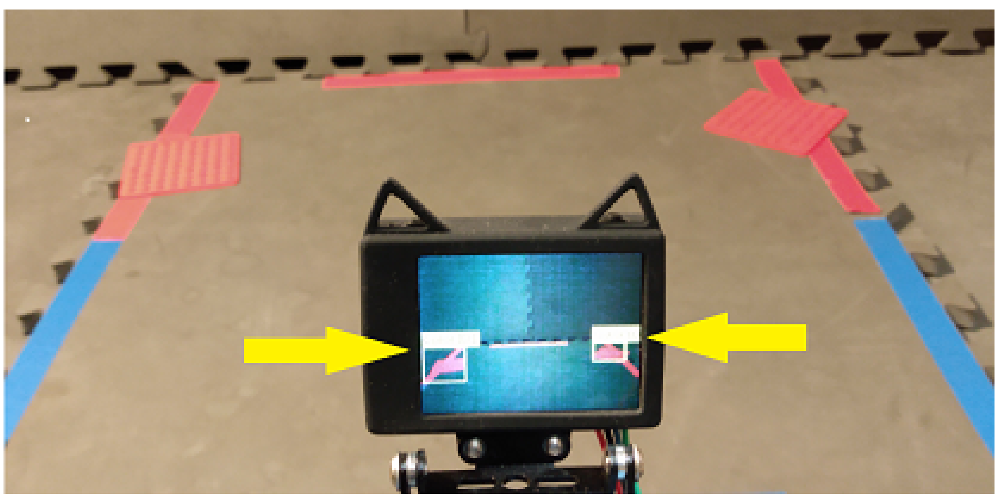


Fig. 276: HuskyLens Detecting Two Red Objects

As shown above, the HuskyLens should recognize and label your colored objects with ``**Color:ID1**``. Here, both red objects are identified (yellow arrows).

In the programming software (same OpMode), now select a different algorithm called `COLOR_RECOGNITION`:

In the Java sample OpMode, change the algorithm selection as follows:

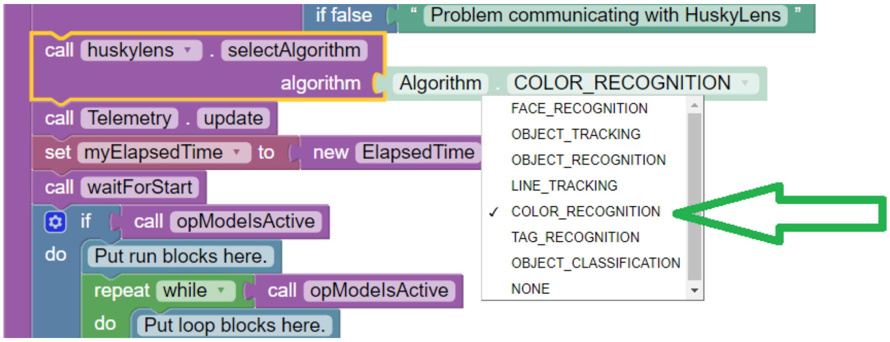


Fig. 277: Selecting COLOR_RECOGNITION algorithm

```
huskyLens.selectAlgorithm(HuskyLens.Algorithm.COLOR_RECOGNITION);
```

Save this OpMode, then select and run it on the Driver Station. Make sure the active configuration includes the HuskyLens.

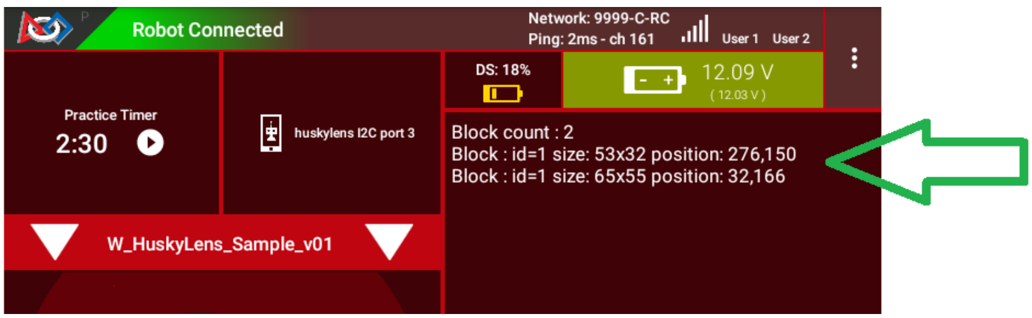


Fig. 278: DS Telemetry Two Objects

As shown above, the OpMode provides the size and location of the white Bounding Boxes (called "blocks"). This is done in a **FOR loop**; multiple recognitions are processed one at at time.

In the Java sample OpMode, **inside the FOR loop**, you could save or evaluate **specific** info for the currently recognized Bounding Box: `blocks[i].width`, `blocks[i].height`, `blocks[i].left`, `blocks[i].top`, and (for the Box's center) `blocks[i].x` and `blocks[i].y`. The Color ID `blocks[i].id` is always 1 here, for single-color detection. These values have Java type `int`.

Even if your Team Prop's color closely matches the color of the red or blue Spike Mark, you could write OpMode code to reject the narrow shape (aspect ratio) of an empty Spike Mark's Bounding Box.

Here's an example with a trained **blue object**:

Both blue objects were recognized by the OpMode:

Again, your code can evaluate the size and location of any provided Bounding Box, to verify a "real" recognition of your object.

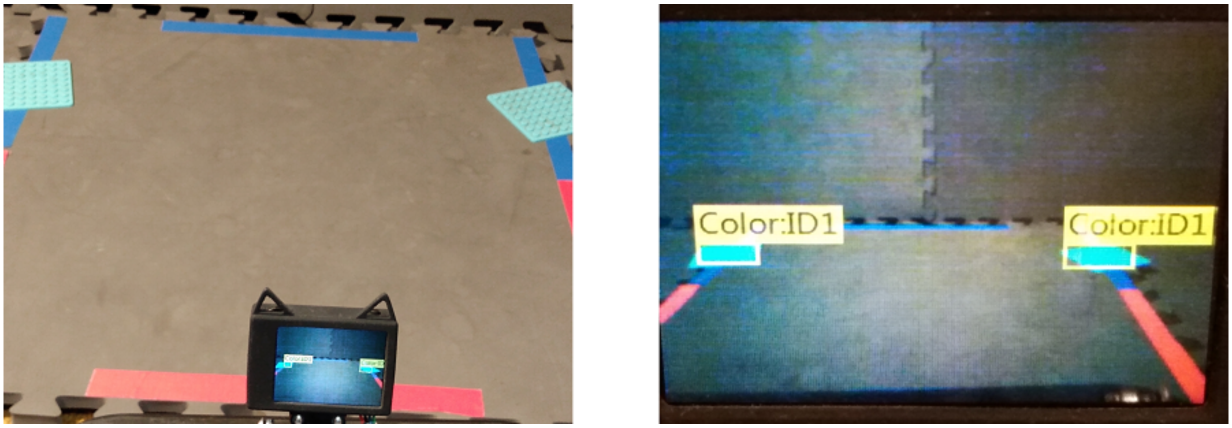


Fig. 279: HuskyLens Two Blue Objects

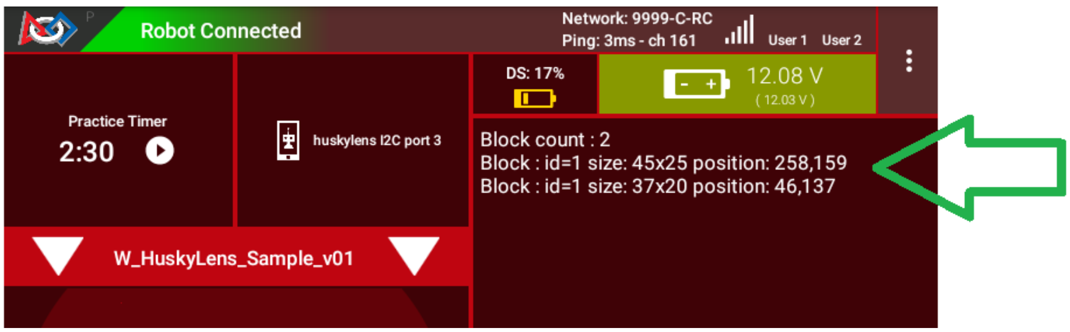


Fig. 280: Telemetry for Two Blue Objects

Competition Notes

1. Team Prop

Now you are ready to experiment with color recognition of an actual Team Prop, also called a Team Game Element. Study Game Manual 1 and the FTC Q&A for the Team Prop requirements. Choose your shades of "red" and "blue" (see note below), and follow the same steps as above.

2. Color

The above trained **blue object** is not the same shade of blue as the blue Spike Mark. This difference increases the chance of a distinct and correct recognition of the object color.

Game Manual 1 (Rule TE2) specifically allows the Team Prop to be a different shade of Red or Blue, compared to the official tape color of Spike Marks:

> *"The Team Game Element may include multiple shades of the assigned color."*

…and emphasized in the FTC Q&A:

> *"Light blue and pink are acceptable colors providing it is obvious to the field personnel which alliance the Team Prop belongs to."*

3. Lighting

The HuskyLens documentation provides a warning (shown above) that ambient lighting can impact recognition of a trained color.

For this reason, competition training should ideally be done with the Team Prop (Team Game Element) on the Spike Mark, and the HuskyLens in its planned match start position, "on-robot".

Also, the trained ambient lighting must be similar to expected match conditions. This may suggest performing the final color-training as part of tournament or match set-up. With practice, it could be done in a few seconds.

4. Programming

In this Sample OpMode, the main loop ends only upon touching the DS Stop button. For competition, teams should **modify this code** in at least two ways:

- for a significant recognition, take action or store key information – inside the FOR loop
- end the main loop based on your criteria, to continue the OpMode

As an example, you might set a Boolean variable `isPropDetected` to `true`, if a significant recognition has occurred.

You might also evaluate and store which randomized Spike Mark (red or blue tape stripe) holds the Team Prop.

Regarding the main loop, it could end after the HuskyLens views all three Spike Marks, or after your code provides a high-confidence result. If the HuskyLens' view includes more than one Spike Mark position, perhaps the **Bounding Box** size(s) and location(s) could be useful. Teams should consider how long to seek an acceptable recognition, and what to do otherwise.

In any case, the OpMode should exit the main loop and continue running, using any stored information.

Multi-Color Training

After completing the above tutorial with a single trained color, you may wish to train **two colors** (e.g. a Red shade and a Blue shade).

This would avoid the need for multiple color-training sessions during an FTC tournament. With single-color, you would train for Red before playing an FTC match as Red Alliance, and train for Blue before playing as Blue Alliance.

With multi-color, your Red-Alliance Autonomous OpMode could seek Red as **``Color:ID1``**, for example, and your Blue-Alliance Autonomous OpMode could seek Blue as **``Color:ID2``**.

The **HuskyLens instructions** for learning multiple colors are posted online. You could try to follow those, or use the equivalent description here. Again, some practice may be required!

Reminder: on the top of the HuskyLens, the wheel at the left side is called the **Function button** (actually a dial and button). At the right side is the small **Learning button**.

Step 1. Dial the Function button to the right or left until **"Color Recognition"** is displayed at the bottom of the screen.

Long press (press and hold) the Function button to select Color Recognition.

Step 2. This brings up the next menu, containing the choice "Learn Multiple". If needed, dial the Function button to highlight "Learn Multiple".

Short press (press and release) the Function button to select Learn Multiple.

This brings up the OFF-ON slider bar for "Learn Multiple". If needed, dial the Function Button to move the blue square to the **right side** of the blue slider bar. See yellow arrow:

Short press the Function button to set "Learn Multiple" to **ON**.

Step 3. Dial the Function button to the left, and **short press** to select "Save & Return".

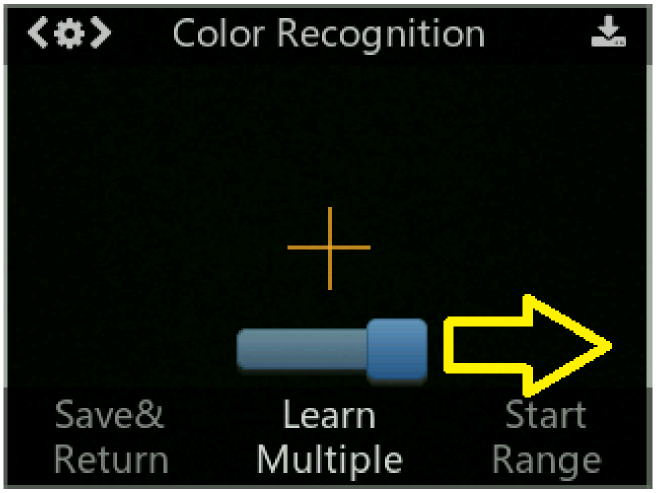


Fig. 281: HuskyLens - Learn Multiple

At the screen prompt "Do you want to save the parameters?" or "Do you save data?", **short press** the Function button to select "Yes". This saves the mode (again) as "Learn Multiple" and exits the settings menu.

Now ready for learning!

Step 4. As before, point the plus-sign "+" icon in the center of the HuskyLens screen at your object's main color area. A **white frame** appears on the screen, targeting the main color. Aim the HuskyLens so the white frame includes only the target color.

With the main color framed, **long press** (press and hold) the small **Learning button** (right side). A **yellow frame** appears on the screen, indicating that HuskyLens is learning the color.

During this long press, move the HuskyLens while pointing at the color area, to let HuskyLens learn the color from various distances and angles. Then, release the Learning button to complete learning that color.

The long-press learning period can last for just a few seconds. After releasing the Learning button, **``Color:ID1``** is now trained, with its label shown on-screen. Easy!

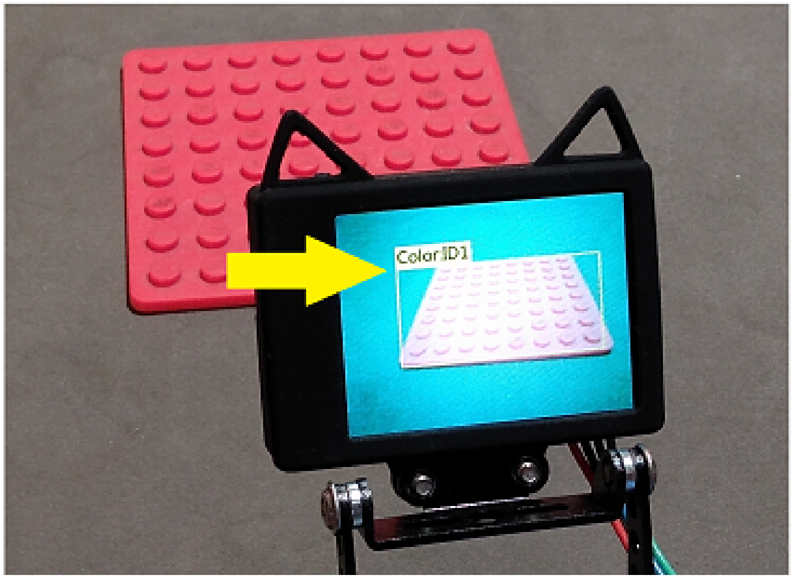


Fig. 282: HuskyLens - RED (Color 1) Trained

Step 5. As prompted on the screen, **short press** the Learning button again (before the 5-second time-out). This prepares for learning the next color.

Step 6. Point the lens at your second color, and repeat the previous Step 4. Namely, **long press** the Learning button, aim and move, then **release** to complete learning that color.

Now **``Color:ID2``** is trained, with its label shown on-screen.

Step 7. As prompted, **short press** the "other" button, the Function button. Or, allow the 5-second time-out to complete. In either case, this completes the multi-color training. All done!

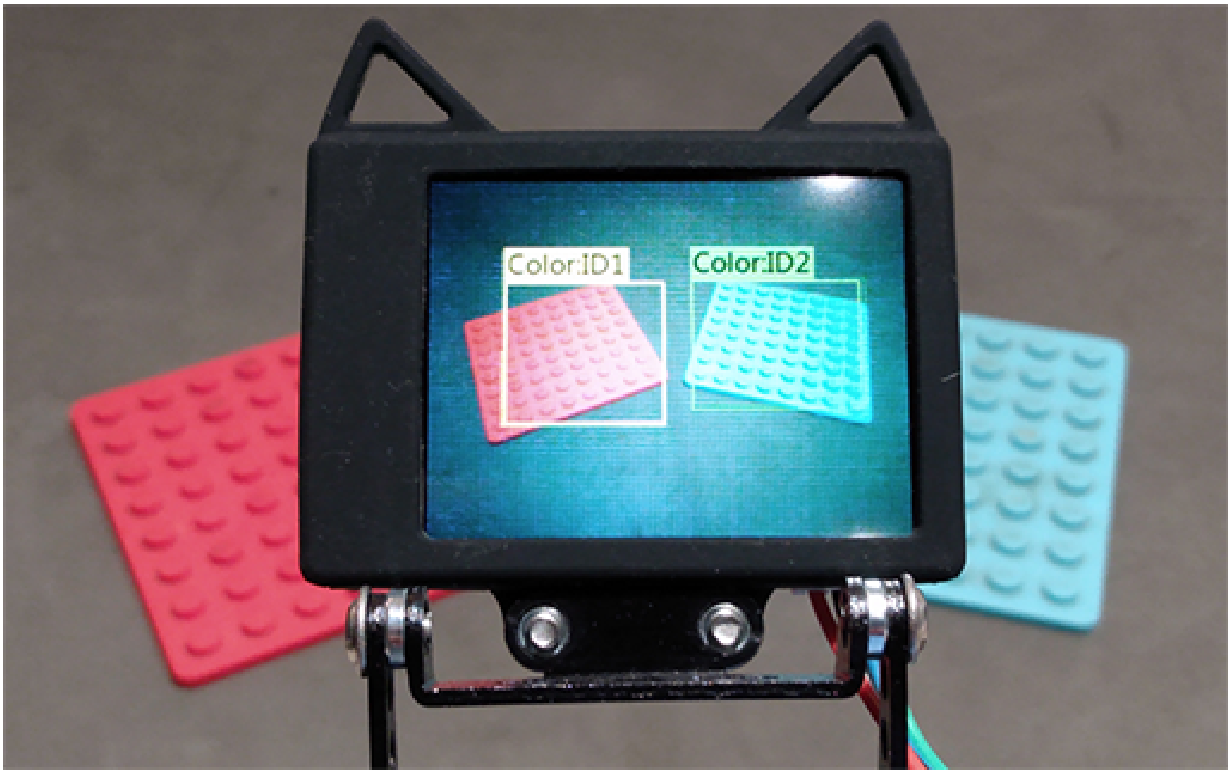


Fig. 283: HuskyLens - Two Colors Trained (ID1 and ID2)

If you want to do all this **over again**, short-press the Learning button, then (as prompted) short-press again to **``Forget`` all of the learned colors**.

This makes the plus-sign "+" icon appear again. Repeat the above, from Step 4, to train colors again.

Multi-Color Detection

For your OpMode code to read **``Color:ID2``**, for example, the Algorithm must be set to `COLOR_RECOGNITION` and the field `HuskyLens.Block.id` will be **the value 2**. This can be seen in the Telemetry portion of the Sample OpMode you used above.

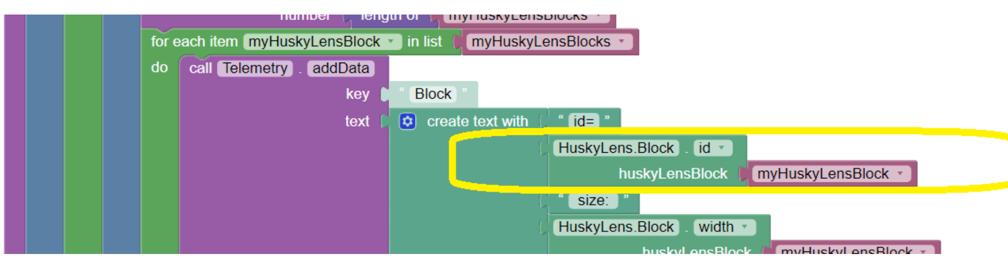


Fig. 284: Adding Telemetry for Colors

Here's the DS Telemetry from the Sample OpMode used above for single color, **with no coding changes**:

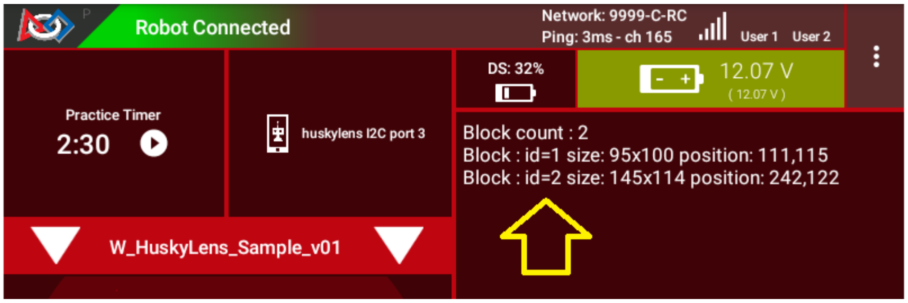


Fig. 285: Example Telemetry showing Both Colors

Now there are two trained and recognized colors, with ID Codes 1 and 2 – see yellow arrow above.

These two lines of Telemetry are generated in different cycles of the same FOR Loop. They display together, since the `Telemetry.update` Block appears **after** the FOR Loop has completed all of its cycles. Namely, the FOR Loop has processed each HuskyLens "color block" in the `List` of HuskyLens "blocks".

In the Java sample OpMode, add these lines **inside the FOR loop**:

```
int thisColorID = blocks[i].id;                      // save the current recognition's Color ID
telemetry.addData("This Color ID", thisColorID);     // display that Color ID
```

Besides `.id`, other Java fields are available for the currently recognized Bounding Box: `.width`, `.height`, `.left`, `.top`, plus `.x` and `.y` (center location).

The color ID numbers are assigned **in order of training**. You cannot renumber these later, so plan your training and OpMode coding to agree with each other.

Tip: Advanced tip: If your color recognition is heavily affected by ambient lighting, you could try training your object in various lighting conditions **as different HuskyLens colors**. Namely, the Red-shade Team Prop could be trained as **``Color:ID1``** in bright light, and trained as **``Color:ID2``** in dim light or shadow. Your OpMode could accept **either** Color ID (1 or 2) as "Red". Likewise, Blue shades could have Color IDs 3 and 4.

Object Training

This tutorial ends with HuskyLens **color training**. Now you are familiar with the basic steps for HuskyLens operation, training, and FTC programming.

You are encouraged to proceed with training the HuskyLens to recognize an **actual object**. This could be one of its 20 pre-trained models ("Object Recognition") or a **custom model or image** that you train ("Object Classification"). In each case, follow a process similar to color training, using the HuskyLens documentation.

You may find that HuskyLens **object recognition** provides more (educational) exposure to the process of AI and Machine Learning, along with more reliable results than color recognition.

Best of luck this season!

Questions, comments and corrections to westsiderobotics@verizon.net

21.8 Additional *FIRST* Website Resources

- FIRST Website Programming Resources Link

Chapter 22

CAD Resources

Computer Aided Design (CAD) and 3D animation software is used in *FIRST* Tech Challenge by teams to design and visualize complex systems prior to manufacturing. There are many software options for CAD and there's no way to list them all. Some software is provided free of charge, some software is provided for a fee, and some require subscriptions. However, many organizations provide free access to "premium" CAD software to *FIRST* teams.

When looking for a CAD package, consider what computers the software will run on. Some CAD packages are desktop solutions and require a fairly competent computer with ample resources to run. Some CAD suppliers also provide cloud-based tools, which move the processing load onto cloud servers - these solutions often merely require an internet-connected computer with a web browser.

Here are a few tools commonly used by teams:

Software for Beginners

- Autodesk TinkerCAD (free) (desktop)
- FreeCAD (free) (desktop)

Software for Intermediate Users

- *Autodesk Fusion 360* (Free to *FIRST* teams) (desktop)
- *Dassault Systemes 3DEXPERIENCE* (Free to *FIRST* teams) (cloud)
- *PTC OnShape* (Free to *FIRST* teams) (Cloud)
- Trimble SketchUp (Free plans available) (desktop)

Software for Professional Users

- *Autodesk Inventor* (Free to *FIRST* teams) (desktop)
- *Dassault Systemes Solidworks* (Free to *FIRST* teams) (desktop)
- *PTC Creo* (Free to *FIRST* teams) (desktop)

22.1 AUTODESK CAD Resources

Autodesk is dedicated to preparing the next generation of tinkerers, makers, designers, engineers, and revolutionizers to lead in the Future of Work. With advanced technologies and workflows accelerating change in industries and careers, we at Autodesk are excited to partner with you on your professional journey. We invest in students by offering our broad portfolio of cloud-based integrated CAD/CAM platform technologies because we believe your ideas and innovation have the power to make this world a better place for everyone. Lead the change and change the world.

22.1.1 Obtaining AUTODESK Software

Autodesk makes software available to all *FIRST* teams via their *FIRST* Education Community FIRST ROBOTICS COMPETITION Page.

Autodesk's desktop-based CAD and Animation tools are world-class and available for download. Users must first create an Educational Access account, verify your account, and then complete your profile to ensure eligibility. Once completed, you'll be provided with one-year education access (renewable). To get started, visit the Autodesk Education Website.

22.1.2 AUTODESK Training Videos

A few resources for training:

- Autodesk Fusion 360 Self-paced Learning Courses
- Autodesk Inventor Self-paced Learning Courses
- Tinkercad Self-paced Learning Courses
- 3ds Max Self-paced Learning Courses

22.2 SOLIDWORKS® CAD Resources

For over 15 years, Dassault Systèmes has been a software supplier to *FIRST®* teams with SOLIDWORKS®. We are also introducing the 3DEXPERIENCE® platform, a technology platform that provides Product Lifecycle Management (PLM), collaboration, community, and Cloud CAD Apps to all *FIRST* teams. Enhance collaborative robot design with your team.

22.2.1 Obtaining SOLIDWORKS® Software

Dassault Systèmes makes select software available to all *FIRST* teams. Dassault Systèmes provides both cloud-based and desktop-based tools. Visit https://www.solidworks.com/product/students/first-robotics-students to learn how to set up an account.

22.2.2 SOLIDWORKS® Training Videos

Learning content for both the cloud-based and desktop-based tools is managed through the 3DS online learning portal, which manages access through your 3DEXPERIENCE ID.

22.3 PTC CAD Resources

PTC is proud to join forces with *FIRST* to empower the engineers and innovators of tomorrow! Through PTC Education, teams can gain free software and services, including Onshape, Creo, Mathcad, Windchill, and Vuforia, easy-to-use training curriculum, and financial grants to select teams. PTC's product development software will enable collaboration, increase efficiency, and enhance accuracy during the robot design process.

22.3.1 Obtaining PTC Software

PTC makes software available to all *FIRST* teams via their *FIRST* PTC Student Download Page.

PTC's cloud-based CAD tool, OnShape, merely requires a FREE Education account available to all *FIRST* teams (each student/mentor on the team needs their own account, mentors/coaches should sign up as an "Educator"). Once you're signed up for a free account, ALL of PTC's OnShape training materials are now available through the OnShape login. PTC also provides access to their desktop-based CAD tool, Creo, free to *FIRST* teams.

22.3.2 PTC Onshape Training Videos

FIRST Tech Challenge specific Training Videos

- Using the FIRST Tech Challenge Library with Onshape

Select PTC Learning Pathway Training Series (available on OnShape login)

- CAD Basics Learning Path Training
- OnShape Fundamentals Training

Chapter 23

Managing Electrostatic Discharge Effects

23.1 Introduction

Electrostatic discharge (ESD) events have the potential to disrupt the normal operation of a competition robot. This section examines causes of ESD events and discusses ways to mitigate the risk that an ESD event will disable or damage a robot's control system.

Note that this section only provides a brief overview of the physical phenomenon that causes ESD disruptions. You can use the following link to view an in-depth white paper, written by Mr. Eric Chin (a FIRST alumnus and a 2018 summer engineering intern), which examines and quantifies the efficacy of various ESD mitigation techniques:

Eric Chin's White Paper on ESD Mitigation Techniques and their Efficacy

Special thanks to Doug Chin, Eric Chin, and Greg Szczeszynski for the work they did to model the problems caused by ESD and to evaluate different techniques to mitigate the risk caused by this phenomenon. Also special thanks to *FIRST* Tech Challenge Teams 2844, 8081, 10523, 10523a, and 10984, and the volunteer team from Arizona (including Robert Garduno, Susan Garduno, Richard Gomez, Matthew Rainey, Christine Sapio, Patricia Strones, and David Thompson) for assisting in testing some of these mitigation techniques under the hot desert sun!

23.2 What is an Electrostatic Discharge Event?

An electrostatic discharge (ESD) event occurs when a highly charged conductive object (like the metal frame of a robot) touches an uncharged or oppositely charged conductive object and discharges to it. Because of the high voltages involved (up to tens of kilovolts), ESD events can produce extremely high electrical currents as the charge that was accumulated on one object flows through a conductive path to the neutral or oppositely charged object.

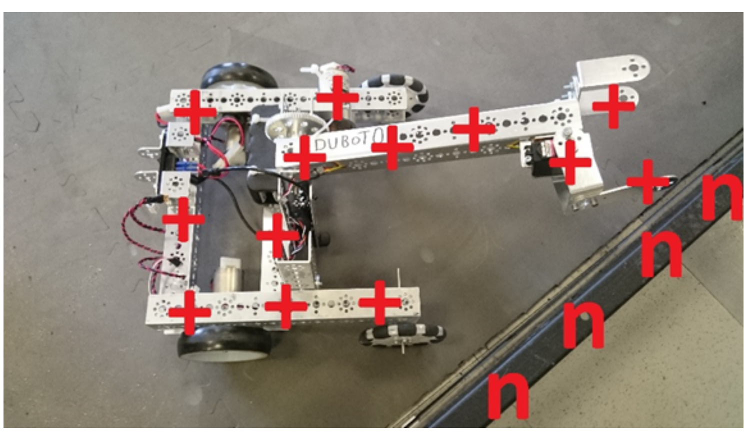


Fig. 1: Positively charged robot next to neutral field wall.

23.3 How Robots Become Charged

Consider what happens when you shuffle your feet on a carpet in wool socks and then touch a door knob. You'll almost certainly get a shock. What causes this phenomenon? When two surfaces interact, there is a small amount of adhesion. This means that they share electrons and if they are made from different materials the electron sharing may be uneven. When the surfaces are taken apart, they can become charged. This is called the triboelectric effect.

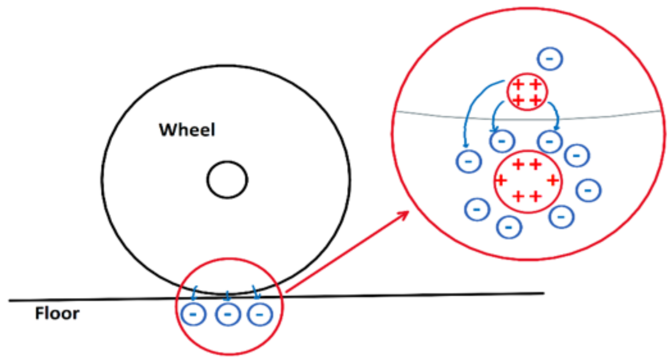


Fig. 2: Robots become charged due to the triboelectric effect.

A robot's wheels moving on field tiles build charge on the robot frame just like your wool socks moving on carpet build charge on your body. Many other plastic and rubber materials behave similarly. It is important to note that triboelectric charging takes charge from one object and gives it to another, so the charges are mirrored. In the case of a *FIRST* Tech Challenge robot, positive charge accumulates on the wheels and negative charge accumulates on the tiles.

Note that a robot with wheels that slide across the soft tiles of a competition field will build electrostatic charge on its frame more rapidly than a robot with wheels that roll across the tiles.

23.4 Discharging a Robot

Current "wants" to flow from objects at higher potential to the objects at lower potential to equalize the voltage difference between them and it will if given a conductive path to do so (like an uninsulated wire). In the case of a robotics competition, if a robot is at a higher potential than another metallic object (such as a portion of the game field), an ESD event will occur if the frame of the charged robot contacts the other object.

If the potential difference is high enough, it is also possible for current to flow through the air in the form of an electrical arc. Arcing occurs when the air between two differently charged conductors becomes ionized and allows current to flow from one conductor to the other. Arcs at voltages seen on FIRST Tech Challenge robots can jump air gaps of more than 3/8" (1 cm). Arcs behave almost like direct contact, so they can carry a significant amount of current. Visible sparks go with large electrostatic arcs.

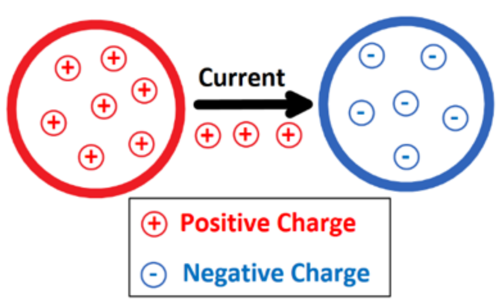


Fig. 3: Electric arc between two spheres of opposite charge.

23.5 What Steps can be Taken to Mitigate the Risk of an ESD Disruption?

23.5.1 Step 1: Treating the Tile Floor with Anti-Static Spray (Event Hosts Only)

One of the most effective ways to reduce the risk of disruption by ESD events is to treat the tile floors of a competition field with anti-static spray. Anti-static spray increases electrical conductivity of the surface of the tiles. This helps prevent the build-up of electrostatic charge on the robots as the move across the tile floor.

FIRST recommends the use of ACL Heavy Duty Staticide spray to treat the tiles. This spray is extremely effective at preventing charge build up on the robots. Also, this spray only needs to be applied once and it will last for an entire event (and it will work across multiple days).

Note that treating the tile floor is something that **only the event host is authorized to do**. Teams are **not permitted** to treat the tile floor themselves.

23.5.2 Step 2: Add Ferrite Chokes to Signal Wires

Ferrite chokes block large changes in current like those seen during an ESD event. This can reduce the risk of damage to or disruption of electrical components when a sensor or other peripheral device receives a shock.

Using ferrite chokes can be a very effective method for mitigating the effects of ESD:

1. Use USB cables that have built-in or snap-on ferrite chokes.
2. Install snap-on ferrite chokes onto your signal cables:
   - Sensor cables
   - Encoder cables
   - Servo cables

23.5.3 Step 3: Electrically Isolating the Electronics from the Metal Frame of the Robot

As a robot moves back and forth across the tile floor during a FIRST Tech Challenge match, charge can accumulate on the metallic frame of the robot due to the triboelectric effect. If a charge builds up on the frame of the robot, but the electronics that make up the Control System are at a different voltage, then a shock can occur if an exposed or poorly insulated portion of the Control System gets close (less than 3/8" or 10mm) to the metal frame.

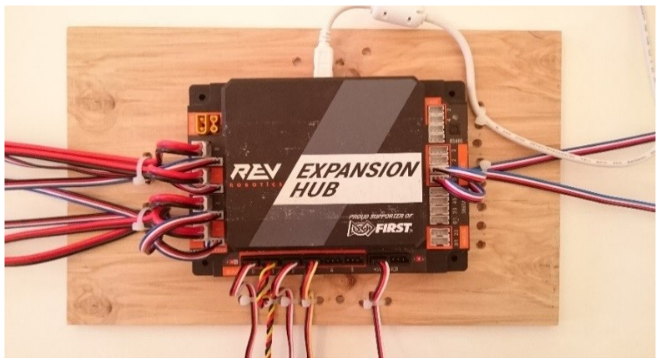
Electrically isolating or insulating the electronics from the frame can help avoid disruptions due to this type of shock.

Sub Step A: Mounting Electronics on a Non-Conductive Material

Mounting the Control System Electronics on a non-conductive material, such as a thin sheet of plywood or a sheet of PVC type A, can help reduce the risk of an ESD event between the frame and the electronics. Using a non-conductive, rigid panel can also help with wire management and strain relieving.


NET CONTENTS: 1 GALLON / 3.75 LITERS
#2002
Heavy Duty
Staticide
Staticide
Kill
Static
on Contact.

Fig. 4: A snap-on ferrite choke.



Sub Step B: Isolate Exposed or Poorly Insulated Parts of the Electronics

Certain parts of the Control System's electronics have exposed metal or are poorly insulated. If these parts are placed too close to the metal frame, a shock can occur if a charge accumulates on the frame.

Fig. 5: Electrostatic shocks can occur at poorly insulated or exposed portions of the electronics.

For example, the 4-wire sensor cables that are used by the REV Robotics Expansion Hub have plastic connectors that are poorly insulated. If a charge accumulates on the metal frame of the robot, and the end of sensor cable is placed close to the frame, a shock can occur and this shock can disrupt or even damage the I2C port of an Expansion Hub.

Similarly, some servo extension cables (see figure above) have exposed portions of metal that could be vulnerable to ESD unless properly isolated or insulated.

Moving these vulnerable areas of the electronics system away from the frame (with an air gap greater than 3/8" or 10mm) can help reduce the risk of an ESD disruption. Using electrical tape to insulate these areas can be equally effective and may be easier to implement.

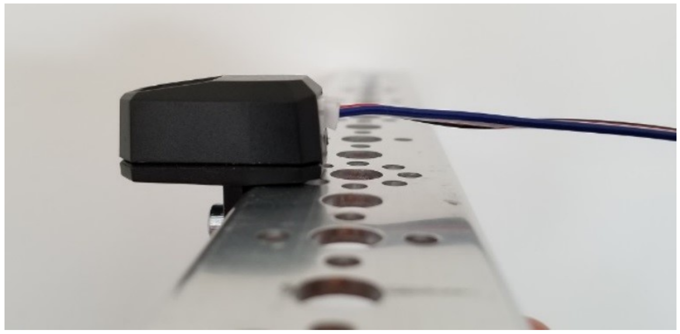


Fig. 6: Keep exposed portions of the electronics more than 3/8” (10mm) away from the frame.

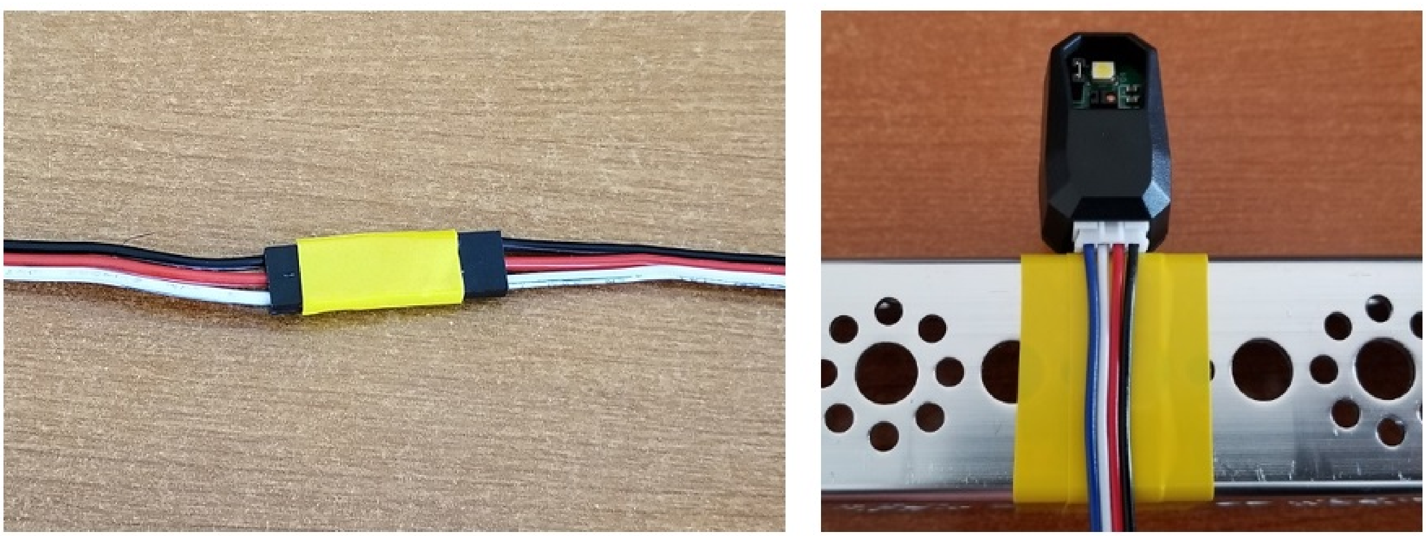


Fig. 7: Electrical tape can be used to insulate exposed or poorly insulated metal.

23.5.4 Step 3: Covering Exterior Metal Features with Electrically Insulated Material

Another ESD mitigation strategy is to cover exposed portions of metallic frame pieces with an electrically insulating material. Covering the conductive exterior parts of a robot with a non-conductive material reduces the risk that they will touch a conductive object at a different electrical potential and trigger an ESD event. Wooden bumpers, electrical tape, and other non-conductive coatings are all effective.

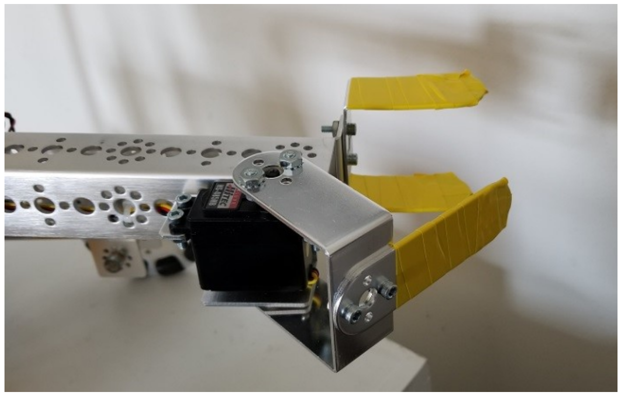


Fig. 8: Insulating portions of the robot that touch other metallic objects on the field can help.

In past seasons, teams who have done this have observed reductions in the frequency and severity of ESD events on their robots.

23.5.5 Step 4: Ground Electronics to Metal Frame with an Approved Cable

Because it is difficult to perfectly isolate the electrical system, it is beneficial to ground the electrical system to the frame of the robot to prevent a potential difference from building up between the frame and the electronics. Doing this can help reduce the risk that a shock can occur between the frame of a robot and the Control System electronics.

It is important that the grounding **only be done using a FIRST-approved, commercially manufactured cable** (i.e., the REV-31-1269 Resistive Grounding Strap). A FIRST-approved cable has an appropriately sized inline resistor. This resistor is critical because it acts as a safeguard to prevent excessive current from flowing through the frame of the robot if a "hot" (positive) wire of the electronics system is inadvertently short circuited to the frame of the robot. Also, the commercially manufactured grounding cable has a keyed connection, which is designed to prevent a user from inadvertently connecting a hot (12V) line to the frame of robot.

Note that if your team uses Anderson Powerpole connectors, then you will need to use the REV Robotics Anderson Powerpole to XT30 Adapter cable in conjunction with REV Robotics' Resistive Grounding Strap:

To ground the electronics, plug one end of the FIRST-approved cable into a spare XT30 port on the Control System electronics. Then bolt the other end using a conductive (i.e., metal) bolt to the frame of the robot.

It might initially seem contradictory to both insulate the electronic components of the control system from the frame and to also ground the electronics to the frame. However, if the electronics are not grounded to the frame, shocks can occur if a charge builds on the robot frame and an exposed or poorly insulated portion of the electronics (such as the base of a REV Robotics color sensor) gets close it. If the electronics are grounded to the frame, the grounding wire helps keep the electronics at the same potential as the frame, preventing arcs between the two systems.

Fig. 9: The REV Resistive Grounding Strap (REV-31-1269) is an approved grounding cable.

Fig. 10: The REV-31-1385 adapter is approved for use with REV’s Resistive Grounding Strap.

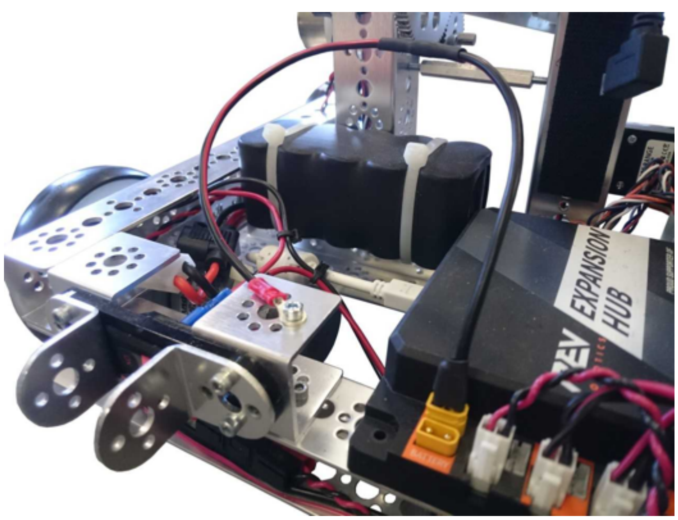


Fig. 11: Ground the electronics to the frame using a FIRST-approved cable.

Chapter 24

Manufacturing

Manufacturing is an extremely important part of building a robot for the FIRST Tech Challenge. While Commercial-Off-The-Shelf parts can be used for the majority, and in some cases the entirety, of a robot, there are many times when a custom part is needed. This section will cover some of the most common manufacturing methods used in FTC for creating these parts. It is important to note that there are many different ways to manufacture parts, and this section will only contain those that are most commonly used.

While a CAD model is not always needed to build a robot or part, it is often required when manufacturing custom parts. For this reason, it is recommended that a team design their robot (or at least the part that needs to be made) in a CAD software. For more information on CAD, see the *CAD Resources* section.

Here are some manufacturing methods that are commonly used in FTC:

- Prototyping (Proof-of-Concept, Testing) (Cardboard, Wood, Foam, etc.)
- *3D Printing* (Smaller parts, Prototyping) (Plastics)
- Machining (Larger Parts, Plates) (Metal, Wood, Composites, Plastics)
- Laser Cutting (Plates, Gears, etc.) (Wood, Composites, Plastics)

24.1 3D Printing

Many parts in FTC need to be a special/unique shape and size, one that isn't sold or available from a vendor. Sometimes, teams need a part that is impossible to machine or cut out, or needs to be lightweight. Other times, teams may want to test and iterate the design of a part rapidly and cheaply. 3D printing is a great solution to all of these problems.

3D Printing is the process of creating a three dimensional object by laying down successive layers of material (typically plastic) from a digital model file. To do this, machines known as 3D Printers are employed which melt and place plastic in order to produce this model.

In order to start 3D printing, teams need to understand how the technology works, what printers are available to buy, how to properly set up a part to be printed, and what materials are available to print with.

24.1.1 3D Printing Introduction

Fig. 1: 3D printed dead wheel odometry parts

3D Printing Methods

There are numerous kinds of 3D printing, but for FTC there are only a few that are practical. The most common is **Fused Deposition Modeling (FDM)**. FDM printers melt a plastic filament and extrude it through a nozzle, which moves around to create the part. FDM printers are the most common type of printer used, and the most practical for robotics teams, so this guide will focus on them.

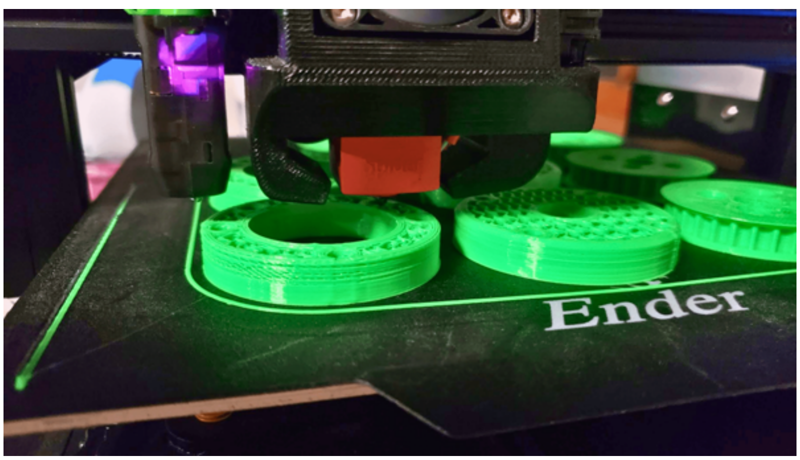


Fig. 2: FDM printer printing a part

Other types of printers include SLA (Stereolithography) and SLS (Selective Laser Sintering), however these are often more expensive, difficult to use, and have less use in FTC, although they have a few specific niches, such as very high precision details.

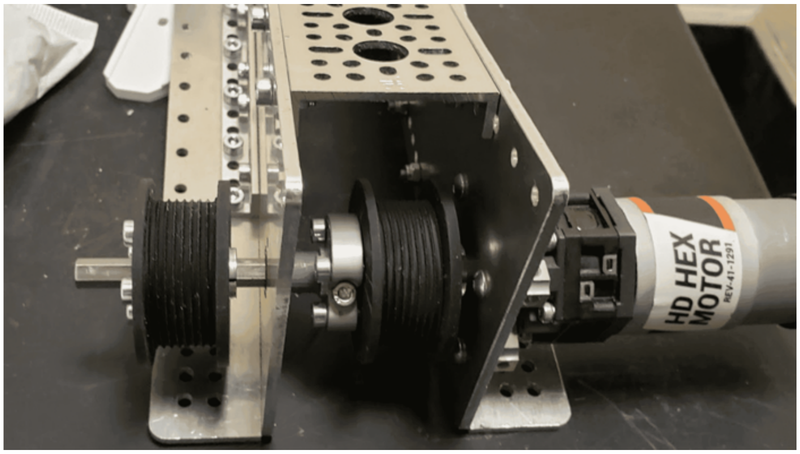


Fig. 3: SLA Resin printed lift string spools

3D Printing Pros

- 3D printed parts can be infinitely customized and optimized for a specific purpose. This can be used to create pulleys or gears with a specific number of teeth, or a part that fits perfectly in a specific place.
- 3D printing can be used to create parts that adapt between different build systems or standards. Many build systems contain their own standards for mounting holes, shaft sizes, or other dimensions. 3D printing can be used to create parts that adapt between these standards.
- 3D printing can be used to make parts relatively quickly and cheaply. This is especially useful for prototyping new designs, or iterating on a design to make it better at little cost and in a short amount of time.

3D Printing Cons

- 3D printed parts are often not as strong as machined or cut parts. This is especially true for FDM printers, which have a layer-by-layer structure that can be a weak point if the part is loaded in a certain way. If consideration is given to this weakness when designing the part, however, the result can be made very strong.
- A 3D printed part can only be as large as the print bed it is printed on. This means that large parts may need to be printed in multiple pieces and assembled later.
- 3D printing can be slow, especially for large parts. Longer prints can take hours or even days to complete, raising the risk of a print failing and wasting time and material.
- 3D printing can be expensive. The cost of a printer, filament, and other materials can add up quickly. However, the cost of a 3D printer has been decreasing rapidly, and filament is relatively cheap.

24.1.2 Example 3D Printed Parts

Here are some example parts that your team could make with a 3d printer to either save funds or improve on customizability.

Mounting Brackets

One of the most common uses, 3D Prints can be used to make mounting brackets for motors, servos, electrical parts, bearings, and various other objects. This provides teams with a great control of precision over how they mount things, and a decreased part count over commercial parts.

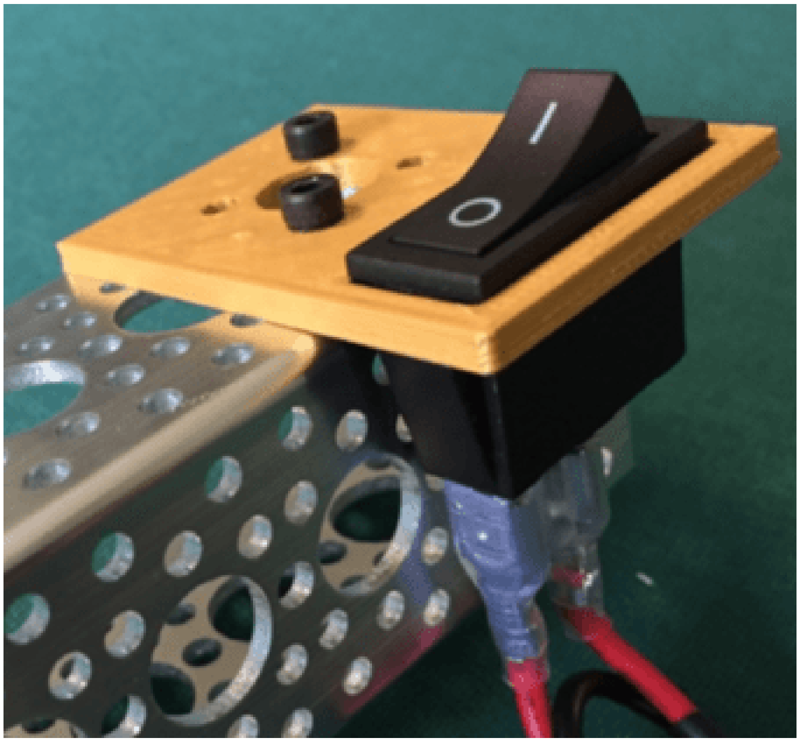


Fig. 4: A Power Switch Mounting Bracket.

Pulleys and Gears

3D Prints can also be used to make your own pulleys and gears. Not only is this a great option for achieving optimal speed and torque ratios, but they save cost as well! A metal pulley typically costs around $10, while a printed one can be as low as 20 cents.

Spacers and Shims

Another common way to utilize 3D prints is to create spacers and shims to constrain objects on your robot, this is both lighter, simpler, and most cost effective (although not always preferred!) than using collars or clamping mounts.

Fig. 5: An example Deadaxle Mecanum setup with a custom 3d printed pulley.

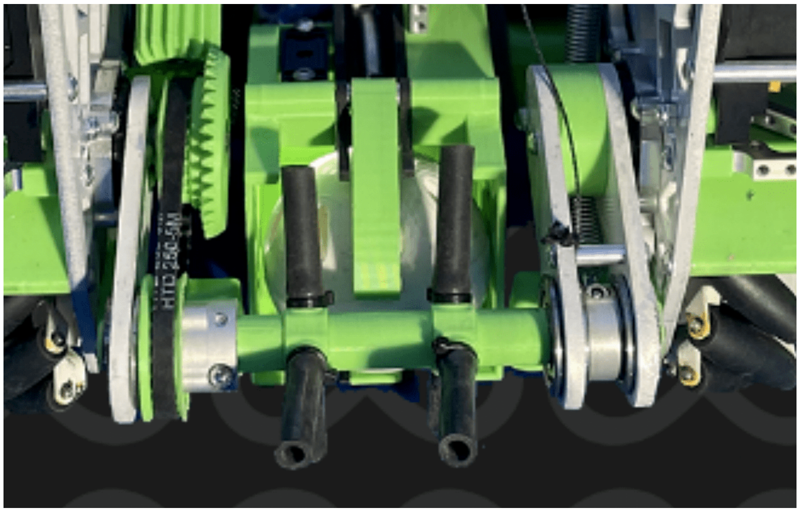


Fig. 6: An FTC FREIGHT FRENZY intake utilizing 3d printed spacers to space out intake wheels.

Scoring Mechanisms

Teams also often use 3D Prints to precisely grip and control each year's game elements. A common way to do this is a custom shaped claw.

Fig. 7: Rendered FTC ULTIMATE GOAL Wobble Goal arm with 3d printed claw parts.

Another common method of using 3D Prints is for creating custom intakes, primarily surgical tubing.

Fig. 8: Rendered example surgical tubing mounts for an FTC FREIGHT FRENZY intake.

Robot Aesthetics

3D Prints can do a whole ton for a robot's aesthetics as well. While it's more advanced, multicolor printing like shown below is a great option for teams that like making their robots look good!

Fig. 9: This FTC Team printed their sponsors logos in multiple colors to represent them!

24.1.3 General Knowledge

The following section is general knowledge regarding the operation, maintenance, and terminology regarding 3D Printing. We'd highly suggest reading through at least the terminology section before getting into the more advanced knowledge documented here!

3d Printing Terminology

3D Printing terminology can often feel overwhelming and complex, to simplify here's a list of some terms to know!

General 3D Printing Terms

- **Additive Manufacturing**: The method of creating a part by building material layer by layer from a CAD model (more commonly known as 3D Printing)
- **Fused Deposition Modeling (FDM)**: The most common type of 3D Printing where objects are produced by laying plastics layer by layer on a bed utilizing a heated nozzle.
- **Stereolithography (SLA)**: A slightly less common type of 3D Printing which uses UV screens to cure plastic resin onto a bed. While more precise, materials are generally weaker and prints require a lot more post processing (hence the lack in FTC).
- **Selective Laser Sintering (SLS)**: One of the least common types of 3D printing, you likely won't come across one of these machines in your FIRST career due to their cost. SLS machines utilize concentrated lasers to bind together patterns in material powder, eventually resulting in a complete part.
- **Filament**: Plastics made of polymer resins that become soft above a certain temperature and harden when they cool. Common filaments include PLA, PET-G, ABS, Nylon, and Acrylic. These generally come in continuous rolls that you feed into your printer.
- **Direct Drive**: A type of FDM extruder where the extruder motor is placed on the carriage together with the hotend assembly, shortening the distance that filament needs to travel significantly.
- **Bowden**: A type of FDM extruder where the extruder is placed at a different spot on the printer separate from the carriage and runs filament through a long PTFE tube to reach the hotend.

3D Printer Parts

- **Carriage**: The moving head of a printer that contains the hotend assembly.
- **Bed**: The heated (or sometimes nonheated) surface that filament is laid upon.
- **Hotend**: The part that melts filament it typically consists of a heat sink, heater, thermistor, and metal nozzle.
- **Gantry**: The frame structure that supports the printing head/hotend as it moves.
- **Stepper Motor**: The small motors that move each axis of the printer with precision.
- **Extruder**: The stepper motor that moves filament into and out of the hotend, whether from up close or far away.
- **Drive Gear**: The gear located on the extruder stepper that grips and moves the filament. Extruders can be either single drive gear or dual drive gear, with dual having more grip.
- **Bowden Tube**: The slippery tube that is located on your printer that allows filament to pass through it. They are made of PTFE, a low friction compound that can degrade at higher temperatures.
- **Axes**: A 3D printer has 3 axes, with the X and Y typically being interchangeable and the Z axis being up and down. There will normally be one or more steppers controlling each of these axes.
- **Hotend Cooling Fan**: All hotends have a heatsink around them to dissipate heat and prevent it from interfering with other printer functions ("heat creep" is a common issue with poor hotend cooling, in which filament crumples upon itself instead of being pushed through the extruder). The hotend cooling fan is directed at the hotend heat sink in order to prevent issues and help further dissapate heat.
- **Part Cooling Fan**: Most printers also have a fan directed below the nozzle to cool parts as they're printed. Some filaments don't print very cleanly without cooling, drooping over themselves, so this is a very important part of the printer for print quality. PLA is easily one of the most volatile materials when it comes to cooling.

Design Terms

- **Tolerances**: Formally described as "the permissible limit of variation", saying that this is how much we expect parts to vary in size due to the inconsistency of manufacturing.
- **Pressfit/Interference Fit**: The tolerance at which pieces will be able to pressed together and stay together, whether that be a bearing in a hole or two 3D Printed parts snapping together.
- **Throughhole/Thruhole/Slipfit**: The tolerance at which a piece can freely pass through another, for example being able to drop an M4 screw into a hole with little to no resistance.

Common 3D Printing Tools

Allen Keys

Allen keys are essential for operating your printer, even if you don't plan on upgrading, regular maintenance and repairs will require these. It's suggested that you buy high quality allen keys that won't strip your screws from the start (this goes for other tools as well!). The most common sizes for allen keys on today's printers are **2.0mm and 2.5mm** but we'd recommend stocking other sizes as well.

Screwdrivers

Screwdrivers are another tool to keep handy for repairing your printer especially for operating in spots hard to reach with allen keys and for screws other than hex key. It's suggested to get a kit with many small bits such as the one pictured below for easy maintenance, unless you already know exactly what type and size screws are utilized on all your 3d printers.

Flush Cutter

Flush cutters, otherwise known as diagonal cutters, snips, or snippers by teams, are an extremely versatile tool that can be used for cutting and trimming things on your printer or your prints. A cheap pair like pictured below is still great for 3D Printing and general use, just make sure that they stay sharp!

PTFE Cutter

Many people think that they can just cut PTFE tube with scissors or flush cutters, but fail to realize the downsides of this. The compression of scissors or flush cutters can compress the sides of the tube or deform it, and when some PTFE tubes have little room for error, this can make a tube unusable. Specialized PTFE tube cutters like the one pictured below can make sure that one of the most important parts of filament transport on your printer is good quality. The one pictured below is clamped onto a PTFE tube, spun around it a few times, and then the cut will be complete with little effort.

Calipers

Calipers are a precision measuring tool used to measure distances often down the the hundredth of a millimeter. These are incredible useful for dialing in pressfits and slipfits, ensuring dimensional accuracy, and tuning your printer. We'd recommend getting a quality pair with solid reviews that will last you a while and remain precise.

Files/Sandpaper

Files and sandpaper are great for getting a nice surface finish and potentially modifying prints if they didn't initially fit your usecase. Oftentimes, quickly filing a part down can save you loads of print time so it's great to have some on hand.

Putty Knives

At its core, a putty knife can basically be used as a spatula to pull 3D prints off the bed. This is a must have for print removal to avoid touching the bed while it's still hot and to give yourself some extra leverage on those prints that are really stuck.

Electrical Tools

Crimpers, a soldering iron, and a good wire stripper are very important to have on hand if you plan modifying your 3D Printer or replacing electrical components. The ability to make your own wires is invaluable, and can often save you a lot of time from ordering materials.


SUNEX

Common 3D Printing Upgrades

A pretty common thing to do with 3D Printers is upgrade them over time, here's a few of the most common upgrades for any printer.

Removeable/Flexible Beds

A lot of printers come with cheap "BuildTak" (typical stickers) beds that work just fine, but some people desire more removeable prints with better adhesion and more customizability, which is fulfilled by removable beds. WhamBam, Bambulabs, and generic Amazon sellers will typically carry these beds for various prices. You'll see some terms which we'll explain here. PEI is the "gold standard" nowadays, simply being a sheet of material that sticks well to 3D prints when heated. More advanced options includev PEX, which expands when heated and holds onto 3D prints better, and then as it cools completely releases the parts. Powder-Coated PEI/Textured PEI are by far the most popular flexible beds now, being cheap, simple, more resistant to damage than PEI or PEX flexible beds, and having exceptional print adhesion.

Glass Beds

Glass beds are also a great upgrade, they stay flat and last a long time. They also tend to be a lot cheaper than flex beds! A glass bed combined with something like glue stick or hairspray works really well to keep prints secure. Most glass beds found on Amazon will work, look for the keywords Carborundum or Borosilicate when looking for glass beds for your printer size.

Hotend Upgrades

A hotend upgrade can be very beneficial for printing with different filaments, speeding up prints, and general reliability. Learning how to mount hotends often takes some CAD skills or a quick Thingiverse search, but hotend upgrades are very worth it. Typically, you will need a hotend structure, heater wire, and thermistor to do a full upgrade (and some wiring tools to connect it to your board). Here are a few hotends from different price ranges that are well regarded and used.

- **Low End:** E3D V6, All-Metal Microswiss Hotend.
- **Mid Range:** Phaetus Dragonfly, Creality Spider.
- **High End:** Phaetus Dragon HF, Slice Engineering Mosquito, E3D Revo.
- **Speed Printing/Engineering:** These hotends are not for the faint of heart and typically require designing custom mounts. They are also typically in a very high price point. These include the Mosquito Magnum, Phaetus Dragon UHF, the Goliath, and the Nova.

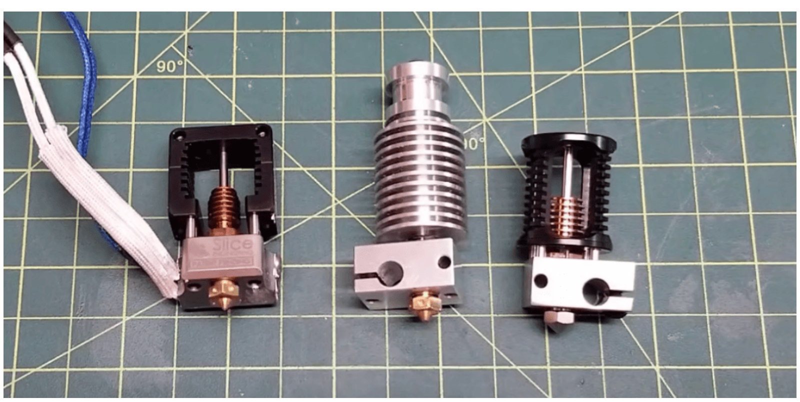


Fig. 10: Left to Right: Slice Engineering Mosquito, E3D V6, Phaetus Dragon

Note: A quick way to increase your hotend's flow rate without breaking the bank and buying a new hotend is by buying and utilizing something called a **CHT Nozzle**. These nozzles split filament flow into 3 parts, allowing each section to melt faster, and resulting in a considerably higher flow rate of plastic.

Fig. 11: This Bondtech CHT nozzle splits flow into three.

Silent Stepper Boards

Older printers or low end printers tend to be very loud stock, which is why silent stepper boards are a great upgrade. These boards use more advanced stepper drivers to smooth inputs, causing steppers to make less whining and whirring noises. For most printers you will have to do some research for the best replacement board for you, but for base model Ender 3 printers, the main compatible boards are the SKR Mini E3 and the SKR Mini Turbo, both of which are drop-in replacements and take no time at all. Luckily, most printers nowadays include this option stock, so it's not a concern for many.

ABL or Auto-Bed Leveling uses either a mechanical or inductive sensor on your toolhead to probe your bed in different locations and uses software to improve your first layer quality and adhesion. While it requires learning a bit about firmware, auto bed leveling is extremely worth it. More and more printers are coming with auto bed leveling stock, butcif yours didn't and you'd like to upgrade, these options are common:

- Mechanical Sensors: BLTouch, CRTouch
- Inductive Sensors: Omron TL-Q5MC2-Z, Pinda Inductive Probes

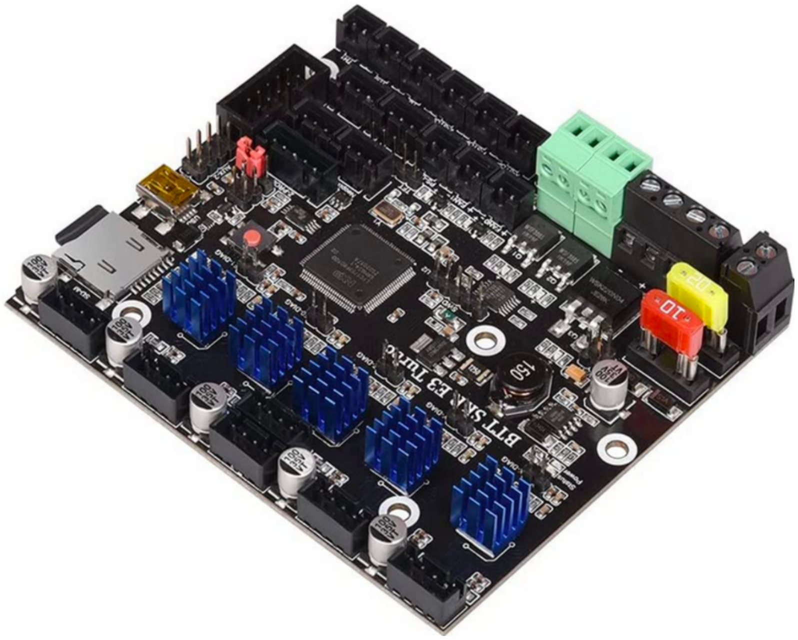


Fig. 12: An SKR e3 Turbo with TMC2209 Stepper Drivers, the current standard for silent printing.

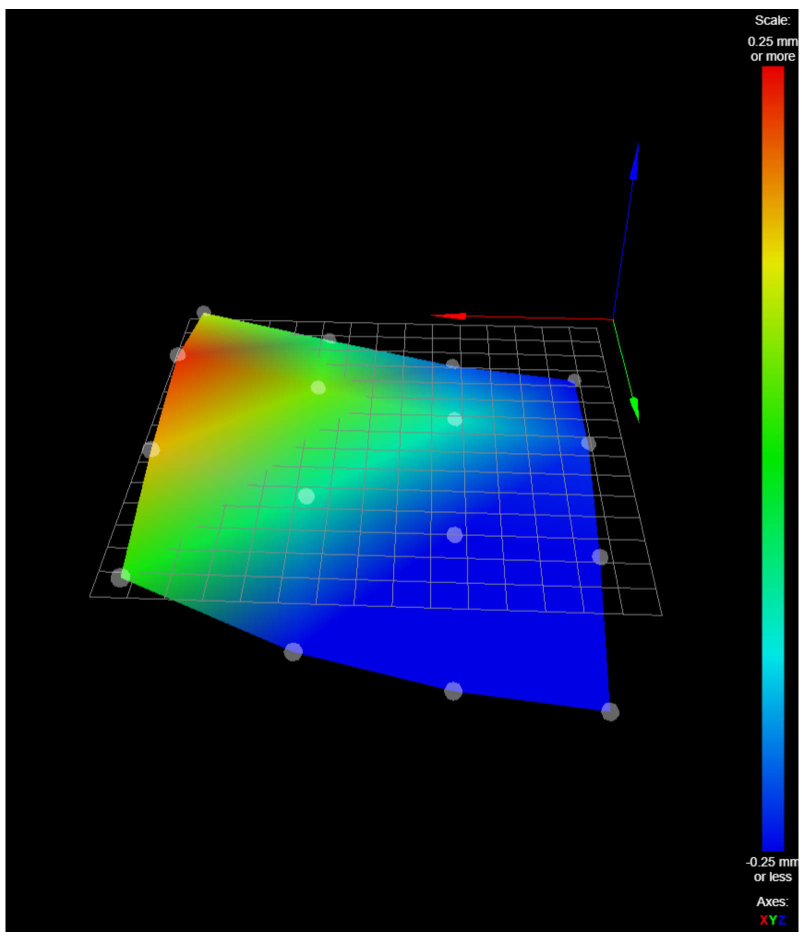


Fig. 13: An example autoleveling reading, showing the imperfections in the bed that the printer will compensate for.

Raspberry Pi/Network Functionality

Network functionality is becoming frequent in 3D printing, with many interfaces allowing you to interact with your printer remotely, and stop/start/watch prints while you aren't even there. Many companies have begun including this feature stock with newer printers, however, even if you have a printer without network functionality, using something such as a Raspberry Pi or old android phone, you can make quick work of this feature using various online guides. If you're using a printer with **Marlin** firmware (you can read through your printer's documentation or sale postings to check) you should research **Octoprint** setup guides. If you are using a printer with the **Klipper** firmware, you should research **Fluidd** and **Mainsail** setup guides.

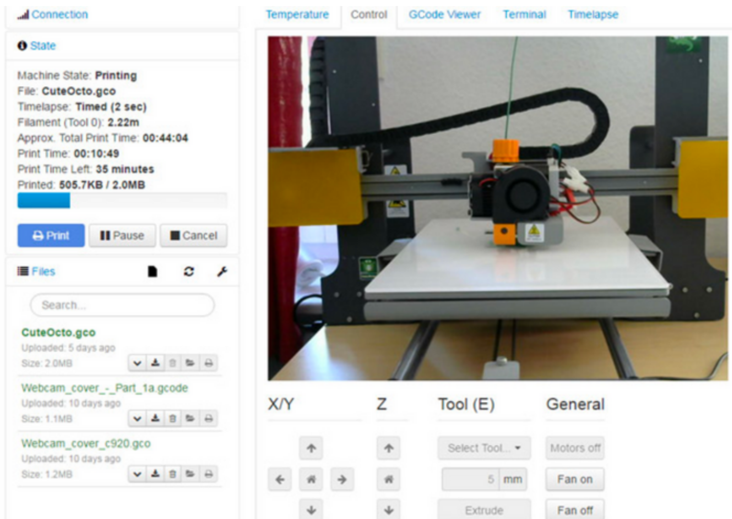


Fig. 14: An Octoprint Control Panel for a Prusa

Dual Z-Axis Support

A common issue people have with Ender 3's and other budget printers is the droop of the Z gantry if it's only driven by a lead screw on one side. This directly results in inconsistencies in prints due to a less stable frame. A fairly common solution to this is to add a lead screw on the other side, evening it out. This requires a board upgrade and an extra stepper. There are plenty of kits for this, or you could attempt to DIY it yourself.

Part Cooling Upgrades

As talked about in Terminology, hotend and part cooling are vital to part quality and preventing jamming issues. If you want to print PLA at higher speeds, it's recommended to upgrade your cooling fans to be larger and more efficient. 5015 and 4020 fans are a great upgrade from stock cooling, and can handle most of what's thrown at them. There are plenty of mounts available for these fans if you search up your printer model and the desired fan size (fan sizes are given in width-depth, so a 5015 comes in a 50mm circle that is 15mm thick) If you want a cooling upgrade with a lot of science behind it, take a look at Voron Design's **AB-BN** or **Stealthburner** projects, cooling systems that were completely engineered with air flow simulation.

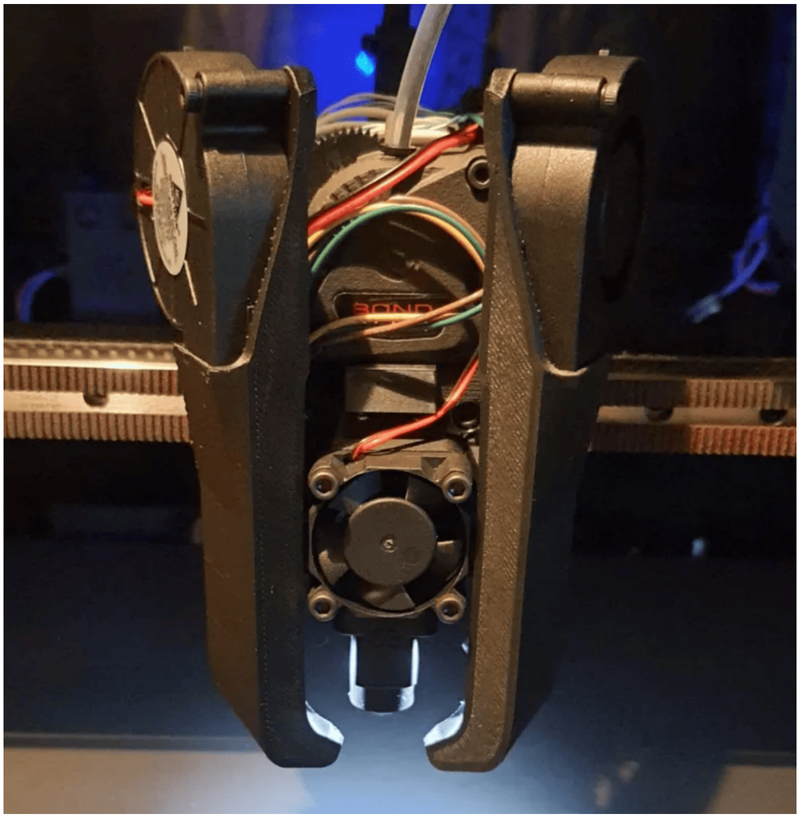


Fig. 15: An example cooling setup with dual 5015 fans (Mantis).

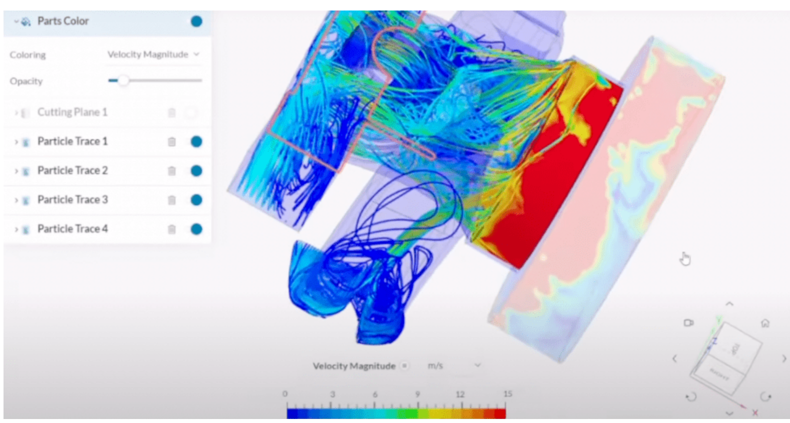


Fig. 16: Simulations of how air runs through a hotend cooling setup (Voron).

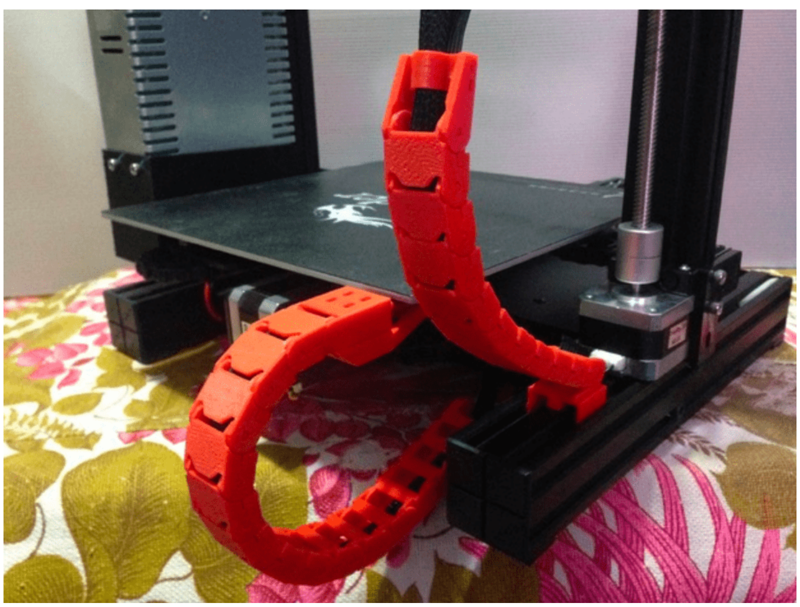
Linear Rails

A lot of 3D Printers use V-wheels for linear motion out of the box, however these can degrade, flex, and lose dimensional accuracy fairly easily. Steel linear rails helps make your carriage a lot more stable and unlocks the ability to print much faster. While this upgrade can be very valuable, it's important to keep in mind that this is one of the most expensive upgrades here, as good quality linear rails can cost upwards of 40$ apiece.

Fig. 17: This is a MGN12 (12mm wide MGN style linear rail) being used for the x axis of a printer.

Cable Drag Chain and Wire Management

To avoid entanglement and fatigue of wires, a lot of people choose to add drag chain to their printers to guide their movement. These chains can either be printed or purchased from a vendor like IGUS/Digikey, so if you're interested in wire management, either shop around or take a look around on Printables/Thingiverse and find a set for your printer model.



Miscellaneous Printed Upgrades

The 3D Printing community is heavily into modifying their printers…using their printers. There are a plethora of upgrades that you can just print yourself! Many are for aesthetics like V-Slot covers and LED lights, but others can improve functionality such as filament guides and belt tensioners. These upgrades are definitely worth exploring if you have some downtime!

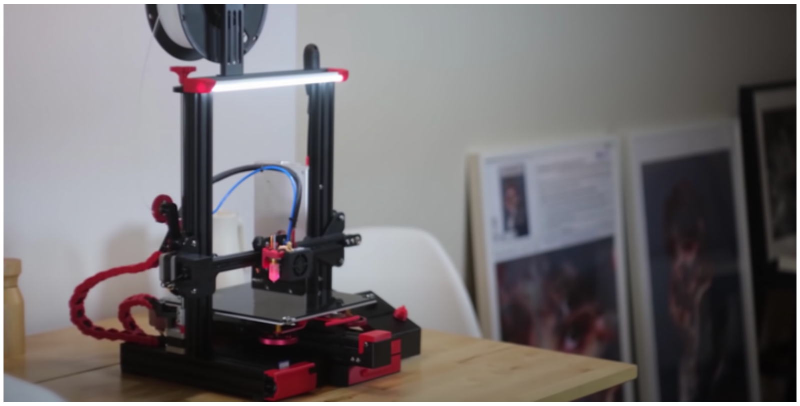


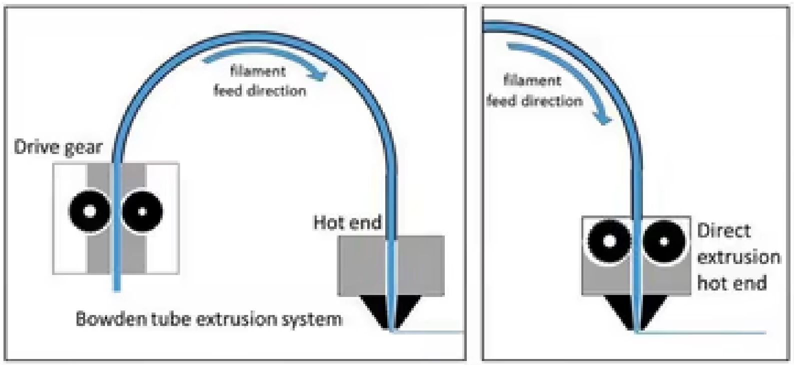
Fig. 18: Scott Yu-Jan's heavily upgraded Ender 3

Hardware Tradeoffs

Note: While these tradeoffs are important to consider they will not make or break your ability to print parts for most applications. Even the most basic entry printers will work well for FTC but if your team has specific needs (printing flexible materials for instance) it's recommended to look through these hardware tradeoffs for education.

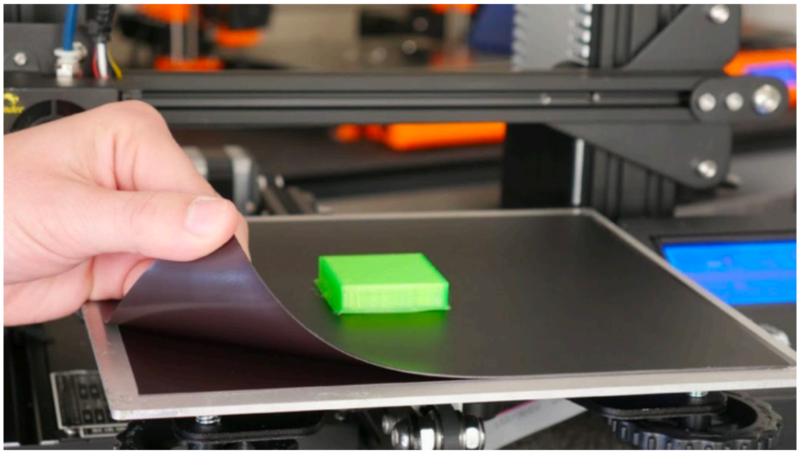
Direct Drive vs Bowden:

Direct drive extrusion systems have a couple advantages over Bowden, most notably the ability to print with a wider range of materials (particularly flexibles) as well as more consistent extrusion and retraction. However Bowden printers are almost always cheaper so we'd recommend evaluating what kinds of printing your team needs and go from there.



Glass vs Flexible Beds

Glass and flexibles beds are both common choices. Glass beds will usually last longer, are easier to maintain, and are more resistant to print head damage. Good Glass beds are also typically cheaper. Flex beds make prints easier to take off and typically have better first layer adhesion, but will run you a premium on cost and are relatively easy to damage or scrape.

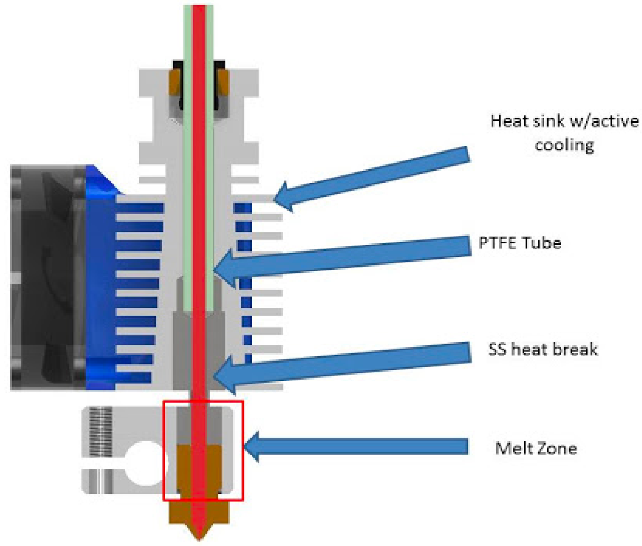


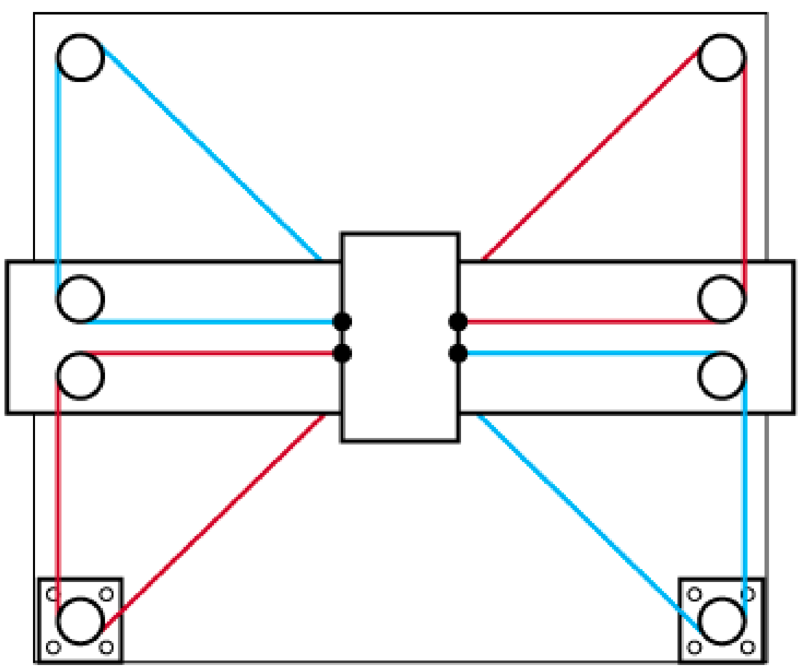
PTFE Lined vs All-Metal Hotends

PTFE tubing is a common low friction tubing used in 3d printing. PTFE lined hotends have a section of this tubing that goes right up to the heated area. These are typically the cheaper option, but it is not recommend to use them whatsoever if you plan on 3d printing anything beyond PLA/PETG. PTFE at temperatures over standard printing temps (normally ~250C is the limit) can "off-gas", putting off dangerous VOCs (Volatile Organic Compounds). All-Metal hotends are more expensive, but remove this dangerous PTFE tube placement. Safety should always be your top priority, so look at All-Metal as long as you're planning on printing at higher temperatures. Notably, Ender series printers come stock with a PTFE lined hotend, so buying all-metal is one of the large upgrades that many do to their Ender 3s.

Cartesian vs CoreXY

Cartesian motion 3D printers, otherwise known colloquially as "bed-slingers" have been the standard for most consumer level printers and are practically everywhere, with one stepper controlling each axis, and a moving bed with not much complexity. An alternate form of 3d printer movement taking the is called CoreXY. These methods of control use a differential to control both X and Y axes with variable quantities of energy from 2 motors. This increases power and speed while decreasing gantry weight. Some of the most notable CoreXY printers include Vorons, BambuLab printers, and the Creality K1. Due to a lot of engineering effort, CoreXY is now considered the faster of the two kinematic, and is recommended to increase manufacturing speed. Cartesian printers are more tried and tested however, and Cartesian printers like the Prusa mk3, Prusa mk4, and Ender 3 are a better option if you desire the incredible resources and consistency behind them.


Heat sink w/active cooling
PTFE Tube
SS heat break
Melt Zone



24.1.4 Printer Choice

This section will focus on printers from each budget range that could be useful for FTC teams. There will be many features listed and drawbacks listed for every budget range, but there is one feature that we would like to note that teams should not buy a printer without note of it being included.

Thermal Runaway Protection: This is a feature where if a thermistor on the printer disagrees with the heater input and temperature trends don't make sense, the 3D printer will shut itself down. This is essential to prevent possible fires and teams should not buy 3D printers without this feature. Most 3D printers you buy today will have this due to firmware updates, despite the age of the printer, but it should still be checked.

Please search up whether the printer model you intend on buying has this feature. If you search up Ender 3s, you will find some results that say it does not, but this is dated information and not true, as Ender 3s are now shipped with that enabled in the firmware by default. All other printers that we list in our sections below are modern enough to have this feature as well.

Budget Printers (Under $300 USD)

Note: Just because you buy a budget printer doesn't mean you can't upgrade it later to be even better! (Take a look at Common Upgrades)

Creality Ender-3/Pro/V2 ($100-$250)

If you're looking for the most cost effective printer that will still do a great job, this is a great option. The Ender-3 series is an open source classic in the FTC and 3D printing community. It has a huge support network and despite it's low cost, has proven itself to be a very capable printer.

Ender 3 Features

All Ender 3s have:

- A huge support network
- Tons of printable and purchasable upgrades
- Open Source Hardware
- 220mm x 220mm x 250mm Print Volume

The Ender 3 Pro has:

- A more stable Y axis
- A more powerful power supply
- A flexible bed

The Ender 3 v2 has:

- A glass bed
- Built in belt tensioners
- The power supply and Y axis from the Pro

Note: The V2 and Pro can often be found on sale for the same price as the base model, making them very good options.

Ender 3 Drawbacks

- Technology is dated due to initial release in 2018 (Lack of Auto Bed Leveling)
- PTFE Lined Hotend
- Bowden Style Extruder
- V-Roller Motion system and singular lead screw setup causes frame flexing.
- Can take a bit of tweaking/upgrading to work consistently.

Fig. 19: Creality Ender 3 Base Model

Sovol SV06 ($259)

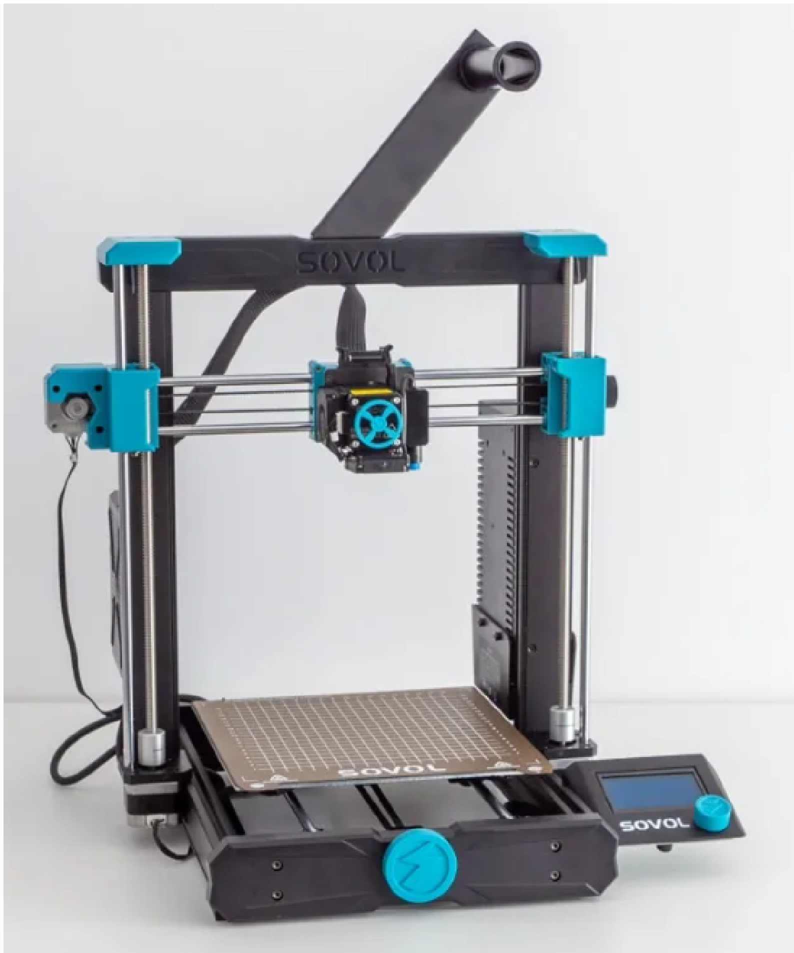
If you're willing to spend just a little bit more money for more quality of life features out of the box, the Sovol SV06 is a good option. It maintains an Ender-like frame but adds in modern features that the Ender lacks which can save tinkering and maintenance time down the road.

SV06 Features

- Auto Bed Leveling
- Flexible Bed
- All-Metal Direct Drive Extruder
- Built in Belt Tensioners
- Dual Z-Axis
- 220mm x 220mm x 250mm Print Volume

SV06 Drawbacks

- Known Quality Control Issues from factory (X-Axis not flat)
- Not many slicer profiles available (Slowly being solved)



Other Budget Printers/Ender Clones ($150-$300)

If you're ok with forgoing the huge knowledge base behind the Ender-3 series in exchange for a few more features out of the box, some Ender-3 clones can be a good option. Notable ones include the Elegoo Neptune, Anycubic Vyper, and Voxelab Aquila. It's notable that while these are more or less "Ender 3 Clones", all three companies and printer models listed here are still established companies with community trust and acceptable customer service. These printers can be a great buy and are often cheaper but make sure to do research before purchasing.

Common Features

Ender 3 Clones typically have at least a couple of the following…

- Auto Bed Leveling
- Upgraded Print Surface
- Built in Belt Tensioners
- Colored Touchscreen

Common Drawbacks

- Less troubleshooting help/knowledge base compared to the Ender 3
- Any drawbacks of the individual printer. Make sure you keep an eye out for things you want in a printer when researching.

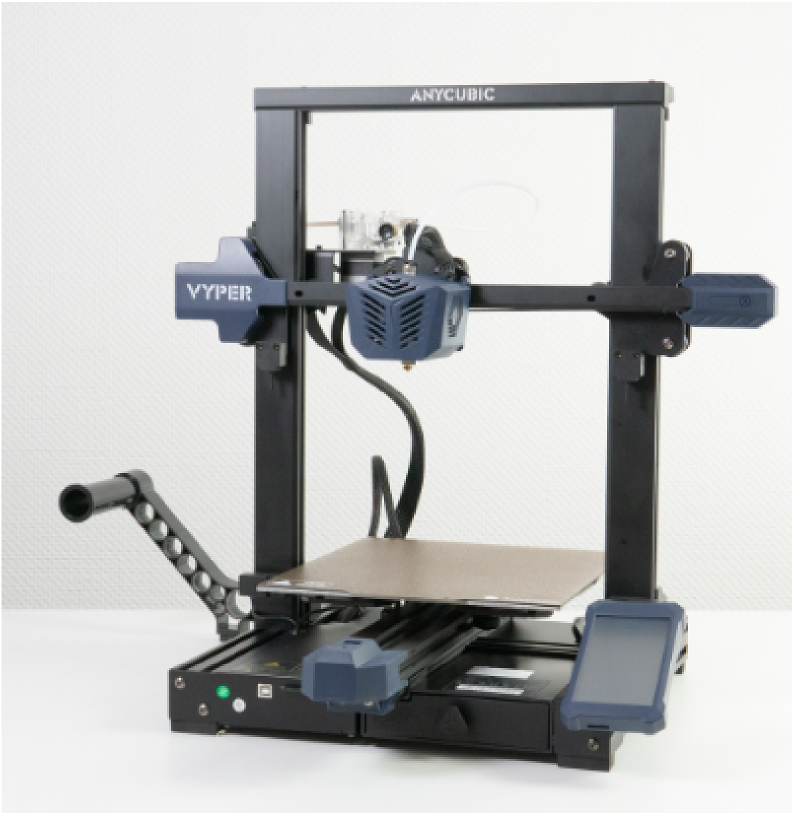


Fig. 20: Anycubic's Viper

Mid-Range Printers ($300-$600 USD)

AnkerMake M5C ($399)

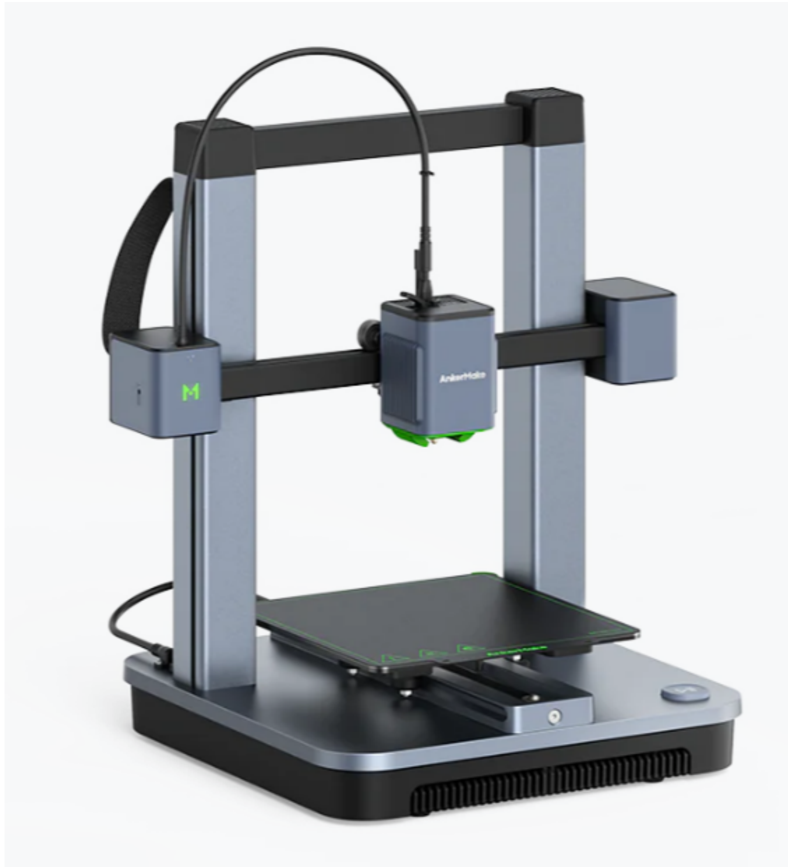
Anker’s budget 3d printing machine, the AnkerMake M5C, is an absolute bargain for the features and speed it brings. This printer has speed printing capabilities and an extremely friendly interface with almost no setup. This is one of the cheapest printers that can reach 0.5m/s speeds while printing, which alone makes it stand out.

M5C Features

- Cartesian Motion System optimized for speed
- Auto Bed Leveling
- Removable Bed
- All-Metal Direct Drive Extruder
- WiFi Printing capabilities
- 220mm x 220mm x 250mm Print Volume
- Extremely friendly software and setup for new users

M5C Drawbacks

- Replacement parts are proprietary from AnkerMake (albeit well-priced)
- Reliant on Anker for future firmware/software updates
- Limited on speed compared to CoreXY, but still has unrivaled speed at this price and availability
- No screen on the printer, all monitoring must be done digitally
- V wheel motion system can wear after long extended use



Prusa Mini ($429)

If you’re ok with paying a premium and getting a smaller build volume in exchange for a printer that just works every time, the Prusa Mini is a great option, as Prusa has had millions of hours running these machines. Just about every issue with this printer has been found, patched, and pushed to the consumer.

Prusa Mini Features

- Auto Bed Leveling
- Removable Spring Steel Sheets
- Prusa’s consistency guarantee
- Open Source Hardware
- Easily Transportable
- 180mm x 180mm x 180mm Print Volume

Prusa Mini Drawbacks

- Premium price
- Cantilever/unsupported X axis
- Lead times due to printer desirability
- No stock network capability

Note: You can now order Prusa printers from either their headquarters in Czechia or their subsidiary and sole authorized reseller **PrintedSolid**, based in Delaware. If you live in the USA and plan to order a Prusa printer, save yourself the headache of customs and long shipping times and order from their USA subsidiary.

BambuLab P1P ($599)

Built for speed by BambuLab, the P1P is a CoreXY machine with well engineered proprietary hardware and software. A machine that is proving very reliable for many despite BambuLab's short time in the 3d printer marketplace so far, the P1P is an amazing mid range option with a lot of manufacturing capability for it's cost. This printer is also compatible with BambuLab's multimaterial system, and can be upgraded to their new offering, the P1S, for just 150$ if your needs eventually outgrow the P1P.

Note: This printer is a PLA workhorse, being able to print it about as fast and well as the BambuLab X1C at half the price. Even if you have the budget for an X1C, it may be worth considering buying 2 P1Ps instead if you don't need all the bells and whistles the X1C has and plan to only print PLA/PETG.

P1P Features

- Extremely fast CoreXY motion system
- Auto Bed Leveling
- Removable Beds of All Surfaces
- All-Metal Direct Drive Extruder
- WiFi Printing capabilities and remote print monitoring
- 256mm x 256mm x 256mm Print Volume
- Automatic print failure detection

P1P Drawbacks

- Replacement parts are proprietary from Bambu Lab (albeit well-priced)
- Carbon Fiber rods can wear out over time
- Reliant on Bambu Lab for future firmware/software updates

Creality Ender-3 S1/Pro/Plus ($379-$549)

If you want a printer that can do most things well at a reasonable price, the Ender-3 S1 is a good fit for you. It has a standard build volume but is packed with pretty much every modern and quality of life upgrade installed out of the box, although you are paying for this premium. Additionally, it has a similar community backing to that of the original Ender-3 series due to similarities between the S1s and the originals.

S1 Features

All Ender 3 S1s have at least:

- Auto Bed Leveling
- Removable Spring Steel Sheets
- Direct Drive Extruder
- Built in Belt Tensioners
- Dual Z-Axis
- 220mm x 220mm x 270mm Print Volume

The S1 Pro also has:

- All-Metal Direct Drive Extruder

The S1 Plus has:

- 300mm x 300mm x 300mm Print Volume

S1 Drawbacks

- PTFE Lined Hotend on Normal and Plus Versions
- No stock network capability
- Speed limited compared to other printers at this price range

High-End Printers ($600+ USD)

Note: Tip for both of the Prusa printers listed on this page. You can now order Prusa printers from either their headquarters in Czechia or their subsidiary and sole authorized reseller **PrintedSolid**, based in Delaware. If you live in the USA and plan to order a Prusa printer, save yourself the headache of customs and long shipping times and order from their USA subsidiary.

Fig. 21: This is the Ender 3 S1 base model.

Prusa MK3S+ ($649-$899)

If you're looking to buy a printer that just works every time, the Prusa MK3S+ is amazing. Prusa has had millions of hours running these machines, and just about every issue with this printer has been found, patched, and pushed to the consumer. If it's any testament to their consistency, the 3d printed parts used on the Mk3s+ are printed mostly on Mk3s+ printers. This is Prusa's previous flagship printer and is more tested and cheaper than the Mk4, but if you are looking at Prusa make sure to explore the Mk4 as well due to it's more updated features.

Mk3s+ Features

- Easy to Repair
- Auto Bed Leveling
- Removable Spring Steel Sheets
- All-Metal Direct Drive Extruder
- 250mm x 210mm x 210mm Print Volume
- Unrivaled consistency as a workhorse

Mk3s+ Drawbacks

- Slow printing speed
- Dated technology (such as lack of WiFi)
- Last generation 3d printer, Mk4 improves on issues

Prusa MK4 ($799-$1099)

This printer is for anyone interested in consistency with an improved featureset. The Prusa MK4 is the successor to the MK3S+ with the same reliability and consistency hallmarks but adds newer features such as WiFi as well as being much faster and more user friendly. This machine is still in it's early stages of development and improvement as of 2023, and will surely make leaps and bounds in consistency, features, and software optimization as Prusa finds more issues and patches them.

Mk4 Features

- High Speed (Nearing comparison to BambuLab printers)
- Auto Bed Leveling
- Removable Spring Steel Sheets
- All-Metal Direct Drive Extruder with a planetary gearbox
- WiFi Printing capabilities and remote print monitoring
- 250mm x 210mm x 210mm Print Volume

Mk4 Drawbacks

- Cartesian kinematics make matching the speed of CoreXY printers difficult

BambuLab X1C ($1199-$1449)

This printer is for you want a no-compromises 3d printer that can handle pretty much anything you throw at it with incredible speed and reliability while using engineering-grade filaments. This printer comes at $1199 for just the printer and $1449 for the combo that includes BambuLab's multimaterial system which can handle 4 filament rolls at once.

Note: This printer is **expensive** and is targeted towards advanced filaments. The P1P can print basic filaments such as PLA/PETG about as fast and well as the BambuLab X1C at half the price. If you have the budget for an X1C, it may be worth considering buying 2 P1Ps instead if you don't need all the bells and whistles the X1C has and plan to only print PLA/PETG.

X1C Features

- Extremely fast CoreXY motion system
- Auto Bed Leveling
- Full Color Touchscreen
- Removable Beds of All Surfaces
- All-Metal Direct Drive Extruder
- WiFi Printing capabilities and remote print monitoring
- 256mm x 256mm x 256mm Print Volume
- Heated chamber allows for more advanced engineering materials
- LIDAR sensor for flow calibration and first layer quality checking
- Stock hardened hotend capable of most filled and abrasive filaments
- Automatic print failure detection

X1C Drawbacks

- Replacement parts are proprietary from Bambu Lab (albeit well-priced)
- Carbon Fiber rods can wear out over time
- Reliant on Bambu Lab for future firmware/software updates
- This printer doesn't shine if you don't use it for advanced filaments, and may not be worth the cost if you don't plan to

BambuLab P1S ($699-$949)

If the X1C's frills such as LIDAR, touchscreen, and a hardened extruder didn't sound all that useful to you, the P1S could be a good option. You still get an enclosure and auxiliary cooling, while the P1P doesn't, which allows you to print filaments like ABS/ASA without difficulty, but this printer needs a fair few upgrades to print filled filaments and more advanced engineering filaments safely.

P1S Features

- Extremely fast CoreXY motion system
- Auto Bed Leveling
- Removable Beds of All Surfaces
- All-Metal Direct Drive Extruder
- WiFi Printing capabilities and remote print monitoring
- 256mm x 256mm x 256mm Print Volume
- Heated chamber allows for more advanced engineering materials
- Automatic print failure detection

P1S Drawbacks

- Replacement parts are proprietary from Bambu Lab (albeit well-priced)
- Carbon Fiber rods can wear out over time
- Reliant on Bambu Lab for future firmware/software updates

AnkerMake M5 ($699)

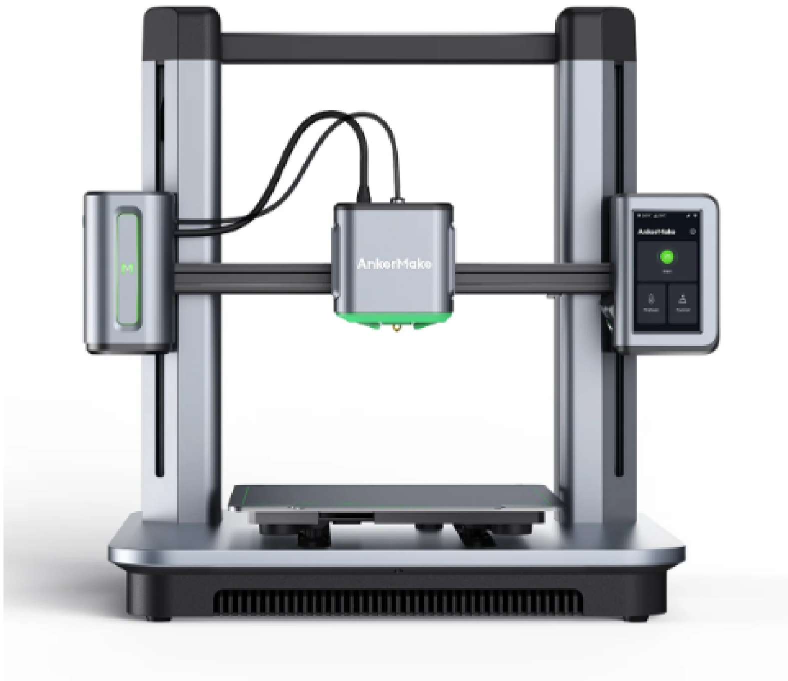
Anker's new entry into the 3d printer market, the AnkerMake M5, is a very reasonably priced printer for the features it brings. With print failure detection, speed printing capabilities, and an extremely friendly interface with almost no setup, the M5 is a great option.

M5 Features

- Cartesian Motion System optimized for speed
- Auto Bed Leveling
- Removable Bed
- Direct Drive Extruder
- WiFi Printing capabilities and remote print monitoring
- 235mm x 235mm x 250mm Print Volume
- Automatic print failure detection
- Extremely friendly software and setup for new users

M5 Drawbacks

- Replacement parts are proprietary from AnkerMake (albeit well-priced)
- Reliant on Anker for future firmware/software updates
- Cartesian motion system limits speed, still doesn't match CoreXY machines at a similar price range
- PTFE Lined Hotend
- V wheel motion system can wear after long extended use



DIY Printers (Voron, HevOrt, VZBot)

> **Warning:** For teams just getting into 3D printing or teams that haven't had experience with at least 2-3 other printers we would highly advise against a DIY printer. These printers are **projects** and take significant effort and upkeep, which make them a poor choice for a first or second printer.

If you want to take a deeper dive into 3D Printing and achieve both extreme speeds and quality, a DIY printer may be a good choice. DIY printers can be tailored to your specific needs and perform extremely well but are typically a huge time and financial commitment.

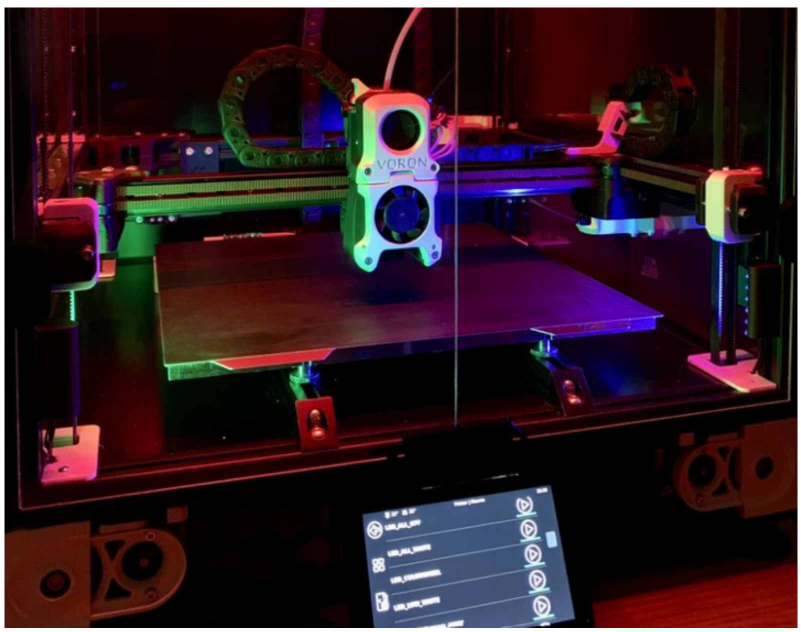
Features

Most well-documented DIY Printers feature…

- Extremely fast CoreXY motion systems
- Auto Bed Leveling
- Removable Spring Steel Sheets
- Direct Drive Extruders with All Metal Hotends
- Large Build Volumes (250mm^3 or more)
- Klipper Firmware and live dashboards for print monitoring
- High customizability and a strong community that creates modifications

Drawbacks

- Extremely complex to assemble, wire, and configure
- Have to source your own parts, no “official” place to buy from
- Huge time commitment
- Huge financial commitment (typically $1000+)

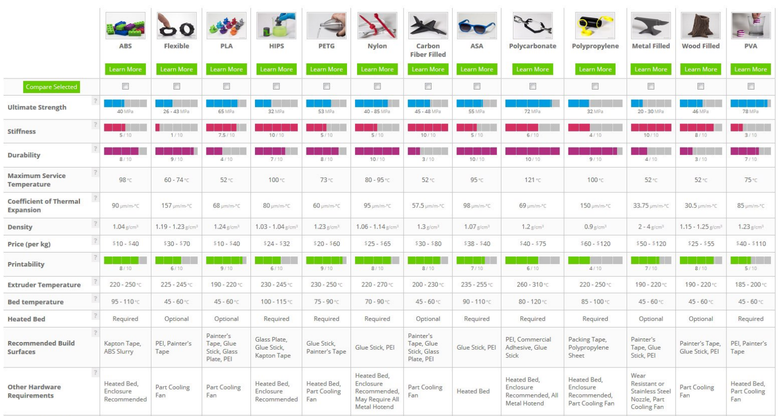


24.1.5 Filament Choice

There are many different materials and colors of plastics called “filaments” that can be used for 3D printing, but for FTC there are only a few that are of practical use for most teams. Since this guide is currently only addressing FDM (Fused Deposition Modeling) printing, materials like resin (used in SLA printing) will not be discussed. |

> **Warning:** Currently, the majority of commercial 3D printers use a 1.75mm diameter filament, so when shopping for filaments, teams should take care to avoid other diameters, such as 2.85mm, as these will not work with most printers.

This section is all about FDM filament choices for 3D printing. There are a variety of plastics that are commonly used, with some being more common than others, and some fulfilling specialty use cases. There are specialized to FTC descriptions in this section, but if you want a simple comparison chart like shown below, check out this Filament Properties Table.



Compare Selected	ABS	Flexible	PLA	HIPS	PETG	Nylon	Carbon Fiber Filled	ASA	Polycarbonate	Polypropylene	Metal Filled	Wood Filled	PVA
	Learn More	Learn More	Learn More	Learn More	Learn More	Learn More	Learn More	Learn More	Learn More	Learn More	Learn More	Learn More	Learn More
Ultimate Strength	40 MPa	26 - 43 MPa	65 MPa	32 MPa	53 MPa	40 - 85 MPa	45 - 48 MPa	55 MPa	72 MPa	32 MPa	20 - 30 MPa	46 MPa	78 MPa
Stiffness	5 / 10	1 / 10	7.5 / 10	10 / 10	5 / 10	5 / 10	10 / 10	5 / 10	6 / 10	4 / 10	10 / 10	8 / 10	3 / 10
Durability	8 / 10	9 / 10	4 / 10	7 / 10	8 / 10	10 / 10	3 / 10	10 / 10	10 / 10	9 / 10	4 / 10	3 / 10	7 / 10
Maximum Service Temperature	98 °C	60 - 74 °C	52 °C	100 °C	73 °C	80 - 95 °C	52 °C	95 °C	121 °C	100 °C	52 °C	52 °C	75 °C
Coefficient of Thermal Expansion	90 µm/m-°C	157 µm/m-°C	68 µm/m-°C	80 µm/m-°C	60 µm/m-°C	95 µm/m-°C	57.5 µm/m-°C	98 µm/m-°C	69 µm/m-°C	150 µm/m-°C	33.75 µm/m-°C	30.5 µm/m-°C	85 µm/m-°C
Density	1.04 g/cm³	1.19 - 1.23 g/cm³	1.24 g/cm³	1.03 - 1.04 g/cm³	1.23 g/cm³	1.06 - 1.14 g/cm³	1.3 g/cm³	1.07 g/cm³	1.2 g/cm³	0.9 g/cm³	2 - 4 g/cm³	1.15 - 1.25 g/cm³	1.23 g/cm³
Price (per kg)	$10 - $40	$30 - $70	$10 - $40	$24 - $32	$20 - $60	$25 - $65	$30 - $80	$38 - $40	$40 - $75	$60 - $120	$50 - $120	$25 - $55	$40 - $110
Printability	8 / 10	6 / 10	9 / 10	6 / 10	9 / 10	8 / 10	8 / 10	7 / 10	6 / 10	4 / 10	7 / 10	8 / 10	5 / 10
Extruder Temperature	220 - 250 °C	225 - 245 °C	190 - 220 °C	230 - 245 °C	230 - 250 °C	220 - 270 °C	200 - 230 °C	235 - 255 °C	260 - 310 °C	220 - 250 °C	190 - 220 °C	190 - 220 °C	185 - 200 °C
Bed temperature	95 - 110 °C	45 - 60 °C	45 - 60 °C	100 - 115 °C	75 - 90 °C	70 - 90 °C	45 - 60 °C	90 - 110 °C	80 - 120 °C	85 - 100 °C	45 - 60 °C	45 - 60 °C	45 - 60 °C
Heated Bed	Required	Optional	Optional	Required	Required	Required	Optional	Required	Required	Required	Optional	Optional	Required
Recommended Build Surfaces	Kapton Tape, ABS Slurry	PEI, Painter's Tape	Painter's Tape, Glue Stick, Glass Plate, PEI	Glass Plate, Glue Stick, Kapton Tape	Glue Stick, Painter's Tape	Glue Stick, PEI	Painter's Tape, Glue Stick, Glass Plate, PEI	Glue Stick, PEI	PEI, Commercial Adhesive, Glue Stick	Packing Tape, Polypropylene Sheet	Painter's Tape, Glue Stick, PEI	Painter's Tape, Glue Stick, PEI	PEI, Painter's Tape
Other Hardware Requirements	Heated Bed, Enclosure Recommended	Part Cooling Fan	Part Cooling Fan	Heated Bed, Enclosure Recommended	Heated Bed, Part Cooling Fan	Heated Bed, Enclosure Recommended, May Require All Metal Hotend	Part Cooling Fan	Heated Bed	Heated Bed, Enclosure Recommended, All Metal Hotend	Heated Bed, Enclosure Recommended, Part Cooling Fan	Wear Resistant or Stainless Steel Nozzle, Part Cooling Fan	Part Cooling Fan	Heated Bed, Part Cooling Fan

This section also talks about proper ways to store filament and the ever present complaint of “moisture in filament” in the filament storage section.

Common Filaments

> **Tip:** These common filaments come in the most variety of color. Printing parts in colors that match a team’s brand is an easy way to customize a robot!

This page will go through the most common filaments and their properties. While each filament has advantages and disadvantages, most teams will find that the filaments listed on this page, PLA (Polylactic Acid) and PETG (Polyethylene Terephthalate Glycol) will be their best choice for strength, durability, cost, and aesthetics. These filaments are the easiest to print with, and are available from many different manufacturers at a low cost. Many other filaments that will be discussed add some specific property (ex: TPU/TPE for flexibility), but are more difficult to print with, and are often more expensive.

The temperatures listed here are simply a range of the common temperatures for that filament, and may vary depending on the specific filament.

PLA (Polylactic Acid)

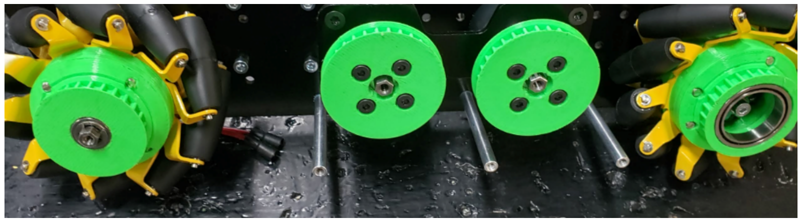


Fig. 22: A custom drivetrain with PLA parts used across it.

Polylactic Acid, or PLA, is the most common 3D printing filament used today. It is made from biological sources such as corn starch or sugar cane. PLA is easy to print with, and is usually the best choice for most robot parts. It prints at a low temperature, and tends to warp very little. PLA is very stiff, but can be brittle, especially under shock loads (impacts), and parts should be designed with this in mind.

PLA is also sold in a massive number of variations from different manufacturers, like PLA+ or PLA Pro. These filaments contain different additives to improve the properties of the filament, such as increased strength or better printability. While more expensive, these filaments can be a great option that make PLA much more capable.

- PLA hotend temperatures: 190-230° C
- PLA bed temperatures: 20-70° C; PLA does not require a heated bed, but it is recommended.

Warning: Warning #1 about PLA: Due to the relatively low melting point of PLA, it is not advisable to leave PLA parts in locations such as a hot car, as this can produce severe warping in those parts.

Warning: Warning #2 about PLA: PLA is extremely sensitive to UV light and sunlight. After multiple days outside during something such as an outreach event, if your robot is not under shade you may experience part failures. Make sure to account for this and either avoid long outdoor outreach events with your robot exposed or be prepared to replace parts after such events.

PETG (Polyethylene Terephthalate Glycol)

PETG is another very common filament that some consider an upgrade to PLA. Being only slightly more difficult to print with than PLA, PETG often has more stringing and other artifacts on parts. PETG's tensile strength is a technically lower than that of PLA, however it is much more flexible and less brittle. Because of this, PETG is more resistant to shock loads than PLA, and is a good choice for parts that may be impacted. PETG is also more resistant to heat than PLA, and is unlikely to warp when left in a hot location.

- PETG hotend temperatures: 230-250° C
- PETG bed temperatures: 60-80° C

Fig. 23: A linear slide insert printed in PET-G filament.

Warning: PETG is well known for bonding extremely well to print beds, especially those made out of glass and PEI, to the point of tearing chunks out of the print surface. If you are printing with PETG, it may be a good idea to apply some glue stick or hairspray to the surface to prevent this.

Advanced Filaments

Warning: If your printer's hotend (the part that melts the filament) has a PTFE (Teflon) lining where the PTFE tube goes all the way down to the heat block (common in lower price printers like the base Ender 3), then you should not be printing at or above 250° C. Doing so will cause the PTFE tube to degrade and melt, potentially releasing toxic fumes. If you need to print at these temperatures (used for some of the advanced filaments listed here), and you have a PTFE lined hotend, you can look at upgrading to an all-metal hotend.

ABS (Acrylonitrile Butadiene Styrene) / ASA (Acrylonitrile Styrene Acrylate)

Fig. 24: ABS can best be recognized as the plastic used for Lego Bricks.

Before PLA became readily available, ABS was the most common filament used for 3D printing. Nowadays, it's regarded as a more advanced filament with a specilized setup needed. ABS is very strong, having a high ductility and able to withstand shock loads well. These strengths come with major difficulties, however, as an enclosure is often needed to increase the ambient temperature in order to prevent severe part warping. This enclosure is also a good idea due to ABS's production of styrene when it melts, a carcinogenic gas that can cause headaches. ABS should only be printed within filtered enclosures or in extremely well ventilated areas. If you have a proper setup with a heated enclosure, ABS can be very worth your time, but if you don't have a setup for it, it's not worth the pain to try and print it. Notably, ABS can be very inexpensive and often found near the same price as PLA or even cheaper.

Note: ASA, a slightly more expensive but similar material, prints at approximately the same temperatures but prints slightly easier and produces considerably less styrene. It also may not require an enclosure for smaller prints. Alongside that, it's UV Resistant and typically offers slightly better mechanical properties compared to most ABS plastics.

- ABS hotend temperatures: 230-250° C
- ABS bed temperatures: 100-120° C

- ABS enclosure temperature: 30-40° C

Note: It is important to properly vent/filter fumes emitting from ABS/ASA, as prolonged exposure to either can lead to poisoning. Typically it's recommended to use an activated carbon air filter, however most enclosures aren't airtight, and therefore it's also important to leave your printer somewhere safe if you plan on printing large or numerous parts.

Polyamide Filaments

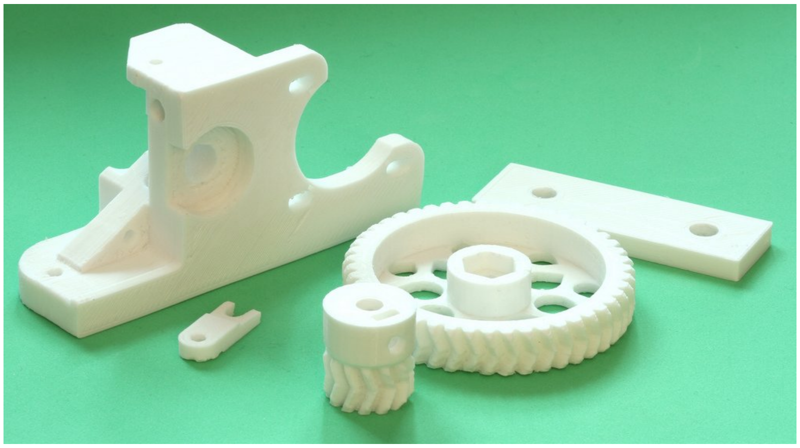


Fig. 25: A collection of Nylon parts, including gears, a great usecase for Nylon

Note: Nylon is a category of Polyamide, and these generalizations should apply to all filaments based on Nylon or a general Polyamide.

Being more of a category of filaments, polyamide (PA) based filaments can boast impressive capabilities while being relatively easy to print. These show their strength in extreme durability and resistance to wear, making them ideal for gears and pulleys if you can safely print them. Common filaments in this category include Pure Nylon, Polyamide, PA-KV (kevlar filled nylon), and PA-CF (carbon fiber filled polyamide). Polyamide filaments will commonly require printing temperatures in excess of 250° C, making an all-metal hotend necessary, and require heated bed temperatures in the range of 80-100° C. Some of these filaments are abrasive, requiring a hardened steel nozzle on your hotend to avoid damage. These filaments are also **extremely** hygroscopic, making proper storage a necessity even while actively printing. Nylon prints also tend to shrink when cooling.

Carbon Fiber Filaments

Fig. 26: The abrasiveness of Carbon Fiber filament is visible.

Carbon fiber (CF) filled filaments are everywhere, bringing increased stiffness and strength to many parts while keeping them light. You can find PLA-CF, PETG-CF, PA-CF, CF-ASA, PC-CF, and many others. CF filled filaments typically keep many properties from the filament they are based on, including how hygroscopic the filament is, the printing temperatures, and the ease of printing. The one large difference is the abrasion of the filament, meaning that you need a hardened steel nozzle or other abrasion resistant nozzle for your hotend.

Note: Carbon Fiber reinforced PLA is one of the most commonly used CF Filaments, but comes with a notable drawback that it becomes incredibly brittle. It's important to to keep these kinds of trade-offs in mind, as while it's seen as "stronger", it doesn't hold up as well as most regular PLAs when it comes to dealing with impacts and may suffer from poor layer adhesion.

TPU/TPE (Thermoplastic Polyurethane/Elastomer)

TPU and TPE are flexible filaments that can be used to create parts that are flexible and can bend. These filaments are sold under a variety of different durometers (a measure of a material's hardness). You will find 95A durometer the most common durometer due to it's printability. TPU/TPE's flexibility grants it an extremely high impact resistance, making it very durable as well. In FTC, TPU/TPE is often used to make flexible components such as intake rollers, wheel bumpers, and occasionally low-load toothed belts. Finally, TPU is extremely hygroscopic, and proper filament storage practices should be used.

Note: Since TPU/TPE is very flexible, printers with a Bowden extrusion system, where the extruder motor and gear is not

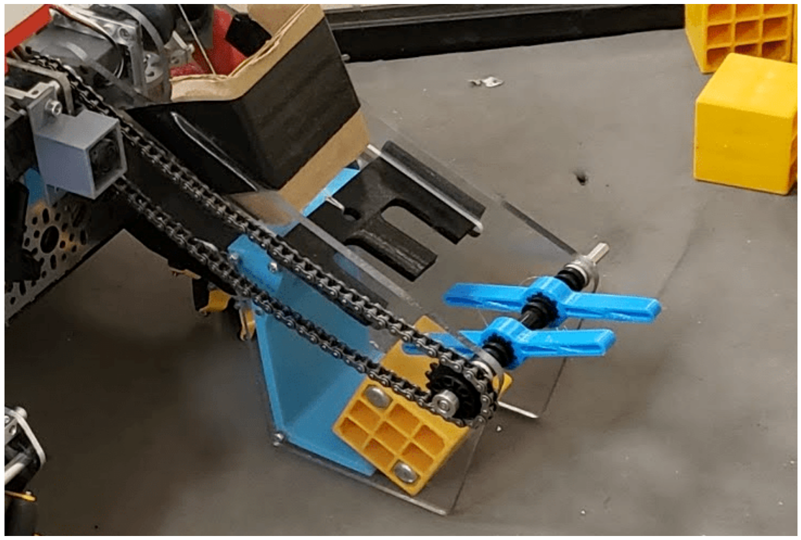


Fig. 27: An intake using custom TPU parts to grab game elements.

located near the hotend, will have a very difficult time printing with it.

- TPU hotend temperatures: 210-250° C
- TPU bed temperatures: Heated bed not required, but do not exceed 60° C
- TPU printing speeds should never exceed 50mm/s on a direct drive printer and 20mm/s on a bowden printer
- Direct drive extrusion system highly recommended

Filament Storage Details

General Guidelines

Here are some general guidelines for how careful you should be to store filaments.

- Can be left in open air, sealed storage preferred: ABS, ASA
- Sealed storage highly recommended: PLA, PET-G
- Sealed storage highly recommended while printing and storing: TPU/TPE, Polyamides

Why We Need Filament Storage

It's important to keep in mind that filaments need to be stored in a specific way- and even the basic filaments PLA and PET-G are no exception. Filaments are **hygroscopic** (absorb moisture from the air), and that affects print quality majorly if not taken care of. Store filaments in a closed and sealed container, preferably kept dry with spare silica packets, and don't leave them out whenever possible.

If you suspect a filament has taken on water, look up a drying temperature for it, and put it in the oven or a specialized filament dryer at that temperature for a few hours to get rid of the water.

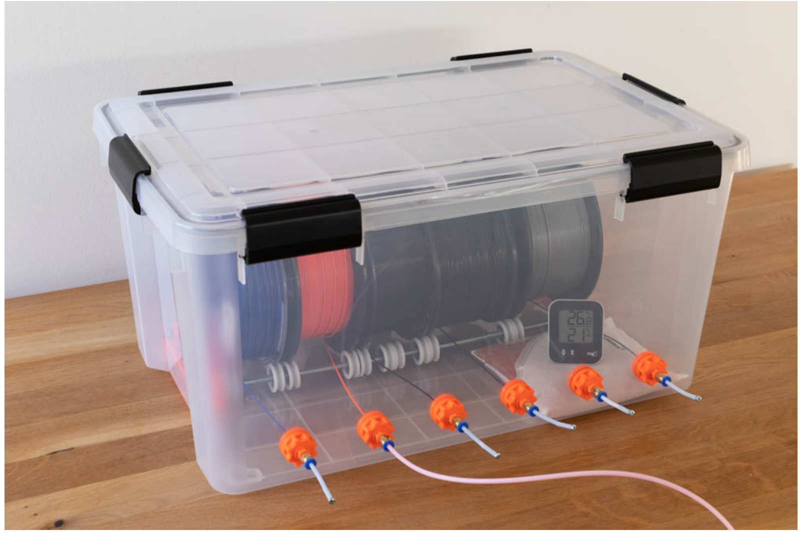


Fig. 28: A large sealed bin can be customized to fit your use with silica packets thrown in.

Tip: An easy way to tell if a filament has taken on moisture is if you hear a slight popping noise when printing- this is the moisture forming bubbles in the filament as it's heated.

Storing While Printing

Some more sensitive hygroscopic filaments don't just need to be stored in sealed containers, but also need to be printed while in a sealed container. Lots of companies sell or design "dryboxes" for this purpose, such as SUNLU's filament dryer box, BambuLab's AMS setup, or custom designs such as one by Prusa Research. Do your own research and find out which of these is best for you if you deal with extremely sensitive filaments.

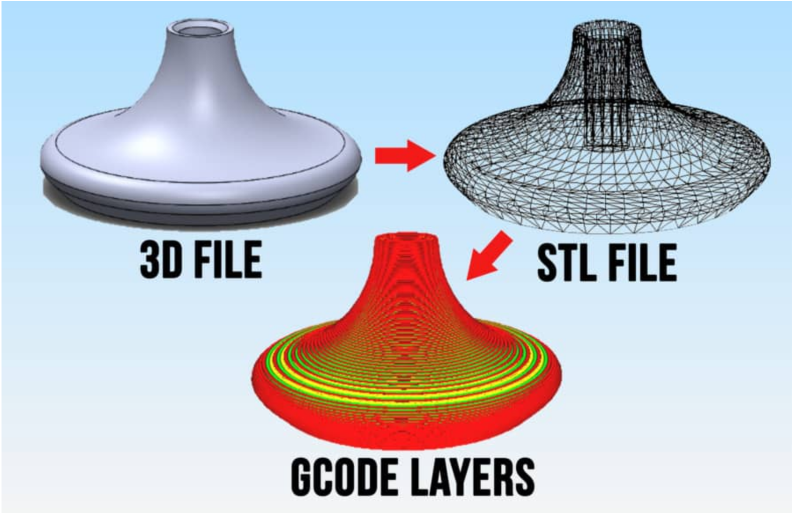
Fig. 29: Prusa's filament drybox solution, 3D printed and DIY.

24.1.6 Designing and Slicing

This section goes over the essentials of how to design CAD files that are easily 3d printable, and slicing softwares that can take cad files into instructions for a 3d printer.

What is Slicing?

One of the most important steps of 3D printing is turning your CAD files into instructions for the printer can carry out and read. This is done in a software called a slicer, which turns CAD files, into machine-readable, G-Code Files.



The basis of this software is that it takes a mesh file, normally an STL file, and splits it into layers and lines that the 3d printer can digest. You get to select various parameters about how it will instruct the printer to follow this data, such as the speed, thickness of lines, temperatures, thickness of the slices, amount of walls, the patterns on the inside of the part, and even more. Most slicers will have over 200 options available to you if fine-tuning is something you're interested in, but there's only about 10-20 settings that you absolutely have to be concerned with.

G-Code files are also notably just compilations of G-Code commands. G-Code commands tell the printer how to move, where to move, and what to turn on and off. These G-Code commands can be useful to learn if your printer has a command line or you want to eventually write start-up or shut-down scripts for your prints.

Slicer Software Choices

There's a variety of slicing software available for use, either community or company maintained. Preference plays a large factor in choosing software, so it's recommended to see what software you like best.

If you are using a Prusa, BambuLab, or AnkerMake printer, your first consideration should be these companies respective softwares. PrusaSlicer is built to work well with Prusa machines, BambuStudio with BambuLab machines, and AnkerMake's slicer for Anker machines.

If you are using any other printer or the proprietary software for your printer doesn't fit well with you, Ultimaker Cura and PrusaSlicer are both amazing options. Both with almost every modern slicer feature, these are great options. Cura is great if you want a clean UI with minimal settings, while PrusaSlicer may enable more powerful control of your printer with more customizable settings and a larger open source community.

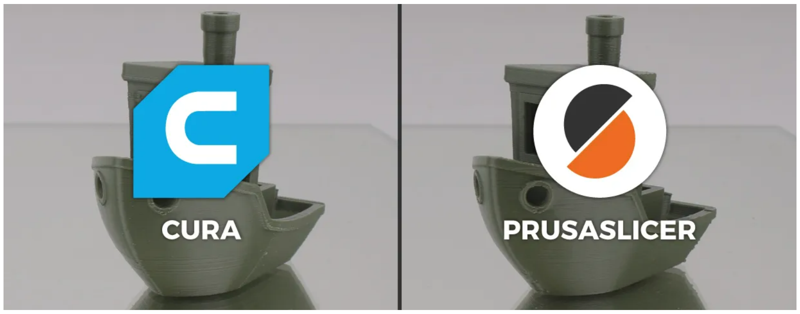


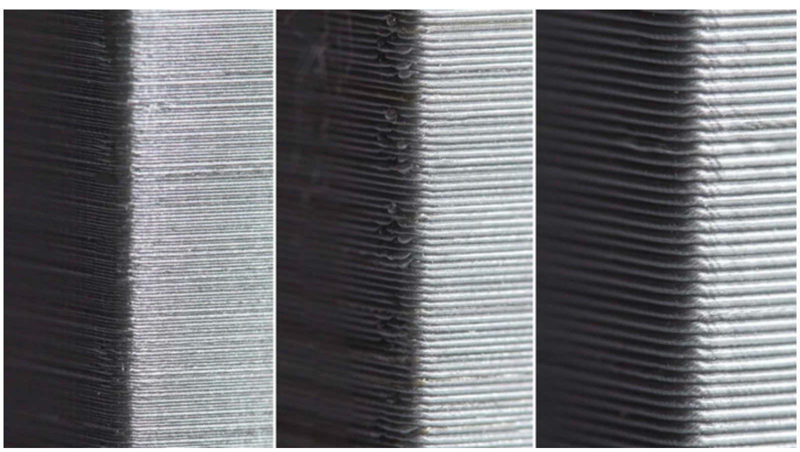
Fig. 30: The two most popular slicer softwares with the most community support, Ultimaker Cura and PrusaSlicer.

Both PrusaSlicer and Cura have lists and lists of printers they have prebuilt slicer profiles for, but in case your printer isn't on those lists, you should search up the slicer you are using and your printers name. If the printer is relatively popular, there will be slicer profiles posted online that you can import into these slicer softwares.

Explanation of Slicer Settings

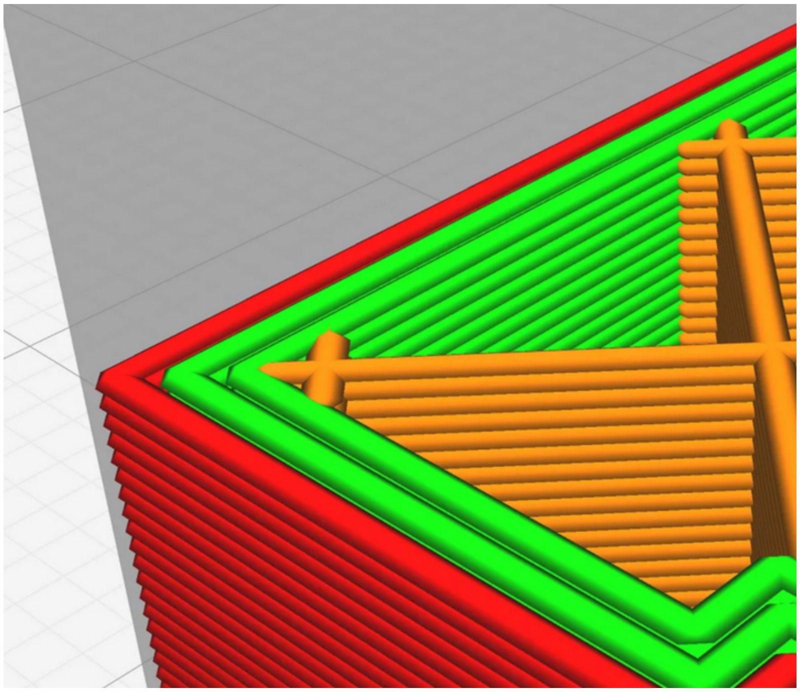
Layer Height

Layer height is the biggest factor affecting quality, speed, and (somewhat) strength, if you are concerned about tolerances and quality, a good starting point is 50% of your nozzle width (typically 0.4mm, so a 0.2mm width at 50%). On the other hand if you want more speed, at the cost of some strength, opt for up to 75% of your nozzle width instead (up to 0.32mm for a 0.4mm nozzle).



Walls

Walls refer to how many solid "outer" layers a print has a we'd recommend anywhere from 3-5 walls for strong prints. Additionally, its a good idea for all prints is to turn on an alternating extra shell. This interlocks the walls with the infill more, making the infill stronger.



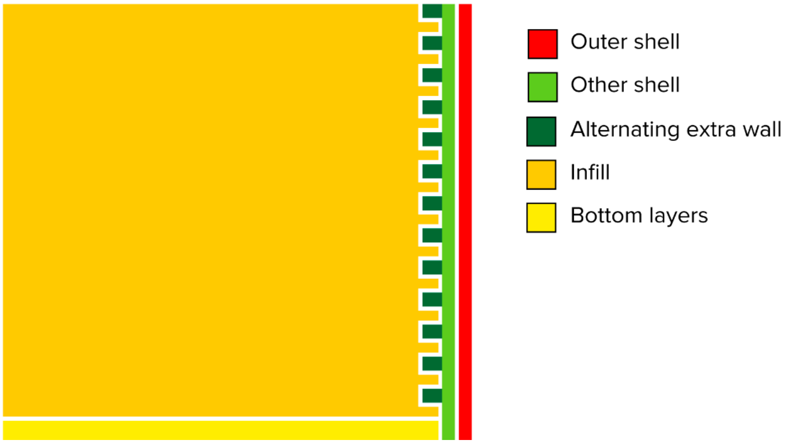
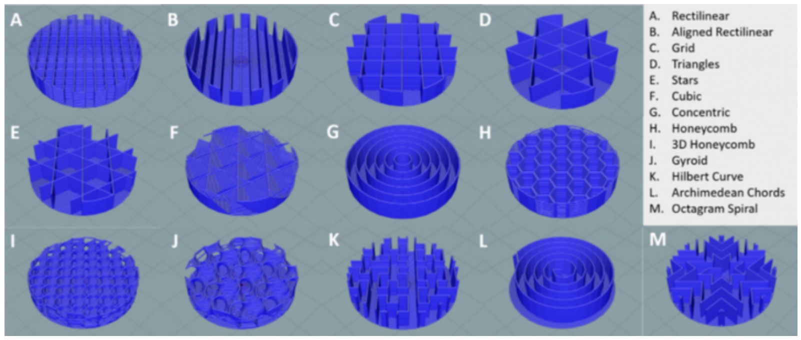


Fig. 31: While sometimes a tough setting to find, this alternating extra shell interlocks with infill and considerably impacts strength.

Infill

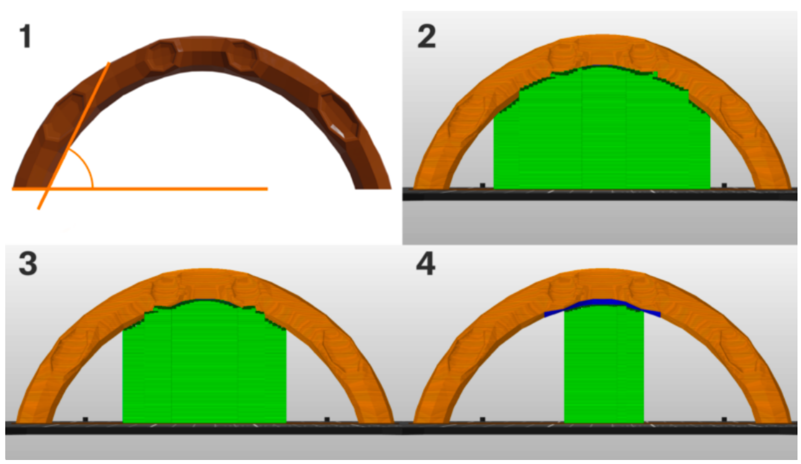
Infill is a measure of how solid your part is. This can vary based on desired strength. Infill patterns are the pattern taken by the percentage of infill that you specify. For Infill patterns it is recommended to use **3D infill patterns** such as "Cubic" and "Gyroid". 2D infill patterns such as "Grid" don't have uniform stability in all directions.



Supports

3D Printers are amazing, but all machines unfortunately, have limitations, and with 3D printers that means, you can't print melted plastic in midair and expect it to stay there. This is where supports come in. Supports are excess printed material designed to be broken away after the print. Generally, we want to avoid support material by designing around it. But if not possible, it's a good idea to use "Support Enforcers"/"Paint On Supports" to closely control what areas you know need support. This makes cleanup after a print a lot easier.

If you want more info on supports, check out Prusa's guide here!



Recommended Wall and Infill Settings

While not applicable for all situations, this is some general guidance on wall and infill strength for PLA parts.

- **Purely Aesthetic Parts**: 5-10% Infill, 2 alternating 3 Walls
- **Typical Use Parts**: 10-20% Infill, 3 alternating 4 Walls
- **Load Bearing and Structural Parts**: 40-60% Infill, 4 alternating 5 Walls
- **Fully Structural and Integral Parts (Take Shocks when Robot is dropped)**: 80-100% Infill, 4 alternating 5 Walls

Note: Depending on how well your printer is tuned, 100% infill can be weaker than slightly lower infill. 90% infill is a recommended maximum to leave space for overextrusion.

General Design for 3d Printed Parts

Gradually as you 3d print more you will acquire a general design sense. You will have a good sense of what you should and shouldn't print, know what your printer is capable of, and know what won't break. A lot of this comes with experience, but there are some general tips that can be given.

Large Prints

If you're using your entire build plate, you generally should not be printing that part continuously. Design the part to be connected in various places with hardware, or even better, design it to have metal reinforcement and use 3d prints for only the unique/specific geometry.

The reason this is said is due to problems with unitization (a practice of combining or splitting multiple parts into smaller or larger objects). If your part takes a whole print plate, and breaks while on the robot, you have to painfully run a very long print again. Segmented parts can be more easily replaced if part of it breaks. Keep your parts small and easy to run replacement prints for.

Avoiding Fine Details

3D Printers can achieve a fairly high level of detail but functional parts of a part should be kept to be large geometry as smaller geometry often fails easier. For example, when printing something like gears, it's advised to do a large module such as 1-2mm, because standard 3D printers just can't print smaller teeth with accuracy and strength. You should make sure all parts of a print are at least 2-3x your nozzle width, to make sure that sufficient strength can be achieved. For a standard 0.4mm nozzle, 1.2mm is really the thinnest any part of your design should be.

Design to Make Easy-To-Print Parts

A major problem teams sometimes encounter is that their parts are difficult to print without extreme amounts of supports or precarious overhangs. This issue can and should be solved during the design process in a practice sometimes called DFAM (Designing for Additive Manufacturing). 3d Printers have limits, and these can be taken into account while initially creating a part, making those manufacturing limitations just as much of a consideration as what the function of the part you're creating is.

Printing Face

When you first conceptualize a part you plan to print, immediately consider what face will lie on the built plate for it to print safely on. In general the more area touching the plate, the better, as it provides more adhesion. Always design with creating large flat surfaces for printing in mind.

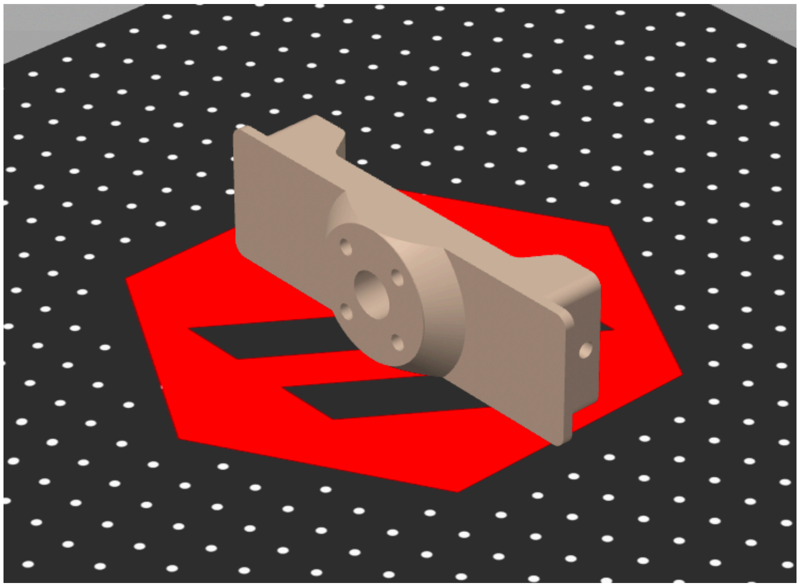


Fig. 32: A part with a slightly concerning contact with the bed. This part could have been designed to rest another, larger face on the bed.

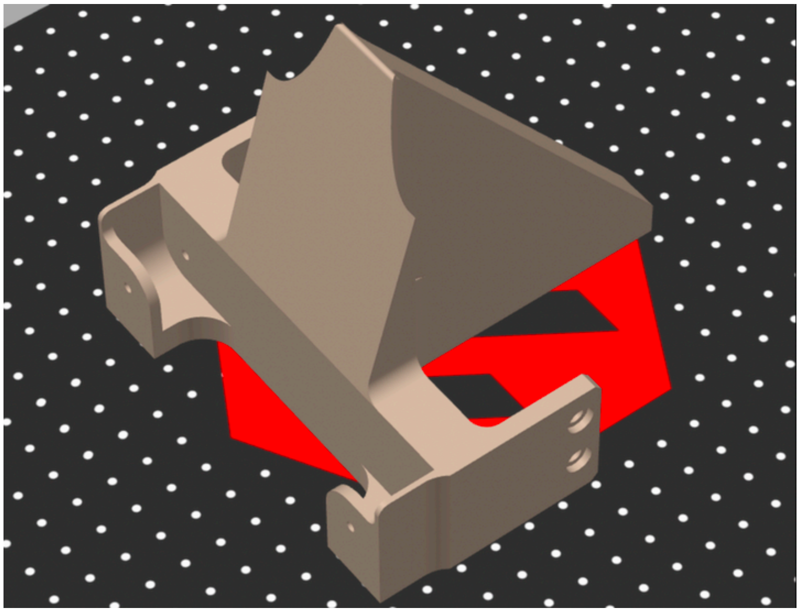
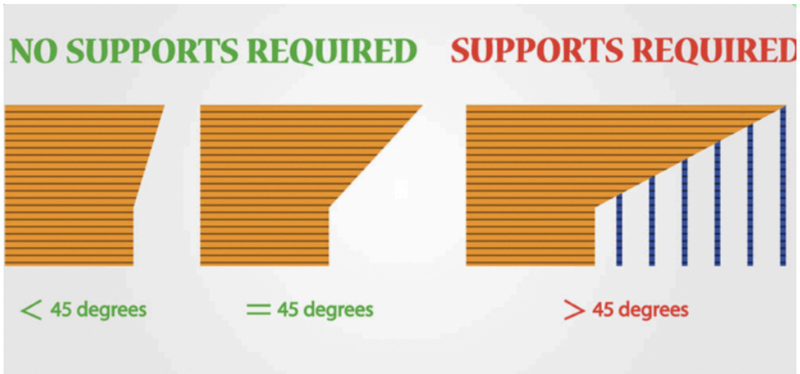


Fig. 33: A part with much better contact with the bed. This part looks great to print.

Overhang Angles

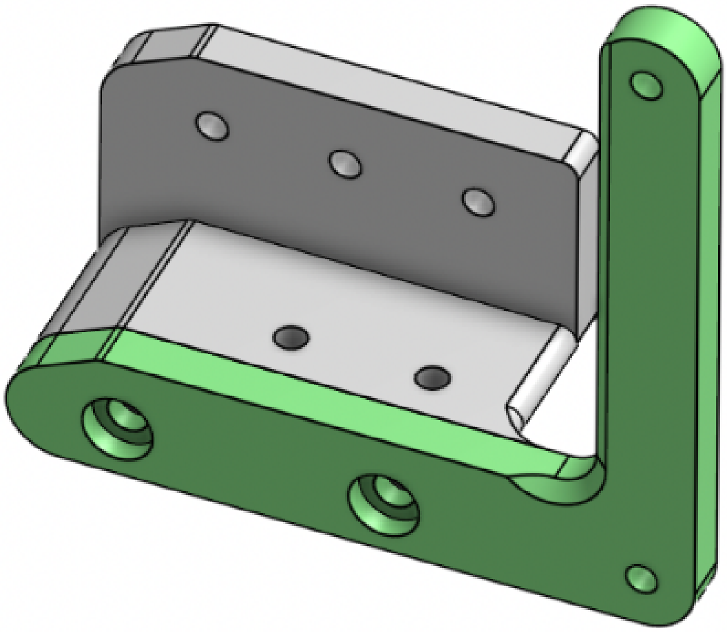
Supports should only be used if absolutely necessary. In almost all parts, you can find some way to create angles or adjust geometry to avoid support usage. Supports lengthen post-processing time, are generally difficult to tune, leave less clean surfaces, and are more prone to failure.



After you determine your printing face for a part, look at it closely and determine whether there's any areas with over a ~45 degree overhang. Then, try and mitigate those areas using chamfers, fillets, and adding material. If mitigation isn't possible, consider splitting the part into multiple pieces and securing them together with fasteners.

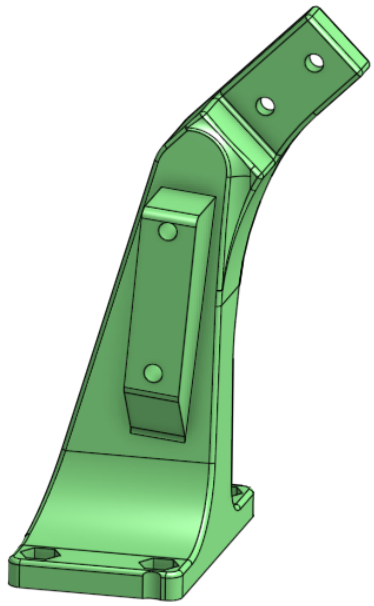
Overhang Angles Case Study #1

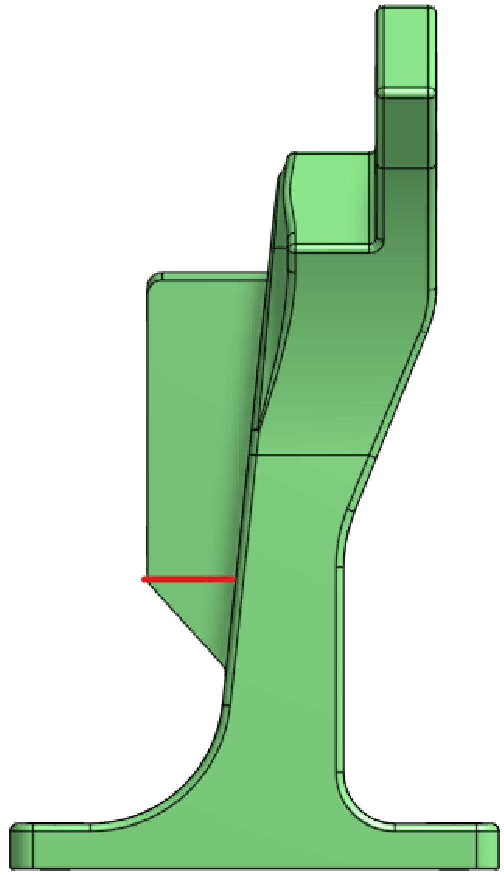
The part you see below would have been very frustrating to print in any orientation without the use of supports, so it was opted to split it in two and use M3 screws to attach the pieces instead. This provided two very easy to print parts.



Overhang Angles Case Study #2

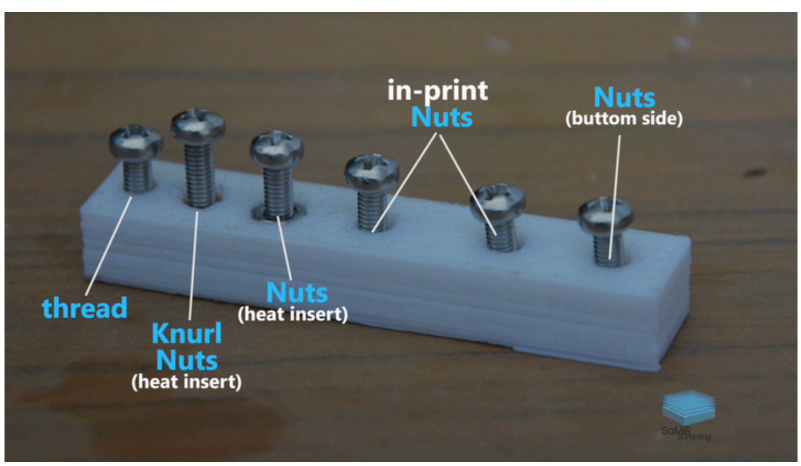
The part you see below is an arm hardstop for a robot, eventually printed with no supports despite its complex geometry. In particular, pay attention to the second picture. The part could have been designed to stop at the red line but was extended to a 45 degree overhang to ensure the printer wasn't printing in midair.





Working With Screws

3D printed parts typically interact quite often with your hardware and screws. A lot of teams simply use locknuts to secure 3D printed parts but while they are strong, they may not always fit or work for your application. Thankfully, there a quite a few other ways to secure 3D printed parts without much effort:



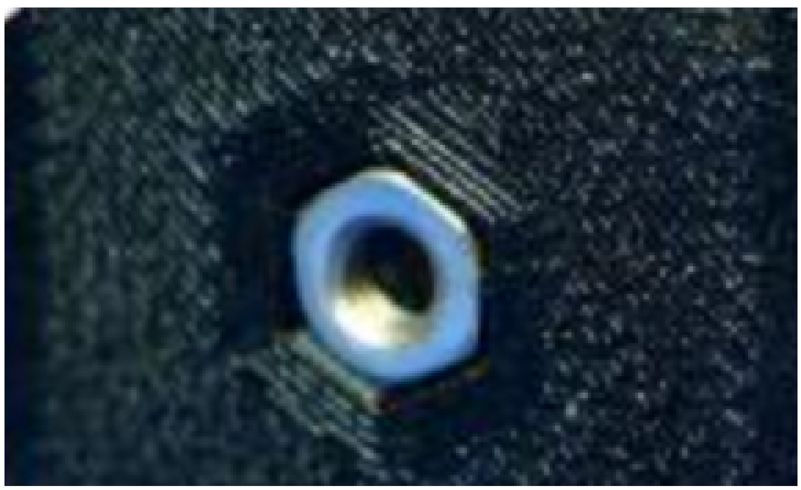
Plastic Threading

For low strength, simple, and aesthetic applications you can simply thread a screw directly into an undersized 3D printed hole. An M4 screw could be threaded into a 3.3mm hole for example, and hold rather well. Specialized thread forming screws, similar in look to wood or concrete screws, make this process a lot easier and increase holding strength.

> **Warning:** While simple to implement, threading directly into plastic should not be used for parts that need to be removed. When plastic threads are commonly screwed/unscrewed, they lose their strength extremely quickly. This method is best for combining parts that are never expected to be separated for maintenance or repairs.

Embedded Nuts

Nuts and Locknuts can be embedded into prints, removing the need for a wrench when tightening a screw. Simply inset the shape and size of the nut into a print and drop it in when attaching the part. These should hold very well, especially if placed in mid-print (pausing the print midway through to place a nut embedded inside)



Heat Set Inserts

An increasingly common solution to securing screws to 3d printed parts, heat set inserts are small brass components that can be melted into a 3d printed part using a soldering iron. There are custom soldering iron tips specifically for this purpose. There's lots of sizes available for whatever purpose you need (Amazon, Aliexpress, whatever storefront you prefer), but keep in mind that inserts with a "chamfer" will be a lot easier to put in.

A very important best practice when using heat set inserts is to place them on the print where when the screw is tightened, they will be pulled farther into the print instead of out. This increases longevity.

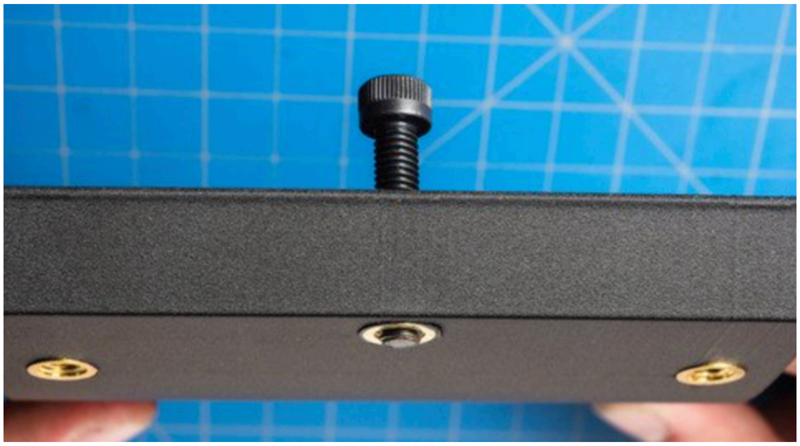


Fig. 34: This part uses heat set inserts as described above, where tightening the screw pulls them farther into the part.

If you'd like a more in-depth guide on using heat set inserts or are considering doing so for your team, go read through Markforged's article on the topic..

Tolerancing Prints

Tolerances are everything when it comes to 3d printing- we're not precision machining parts here, we're laying down plastic and hoping it forms a shape. Differences in temperature, nozzle quality, and airflow can vastly change the size a hole actually is despite what the printers instructions tell it to do. A 4mm hole when printed could end up as actually being 3.6mm. Below are detailed two ways to deal with these tolerance issues for 3d printing.

Designing Tolerances

A common way for teams to deal with prints interacting with each other and hardware is to form a table with a list of commonly used hole sizes for your various applications and then use these hole sizes in their CAD. Numbers depend on your printer and nozzle size, so we would recommend making test prints to see how hardware can fit best in order to form this table. An example test print would be a print with a 2.8mm, 2.9mm, 3.0mm, 3.1mm, and 3.2mm hole to see which best creates an M3 through hole.

This method is generally acceptable if kept well maintained and updated per printer, but it does mean that you have inconsistent hole sizes in CAD that would seem arbitrary to anyone looking at your team's CAD.

Slicer Horizontal Expansion

The technically proper way to deal with tolerances- horizontal expansion compensation. All slicers have horizontal expansion settings to make it so that you can have a 4mm hole turn out as precisely 4mm.

Fig. 35: Example horizontal compensation setting names.

When tuning these, a good starting point is half your nozzle size- if you're using a 0.4mm nozzle, assume 0.2mm expansion and make the compensation numbers -0.2mm.

You will still have to use different numbers for pressfit vs slipfit in cad for this however, so this is not a catch-all method of making it so you can cad parts perfectly to the size you'd like them to be. Constant thoughts about what size something should be to fit hardware properly should always be considered when designing parts.

24.1.7 Specific Skill Guides

Bed Leveling

Note: This guide is primarily for older printers that don't have automatic leveling. If you have a newer printer, this process is likely not needed or there is a more advanced process to follow. Research what processes are available for your printer.

If you have a bed thats adjustable by screws, bed leveling is by far the biggest factor for how well your prints stick across the whole bed. The only tool you should need is a 0.2mm thick piece of metal, a piece of cardstock, or an index card.

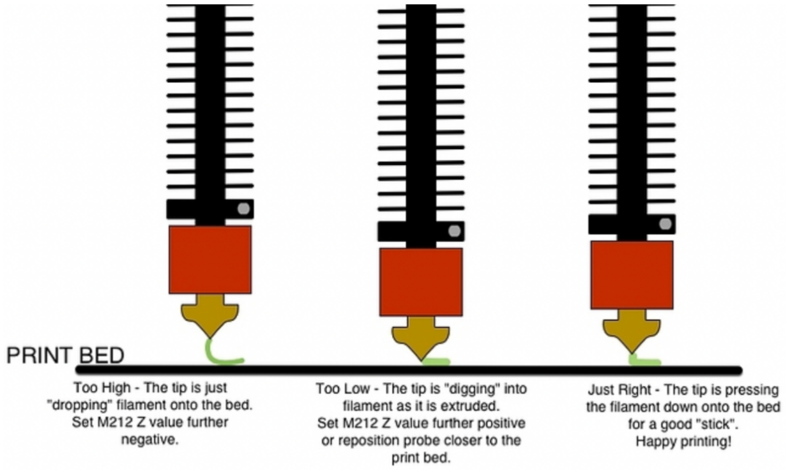
Here's the steps you should follow to level your bed:

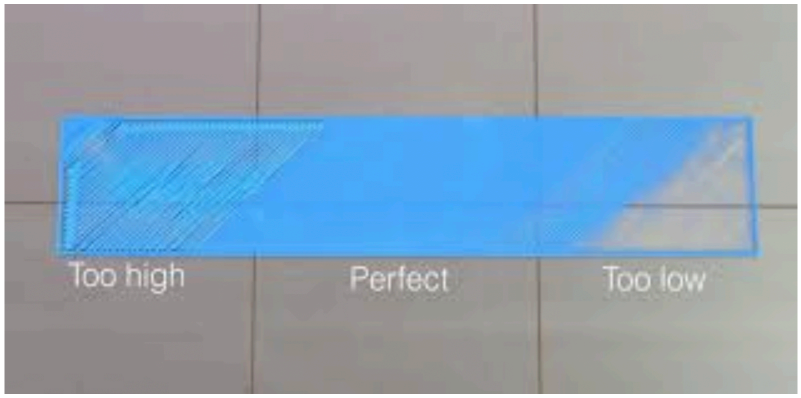
- Ensure that your nozzle is clean and has no extra filament beneath it that would interfere with bed leveling precision.
- Home the Z axis completely, and get it to Z coordinate 0 by using the printer's move commands
- Move the nozzle to just above one of the screws using the printer's move commands (This is done with move commands to make sure that it replicates the movement that the printer will perform when under it's own control)
- Adjust the screw up and down until the 0.2mm material can barely slide with slight friction underneath the nozzle.

- Repeat this process with the other screws available on the bed.
- Move the nozzle to the center of the bed and check bed height using your 0.2mm material. If you leveled all of the screws properly, it should be roughly the same amount of friction.

Z-Offset

Once your printer is leveled between the screws, the bed is flat, but it's not always at exactly the right height. All printers should have a Z offset variable that lets you adjust the height of the nozzle during or before printing. If at the start of your print the bed is either located too high or too low, but remaining consistent (provided you bed leveled), use your printers Z offset feature (also called baby-stepping) to adjust it live while it lays down the first layer. Reference the images below for some good visualizations on how to make sure you're at the proper height!





Print Adhesion

There's a lot of ways that print adhesion can go wrong, and it's one of the biggest things that people new to 3d printing struggle with. Bed Leveling is often the biggest, but if you just can't get that one part to stick even after a perfect bed level, try some of the things below depending on your build surface!

Increase Adhesion on Glass/Sticker/Powder Coated Bed Surfaces

If you're having bed adhesion issues with any of these surfaces, the easiest way to deal with it is PVA Glue. School glue and gluesticks contain a plastic called PVA that can help bed adhesion by providing a film for the part you're printing to hold onto.

- **Gluestick**: If you use gluestick, spread a thin layer of it over the area you're printing on, heat it up, and wait for the gluestick to mainly become invisible before starting the print.
- **School Glue**: If you choose to use school glue, mix it thoroughly 50/50 with tap water in a spray bottle, and spray a thin layer on the bed. Heat it up and begin your print.

If you are using a Glass bed, hairspray can also serve this function, but PVA glue is a more consistent surface and is more effective.

> **Warning:** When using any form of glue on your bed, it's important to clean it every couple prints. Glue buildup can significantly change the height and properties of your printing surface, and sometimes even make it worse. Soapy Water or Isopropyl Alcohol will work well for these purposes.

Increase Adhesion on PEX/PEI Bed Surfaces

If you have a PEX or PEI build surface and can't get anything to stick, it is recommended to clean it off with isopropyl alcohol and then rough it up slightly with steel wool and then proceed using PVA glue of some sort (described above). This will provide slightly more texture for a print to grip onto.

Free Stuck Prints

While having too little bed adhesion is a pretty common problem, having too much can be bad as well, with prints sticking too well. If you encounter an issue like this, there's 2 easy ways to help prints get off the bed.

- **Cooling**: Due to thermal expansion, letting the print cool completely will help it remove itself from the bed. If that wasn't quite enough, consider putting your entire printer bed in the fridge or freezer (or even just outside if it's winter) to contract the bed and part slightly more and possibly free them.
- **Water**: It sounds crazy, but if you put a few drops of water or a light spray around the part on the bed, the surface tension of the water will help lift up the part slightly and remove it from the bed.

Extruder Calibration (E-Steps)

If you're noticing any issues with over/under extrusion or if you just got a new printer, it's highly advisable that you tune your e-steps. E-Steps dictate how many steps the extruder motor takes to extrude a certain amount of filament. This procedure is best covered in videos, so go take a look at Thomas Sanladerer's video on the topic.

Temperature Tuning

In order to have strong, good quality 3d printed parts, you need to get your temperatures and speeds right. Every filament that you buy will have a range of temperatures that it thinks are acceptable to print it at, but the one that works best for you depends on all sorts of conditions. Start in the middle of that range, and work from there.

Temperature Based Underextrusion and Overextrusion

This is the key to tuning filaments. There's a certain sweet temperature when printing that helps lay the perfect amount of plastic down to complete your part. Underextrusion looks like it's not quite able to get continuous plastic out (you need to raise your temperature), and Overextrusion looks like it's simply trying to melt too much (drooping around the sides, so you need to lower your temp).

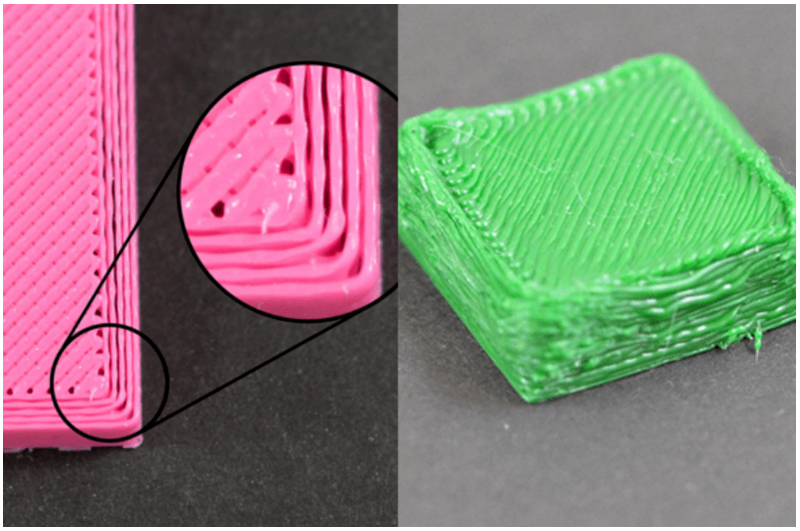


Fig. 36: Underextrusion on the left, Overextrusion on the right.

Note: Extra about PLA: PLA is a really easy filament to work with, unless you're trying to print it fast, in which case it becomes the worst filament you've ever encountered. PLA is a filament that requires a lot of cooling, otherwise it becomes really messy really quickly. Temperature issues can appear in PLA due to either lack of cooling or improper temperatures, so consider both of these factors when tuning for PLA. Consider dropping your printing speed to isolate issues to just temperature.

Basic Post-Processing

Support Removal

Support material removal is a basic form of post processing where, as the name suggests, you remove support material from your print. This can typically be done easily with either just your fingers or a pair of pliers/flush cutters, however, sometimes removing support from small features or holes can be difficult. This is why it's recommended to design away from using supports, and if you must use them, to set up your slicing settings properly in order to make them easy to remove.

Drilling Out Holes

Drilling out printed holes are typically used in order to widen screw holes to achieve a loose fit. This can be done with any drill and properly sized drill bit, however take your time while drilling to ensure that the drill bit is lined up properly to guaruntee that drilled holes are straight.

Brim Removal

Brims are used to have more surface area for your print to contact the build plate. To remove them, you typically just use your fingers, however, if your Z-Offset is too low, it may be easier to use a deburring tool to remove the inner layer lines of the brim.

24.1.8 Troubleshooting Common Issues

Here are some hyperlinks to common issues with printers that are worth looking at during a troubleshooting process.

- *Underextrusion and Overextrusion*
- *Prints Not Sticking*
- *Bed Not Level*
- *Parts Not Fitting*

24.1.9 Volunteer Special Thanks

The *FIRST* Tech Challenge staff would like to extend a special thanks to the following volunteers for their hard work and dedication toward this project:

- Lucas Y., Team 16461
- Davy Hallihan, Team 16461
- Ryan Driemeyer, Team 1002

Chapter 25

Common Team FAQs

If you're looking for quick answers regarding the many facets of being a team from registration to competition to judging. Please refer to these official questions and answers to guide you through the season. If you need further clarification navigate to https://www.firstinspires.org/ to Live Chat or ask game specific questions on the Game Q&A .

25.1 Dashboard/Registration FAQs

Why can't I invite my team members to my team? Each team is required to have two YPP screened lead coaches to see the Invite Youth Member contact option

I just registered with Pitsco, but I still see a balance due on my dashboard, why? Pitsco submits payments to us electronically. This process can take 24-48 hours to complete.

25.2 System FAQs

Who can log into the *FIRST* Tech Challenge Scoring system? Lead Coach 1 & 2

Where can I find events and event results in my region? https://ftc-events.firstinspires.org/ is the source for FIRST-official team, events and event results information for *FIRST* Tech Challenge

25.3 Judging FAQs

What do I need to bring to my Judging Appointment? Teams should bring their robot, Portfolio and as many members of the team who want to participate in the presentation. Please note that at some events, the Portfolio is collected when the team checks in for the event.

What feedback will we receive from the judges? Judges complete the feedback form immediately after the team has completed their formal interview. Feedback is limited to the initial formal interview and does not include the team's performance in follow-up Pit interviews or their Portfolio.

Is my team required to prepare a 5 minute presentation? Teams are not required to prepare a 5 minute presentation, but teams should let the judges know they do not have a formal presentation when they enter the room. The judges will begin to ask the team questions at the beginning of the interview.

25.4 Competition FAQs

Who on my team needs to be with the robot for inspection? It depends on which inspection station you're visiting and how your event is configured. The inspectors at Field Inspection like to see the Drive Team, Human Player (if there is one), and Drive Coach. The inspectors at Robot Inspection really just want to see team members who have the best idea of what's going on with the robot (mechanically and electrically). Look at your inspection sheets, you can generally determine what you're doing based on the checklist.

Why aren't you going to replay that match? There are only certain situations that warrant replaying a match. Consult the Competition Manual to understand what conditions and processes can result in a replay. Typically unless something went wrong with the field or field staff there will not be grounds for a replay. A malfunctioning robot typically does not provide grounds for a replay.

Why did you replay a match for someone else, but not us? The situation was different. If necessary, teams can talk with the head referee in the competition area Question Box.

Why won't you fix that score? We have video (or photographs) to prove the score is wrong! Teams can go the Question Box so they can discuss this issue with the head referee. No photographs or videos will be reviewed.

Why don't you fix/cleanup the wireless environment? It's obvious the wireless environment is disruptive and causing disconnects. Teams can work with the event FTA or Event Director if they have questions about the WiFi environment.

What do we do if we think the scoring referees scored our match wrong or the scorekeeper put the wrong score into the computer. If a team has questions about a match outcome, they should send one student representative to the Question Box to talk to the Head Referee (do not interrupt matches for this conversation). If the referees agree that they made a mistake, they can correct it. If the referees are confident in their score, the team should accept that decision. Check the Competition Manual for more information about key volunteer roles, tournament operations, and how to use the Question Box.

25.5 Technology FAQs

I'm getting a weird error, where can I go to get help with fixing this? The best place to go for help is the ftc-community platform. The ftc-community platform is a community place to ask questions that is monitored by a variety of knowledgeable folks who can likely help you with your questions!

Reviewed by *FIRST* Tech Challenge Game Design Committee

Chapter 26

FIRST Tech Challenge Team Complimentary Software

As a benefit of being a *FIRST* Tech Challenge Team, some sponsors of *FIRST* programs have generously provided complimentary software, licenses, and/or support to teams. These licenses are generally good for a period of one year and many must be renewed annually. Sometimes sponsored content require the use of a donation voucher provided in your FIRST Dashboard team account, sometimes sponsors merely require your Team number and contact information, and some software is simply provided free of charge with no proof of registration requirement. Read details below for each product for more information.

Autodesk

What:

- Autodesk CAD software (including Fusion 360, Inventor, 3ds Max, and more…)

Expires: Contact Autodesk for exact details.

Access Codes: N/A

To Access: Check out the *Autodesk page* for details.

Dassault Systèmes

What:

- Dassault Systèmes CAD software (including SOLIDWORKS and the **3D** EXPERIENCE Platform)

Expires: Contact Dassault Systèmes for exact details.

Access Codes: [COMING SOON]

To Access: Check out the *Dassault Systèmes page* for details.

JIRA

What:

- Atlassian JIRA is the #1 software development tool used by Agile teams
  - Track, release, and manage world-class software
  - Forever free for up to 10 users
  - Get from “due” to “done”
  - Unlimited projects and boards

Expires: N/A

Access Codes: N/A

To Access: Create free account on Atlassian.com

Miro

What:

- Miro visual platform for Project Management and Ideation
  - 3 editable boards
  - Premade templates
  - Core integrations
  - Basic attention management

Expires: N/A

Access Codes: N/A

To Access: Create free account on Miro.com

PTC

What:

- PTC Software Access (including Creo, OnShape, etc…)

Expires: Contact PTC for exact details.

Access Codes: N/A

To Access: Check out the *PTC page* for details.

Trello

What:

- Atlassian Trello brings all your tasks, teammates, and tools together
  - Boards, Lists, and Cards help organize and keep team on task
  - Free Workspaces can have 10 open boards (limited to 10 collaborators)
  - Invite viewers to the boards

Expires: N/A

Access Codes: N/A

To Access: Create free account on trello.com

Chapter 27

FIRST Tech Challenge Team Discounts

As a benefit of being a FIRST Tech Challenge Team, some vendors and sponsors of *FIRST* programs have generously provided discounts on products to teams. These discounts may be for specific categories of products, or may apply site-wide, and may require an application process or a season-specific or team-specific promo code. Read details below for each vendor for more information.

Team Grant Opportunities

What:

- FIRST Tech Challenge Team Grants

Expires: See each individual grant opportunity.

To Apply: Check out the list of current Team Grants available and see if you're eligible!

FIRST Storefront

What:

- All FIRST Tech Challenge Teams are eligible for discounts on control equipment, electronics, and starter kits.
- Teams are limited to one purchase per item per category each season.

Expires: Good for the current FIRST Tech Challenge competition season.

Promo Code: N/A

To Use: See the Kit of Parts PDF for instructions on how to purchase discounted parts.

Chapter 28

Booklets

More 'bite-sized' versions of the FTC Docs. These are meant to be printed out and used as a reference for teams.

28.1 Control System

- Control System

28.2 Manufacturing

- 3D Printing Guide
- Electrostatic Discharge

28.3 Programming Resources

- Programming Resources
- Android Studios Manual
- OnBot Java Manual
- Blocks Manual
- April Tags
- Advanced User
- FTC SDK Guide

28.4 Contributing

- Contributor Guide

Chapter 29

Site Feedback Form

We would love to hear your feedback on this site (*FIRST* Tech Challenge Docs). Please fill out the form below and we will try our best to incorporate your feedback into the documentation.

Note: If you're not able to see the form below, check your browser extensions for messages regarding blocked content. Some extensions such as Privacy Badger may require site exceptions for "Microsoft Forms" in order to correctly load the feedback form content.

Feedback Form

Chapter 30

Contributing to FTC Docs

30.1 Contribution Guidelines

Contributors are the lifeblood of the project. We welcome contributions but remind everyone to be a *Gracious Professional*.

30.1.1 Creating a PR

PRs should be made to the FTC Docs repo on GitHub. Your title should aim to describe the purpose of your PR in a *concise* manner. For more information on creating a PR, see this

30.1.2 Creating an Issue

There are two types of issues: bugs and feature requests. A bug report is an issue that describes a problem with the documentation. A feature request is an issue that describes a new feature that should be added to the documentation. Before creating either make sure to check for duplicates. If you find a duplicate, comment on the issue and add your input where possible. If possible we would love to see a PR that fixes the issue. If you are unsure how to fix the issue that is perfectly alright.

30.1.3 Bug Reports

- A description of the bug
- Expected behavior
- Steps to reproduce the bug (If applicable)
- Screenshots (If applicable)

30.1.4 Feature Requests

- A description of the feature
- Why you think this feature should be added
- Screenshots (If applicable)

30.1.5 Colophon

FTC Docs is built with Sphinx using a theme provided by Read the Docs.

The Dark theme is provided by the FTC Docs development team.

30.2 FTC Docs Style Guide

This guide contains the various reStructuredText (RST) and Sphinx specific guidelines for the FTC Docs project. reStructuredText is the default plaintext markup language used by Sphinx. Sphinx is a documentation generator. Sphinx converts reStructuredText files into HTML web pages. Read the Docs is a documentation hosting platform and provides the base Sphinx theme for FTC Docs.

Contents

30.2.1 Accessibility

A new emphasis in FTC Docs is to improve the accessibility of FTC Docs. The Americans with Disabilities Act (ADA) is a federal civil rights law that prohibits discrimination against people with disabilities in everyday activities.

What is Web Accessibility

Note: This section is copied from: https://www.w3.org/WAI/fundamentals/accessibility-intro/.

Web accessibility means that websites, tools, and technologies are designed and developed so that people with disabilities can use them. More specifically, people can:

- perceive, understand, navigate, and interact with the Web
- contribute to the Web

Web accessibility encompasses all disabilities that affect access to the Web, including:

- auditory
- cognitive
- neurological
- physical
- speech

- visual

Web accessibility also benefits people without disabilities, for example:

- people using mobile phones, smart watches, smart TVs, and other devices with small screens, different input modes, etc.
- older people with changing abilities due to aging
- people with “temporary disabilities” such as a broken arm or lost glasses
- people with “situational limitations” such as in bright sunlight or in an environment where they cannot listen to audio
- people using a slow Internet connection, or who have limited or expensive bandwidth

Web Content Accessibility Guidelines

Web Content Accessibility Guidelines (WCAG) are a set of recommendations for making Web content more accessible, primarily for people with disabilities.

FTC Docs is not completely compliant with WCAG. Our goal to address the Level A success criteria to remove accessibility barriers. Then move to meet the level AA criteria to improve that accessibility. See https://www.w3.org/WAI/WCAG22/quickref/?versions=2.2

The WCAG documents explain how to make web content more accessible to people with disabilities. Web ‘content’ generally refers to the information in a web page or web application,

Please refer to the *FTC Docs Accessibility Guidelines* section of FTC Docs for how to improve the accessibility of content created for FTC Docs.

The following section provide the WCAG principles that the accessibility guidelines are based on.

Principle 1 – Perceivable

Information and user interface components must be presentable to users in ways they can perceive. For the vision impaired this includes providing text alternative for non-text content like images. It could include Closed captioning which is a form of subtitling, a process of displaying text on a television, video screen, or other visual display to provide additional or interpretive information, where the viewer is given the choice of whether the text is displayed.

Create content that can be presented in different ways (for example simpler layout) without losing information or structure.

Make it easier for users to see and hear content including separating foreground from background.

Principle 2 – Operable

User interface components and navigation must be operable. Make all functionality available from a keyboard. Provide users enough time to read and use content. Do not design content in a way that is known to cause seizures or physical reactions. Example: flashing content. Provide ways to help users navigate, find content, and determine where they are.

Make it easier for users to operate functionality through various inputs beyond keyboard.

Principle 3 – Understandable

Information and the operation of the user interface must be understandable. Make text content readable and understandable. Make Web pages appear and operate in predictable ways.

Principle 4 – Robust

Content must be robust enough that it can be interpreted by a wide variety of user agents, including assistive technologies. Help users avoid and correct mistakes.

This success criterion is primarily for Web authors who develop or script their own user interface components. For example, standard HTML controls already meet this success criterion when used according to specification.

30.2.2 reStructuredText Documents

Example Document

```rst
Title
=====

This is an example article.
This is a paragraph.

Here is some code:

.. code:: java

   System.out.println("Hello World");

This is how to include an image:

.. image:: images/BlocksPicture1.jpg
   :alt: Blocks Programming Tool showing a graphical Blocks program.

This is how to include an image with caption:

.. figure:: images/examplemulticolorplates.png
   :alt: 6 multicolor square 3d printed logos.

   This FTC Team printed their sponsors logos in multiple colors to represent them!

Section
-------

This is a section!

Sub-section
^^^^^^^^^^^

This is a sub-section!
```

Note: If you are having issues editing files with the `.rst` extension, the recommended text editor is VS Code with the reStructuredText extension.

Document Filenames

This is for the files that make up the pages of FTC Docs. Use only lowercase alphanumeric characters and - (minus) symbol in the file name. Example: `style-guide.rst`

For documents that will have an identical software/hardware name, append "-hardware" or "-software" to the end of the document name. example: `ultrasonics-hardware.rst`

Suffix filenames with the `.rst` extension.

30.2.3 Titles and Headings

> **Attention:** We should also mention things related to document structure and how we want FTC Docs headings.

30.2.4 Text

> **Attention:** talk about FTC Docs text, paragraphs, lists, tables, admonitions etc. This is where we just have guidelines and don't repeat info in the tutorials or reference guide. Example: we can encourage use of the list style of RST table and avoid the ascii box style of table. Also talk about plain language: https://evolvingweb.com/blog/plain-language-guide-how-write-inclusive-digital-content-2024

30.2.5 Links

Effective link text:

- Avoid link text such as "click here" or "learn more." These call-to-actions do not provide any relevant information to someone using a screen reader.
- Links should be unique and describe where it takes you. If you have multiple links that look or sound similar (but point to different sections), use different words for each link.
- Links to files (e.g. Microsoft Word, PDF, etc.) should also indicate the file type or destination within the link text.
- Avoid linking long URLs. Longer, less intelligible URLs used as link text might be difficult to comprehend with assistive technology.
- Links should be clear and concise. Avoid linking entire sentences or paragraphs.

Tip: See this guide on writing hyperlinks for more information.

Internal Links

Internal Links will be auto-generated based on the ReStructuredText filename and section title. For example, here are several ways to link to sections and documents.

- Use this format to reference a section of the same document: `` `Images`_ `` Note the single underscore. This renders to the link *Images* which is a section further down in this document.
- Use this format to reference the top-level of a document. You can use relative paths `` :doc:`image-and-figure-details` `` renders to *Image and Figure Details*. Or to use absolute paths, put a

forward slash at the beginning of the path ``:doc:`/apriltag/vision_portal/visionportal_webcams/visionportal-webcams` `` renders to *Webcams for Vision Portal*. Note that the link text rendered is the main section title of the target page regardless of the target filename.

The general way to reference a section in another document is to add a label in front of the section in that other document. Note the leading underscore and trailing colon that surround the label. The label must be unique across all of FTC Docs. You can also reference a Figure with the label method.

```
.. _imu axes def:

Axes Definition
---------------
```

Then you reference the label by using `:ref:` and surrounding the label with back ticks as follows:

```
partial sentence including :ref:`imu axes def`

another reference to :ref:`IMU or robot axes <imu axes def>`
```

The second `:ref:` shows a format that lets you specify the link text, otherwise the section heading is used for the link text. This looks like the following:

partial sentence including *Axes Definition*	This shows the section heading.
another reference to *IMU or robot axes*	This shows custom link text.

When using `:ref:` or `:doc:` you may customize the displayed text by surrounding the actual link with angle brackets <> and adding the custom text between the first backtick ` and the first angle bracket <, leaving a space between the text and bracket. For example ``:ref:`RC Overview <control_hard_compon/rc_components/index:Robot Controller Overview>` `` renders to *RC Overview*. This is a link to the Robot Controller Overview section of the index in the rc_components folder.

External Links

Links to other websites and even to the main *FIRST* Inspires site are call *external links*. It's possible to create a link by entering the URL in the text https://www.firstinspires.org/resource-library/ftc/game-and-season-info. Sphinx will build a link when it encounters a URL. But that is not the preferred approach.

Use descriptive link text rather than just embedding a URL. Use the following RST syntax:

```
`Game and Season Materials <https://www.firstinspires.org/resource-library/ftc/game-and-season-info>`_
```

Which looks like: Game and Season Materials

FTC Docs has chosen to open links to external sites in new tabs. This is done with Javascript. We mitigate this somewhat by adding an icon that indicates the link is to an external site and add screen reader only text.

Links to Files

You can directly link to files such as a PDF, but that is an accessibility problem. The issue is the context switch from web browsing to suddenly having to deal with a PDF that has probably opened in a new tab/window without any warning. FTC Docs contains quite a few links to PDFs that should be make more accessible.

The recommended approach to linking to files is to include in the link a warning that the link is actually a file, the file type, and if possible the file size. Ideally that information is in text and included in the link text portion of the link so that a screen reader would read that information and let the user decide if they want to follow the link.

Simple example of a link to a PDF.

RST Code:

```
`Field Setup Guide (PDF, 4.5 MB) <https://ftc-resources.firstinspires.org/file/ftc/game/fieldguide>
↪`__
```

Which looks like:

Field Setup Guide (PDF, 4.5 MB)

Generally in FTC Docs we link to file to enable them to be downloaded for printing or offline viewing. In that case, the user is downloading the file, which is an action, so a button is appropriate. Buttons are appropriate user interface components for user actions. Using a button as a link will visually distinguish a file link from a regular link.

The following RST example show a sentence that precedes the button to give context. We include the document name as screen readers and keyboard only users might tab to the link so we need to indicate in the link what file will be downloadable.

```
Use the following button link to download a PDF of the Field Setup Guide from the *FIRST* Website:

.. button-link:: https://ftc-resources.firstinspires.org/file/ftc/game/fieldguide
   :color: primary

   Download Field Setup Guide PDF, 4.5 MB
```

This looks like:

Use the following button link to download a PDF of the Field Setup Guide from the *FIRST* Website:
Download Field Setup Guide PDF, 4.5 MB

The preferred approach when linking to files is to create what is called a gateway page. The gateway page would describes the file, perhaps giving a summary of the content. This lets the reader decide if it's worth taking the trouble to view or download the file. Ideally, all references to the file elsewhere on the website link to the gateway page which then links to the file.

External reference: https://www.nngroup.com/articles/gateway-pages-prevent-pdf-shock/

Here's a gateway page example for the Field Setup Guide PDF.

The Field Setup Guide has the official instructions for assembling and setting up a *FIRST* Tech Challenge field. Typically there are assembly instructions that build structures that then have setup instructions for placing on the field. There are also teardown instructions that indicate how to take apart the field for storage or transport.

The guide typically has the following sections:

- A list of all tools required for assembly and setup, some tools are only for assembly or for setup.
- A list all the game elements and scoring elements with the quantity of each.
- Instructions for setup of the field perimeter and field tiles.
- Step by step instructions for assembling parts and setting them on the field.
- Most games have tape lines on the field to mark locations or areas of the game. There are also taped areas outside the field for the Alliances, and sometimes for game areas.
- Most games have AprilTags placed around the field that can be used for robot navigation.
- Finally, there are teardown instructions that indicate how to take the field down for storage or transport.

Use the following button link to download a PDF of the Field Setup Guide from the *FIRST* Website:

Download Field Setup Guide PDF, 4.5 MB

30.2.6 Images

Images usually support the text on a page. They provide visual reinforcement. Sometimes an image is the primary source of meaning for the content. It's hard to talk about AprilTags without images of what an AprilTag looks like.

For FTC Docs, please don't add *Decorative Images* that are just for visual presentation. Images should relate to the content of the page.

See *Image Files* for file formats, size, and where to place images the FTC Docs folders.

Simple Images

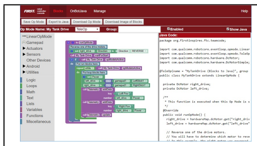
Use the following `.. image` directive when the image does not require a visible text caption. For accessibility, please add an `:alt:` option to the image directive with a functional description of the image. This description will not be visible on a web page or PDF, but will be spoken out loud by screen reader software.

```
.. image:: images/BlocksPicture1.jpg
   :alt: Blocks Programming Tool showing a graphical Blocks program.
```

The `:alt:` line is indented three spaces.

The description should be functional. Describe the image for someone who cannot see it.

Here's what the web page for an image looks like (but reduced in size for this example). The image is a screen shot of the Blocks programming tool on a page that talks about the various programming tools available.

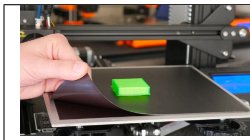


The alt text reads *Blocks Programming Tool showing a graphical Blocks program.* The alt text is not visible in HTML or PDF, but would be spoken aloud by screen reading software. Alt text is displayed in the browser if the image failed to load or the user had selected to not load images because they are on a low bandwidth internet connection.

FTC Docs has many diagrams. Some are simple like this one for a control system diagram on a page that introduces the control system.

The alt text reads *Two gamepads are connected to a phone, the phone is connected by WiFi Direct to another phone on the robot*.

FTC Docs also has photographs. These also need alt text to describe the image. This next example is on a page discussing hardware tradeoffs including stiff and flexible printer beds.



The alt text reads *An image showing how flexible beds peel off of the bed.*

Alt Text Guidelines

- Keep it short. Alt text shouldn't be much longer than around 150 characters.
- Do not include words like 'image' or 'photo' at the beginning.
- End alt text with a period, even if it isn't a full sentence. The period ensures that the screen reader pauses after reading the alt text.
- Front load alt text with the most important words, to help users make a quick and informed decision about whether it's worth listening to the rest of the alt text before moving on.
- Always include an alt attribute. Otherwise, screen readers might announce the image file name.
- Avoid technical jargon and abbreviations unless users are certain to understand them.

If you need more than a sentence or two to describe the image, see *Complex Images*.

Tip: For assistance with alt text descriptions, see *Alt Text*.

Important: If you are editting an existing page that has an `.. image` or `.. figure` directive with no `:alt:` option, please take a moment to add the `:alt:` option with a functional description of the image.

Images With Captions

Use the `..figure` directive when the image requires a visible text caption. Sphinx place the caption text below the image. The caption text is in italic font.

Photo credits are an example of when you need a caption. You should also use a caption when you need editorial or illustrative text to highlight something about the image to the reader or to connect the image to the surrounding text content. This could include the who, what, when, where, and/or why of an image. You might draw the readers attention to some detail of the image, or what is important in the image. Captions can be longer than one line if needed, but should generally be kept short.

Please create alt text even though there is a caption. The alt text and caption should be different because a screen reader will read both. One way to think about this is the alt text should be functional and the caption should editorial or illustrative.

Do not add the word "Figure" to the caption, or "Figure 1", etc. The PDF writer will automatically add "Fig." and a number to all captions.

If you need to reference the figure in the text of the page, use the leading text of the caption if possible, or some unique short identifier or description of the image. Try not to reference the figure by position (such as "see the image above") as visually impaired persons cannot see the position. Also, the PDF writer might move the figure out of context with the surrounding text.

The following example is on a page about 3D printed parts in a section about Robot Aesthetics.

```
.. figure:: images/examplemulticolorplates.png
   :alt: 6 multicolor square 3d printed logos.

   This FTC Team printed their sponsors logos in multiple colors to represent them!
```

Note that `:alt:` and caption are both indented 3 spaces. A blank line is required between `:alt:` and the caption.

Here's what this looks like (with image reduced in size for this example).

Fig. 1: This FTC Team printed their sponsors logos in multiple colors to represent them!

The alt text for the image reads *6 multicolor square 3d printed logos*.

This is a good example of a functional alt text description for a screen reader followed by an editorial caption that is visible.

Complex Images

To make images accessible for the visually impaired, we need to provide a text description of the image. For complex images, you might need a whole paragraph to describe the image. Alt text is not supposed to be more than a sentence or two, we need to provide something short in the alt text knowing that a long description follows.

To include a complex image use the figure directive. Set the `:alt:` text to a short functional description, perhaps a summary of the long description. Set the caption to be an editorial or illustrative comment on the image. See the guidance about alt text and captions in the preceding section. The long description is then added as a paragraph after the caption in the figure directive.

Try to adjust the alt text, caption, and long description so that the text flows such that it would make sense if read aloud by a screen reader.

In the following reStructuredText source example the figure directive has alt text, followed by a one line caption. A descriptive paragraph is added after the caption. It is indented the same as the caption. There is a blank line before and after the caption.

```
.. figure:: images/into-the-deep-field.png
   :alt: A square field with X, Y and Z axes shown.

   The Cascade Effect game field

   In a square field configuration the two Alliances face each other across the field.
   The field is oriented such that the Red Wall is on the right as seen
   from the audience, and the blue wall will be on the left.
```

(continues on next page)

(continued from previous page)

```
The Y axis points across the field from the Red Wall to the blue wall.
The X axis points away from the audience to the rear of the field.
```

Here's what a complex image looks like (with image reduced in size for this example).

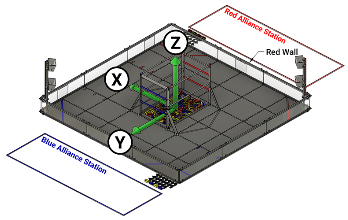


Fig. 2: Into The Deep game field

In a square field configuration the two Alliances face each other across the field. The field is oriented such that the Red Wall is on the right as seen from the audience, and the blue wall will be on the left. The Y axis points across the field from the Red Wall to the blue wall. The X axis points away from the audience to the rear of the field.

The alt text for the example image reads *A square field with X, Y and Z axes shown.* This is a short functional description that a screen reader would say.

The caption is an editorial comment that this field is from the Cascade Effect game. It is not the same as the alt text.

The paragraph after the caption completely describes the image for those who cannot see it. A complete text description is also important for those persons who have trouble processing or understanding visual information. It can help them understand the image they are seeing.

For another example of a complex image see *Complex Image Example*.

Image Files

Image files should be stored in an `images` sub directory in the folder of the current document. This allows the document to reference the image as follows: `.. image:: images/my-image.png`.

Example: The document **field-coordinate-system.rst** is in the **game_specific_resources\field_coordinate_system** folder. The images for that page are stored in the folder **game_specific_resources\field_coordinate_system\images**.

Image file names should follow the naming scheme of **short-description.ext**, where the name of the image is a short description of what the image shows. This should be less than 24 characters. File name extensions should be **.png** or **.jpg**.

Image file formats should be Portable Network Graphics(PNG) or Joint Photographic Experts Group(JPEG).

Warning: Images with the extension .gif and .svg are not supported in PDF format.

Images (including vector graphics) should be less than 500 kilobytes in size and no more than 1000 pixels in width. Please make use of a smaller resolution and more efficient compression algorithms. This facilitates reasonable web page loading for those with slow internet connections. Our HTML documents have a maximum width of 1000 pixels for desktop browsing so image width should be 1000 pixels or less.

An exception to image size is images like the control system diagrams in the *Robot Controller Overview*. Those diagrams are over 2500 pixels wide and greater than 500kb in size. However, that extra resolution is required to properly view the details of all the components and the connections. If those were reduced in resolution, or too heavily compressed as a jpg, relevant details might be lost or hard to see.

30.2.7 More Information

- *FTC Docs Accessibility Guidelines*
- *Image and Figure Details*

FTC Docs Accessibility Guidelines

The Web Content Accessibility Guidelines (WCAG) are part of a series of web accessibility guidelines published by the Web Accessibility Initiative (WAI) of the World Wide Web Consortium (W3C), the main international standards organization for the Internet. They are a set of recommendations for making Web content more accessible, primarily for people with disabilities—but also for all user agents, including highly limited devices, such as mobile phones.

As of December 2024, FTC Docs meets some of the WCAG success criteria, but fails at others. We have good page titles and make use of section headers to organize our content. We fail at accessible site navigation: we don't have a skip to main content link among other problems.

> **Attention:** I think This will be a long page. It's a rough draft. The idea is to document from the WCAG perspective what is important and relate that to FTC Docs. The following sections are just outlines that will get filled in.
>
> See https://docs.google.com/document/d/1_-seP4lzyWeXwlTSd1wlaon35Zm-YqKZUro3rni0Ap4/edit?usp=sharing for an ADA Compliance review (which is primarily a WCAG review).
>
> See https://docs.google.com/spreadsheets/d/1Z_i5QImEdU5CA1j-hBAnpV2jqXNBhC-Rx-bLb0uJnAM/edit?usp=sharing for a more detailed WCAG analysis of the 30 level A success criteria.
>
> See https://docs.google.com/document/d/1b4BjhLhSdbSScADhpJgLYIyqK69RHIxxOdXWzczOBUw/edit?usp=sharing for a proposed plan to address FTC Docs accessibility.

Contents

Principle 1 – Perceivable

Information and user interface components must be presentable to users in ways they can perceive.

FTC Docs mostly consists of web pages with text and images. It should be feasible to make most FTC Docs content perceivable to most users.

Guideline 1.1 – Text Alternatives

Provide text alternatives for any non-text content so that it can be changed into other forms people need, such as large print, braille, speech, symbols or simpler language.

For FTC Docs this should be all content images, plus any user interface images (such as the dark-light theme switcher icon).

FTC Docs To Do

- All content images require Alt text. Some pages do not have alt text for images. For those page switch alt text the text should be reviewed as the alt text for many is not appropriate. For Example, a single word is not a description.
- FTC Docs needs to provide text alternatives to the theme switch icon.
- We need to fix the alt text link to the FTC Logo which says "logo". It should read "FTC Docs Home" or just "Home" as it is a functional link to take users back to the home page.

Guideline 1.2 – Time-based Media

Provide alternatives for time-based media.

FTC Docs (and the *FIRST* Inspires site) make use of videos hosted on YouTube. Those videos have Closed Captioning which is good for those persons who cannot hear the persons speaking in the videos.

FTC Docs To Do

- We do not have a a Described Video narrator which would would describe the action in our videos for the visually impaired. This is more of a main *FIRST* Inspires website issue.
- The one video linked is the About FTC video. We've added a ARIA note that indicates how to start the video and offers a brief description of the video content. e.g. that students and mentors are talking about FTC (not exactly Described Video, but it does give some indication of the relevant visual content).

Guideline 1.3 – Adaptable

Create content that can be presented in different ways (for example simpler layout) without losing information or structure.

FTC Docs mostly does this already. An example of this is that we have a desktop and mobile version of the web site. We also publish a PDF version of FTC Docs.

FTC Docs To Do

- We have some icons (like the theme switcher) that probably should have a visible text description. Icons alone are not good accessible user interface.

Guideline 1.4 – Distinguishable

Make it easier for users to see and hear content including separating foreground from background.

FTC Docs To Do

- FTC Docs uses color alone to mark links, it really needs to underline links, and provide for all the link features like Hover and Visited.
- For FTC Docs we need to review our use of color for both the light and dark themes. Grey text on dark or light backgrounds may not be of sufficient contrast for those person with vision problems. RST "Admonitions" like Notes also have contrast issues due to the colors of the background and text.
- The footer (as of Dec. 2024) also has a problem with contrast.

Principle 2 – Operable

User interface components and navigation must be operable.

This may be hard to fix in FTC Docs without changing the Sphinx or Read the Docs templates, perhaps even requiring changes to how they work. Sphinx and Read the Docs are not designed for accessibility.

Some things like link spacing and how we link to things are under our control.

FTC Docs To Do

- Do not make use of Access Keys, or make them adjustable. FTC Docs sets Next and Previous keys. We have N=next and P=prev accesskey shortcuts that can't be modified and are active when the next previous buttons DO NOT have focus.
- The footer links are too close together, making them a touch target problem (making it more likely a person would press the wrong link by accident).
- We do not have a skip-to-main-content link. Or a way to skip or bypass the sidebar nav links.
- We link to some PDF's without warning the user. We might need to warn/indicate links external to ftc-docs. I have been surprised a few times when links I thought would be an ftcdocs page actually took me to a PDF or to the FIRST Inspires main site.
- FTC Docs has chosen to open links to external sites in new tabs. This is done with Javascript.

  This does preserve your current location in FTC Docs and may be convenient for sighted users. This is an accessibility issue related to unexpected context switching, and creates a new browser tab that some users might have trouble noticing or closing and the back browser command doesn't work. We mitigate this somewhat by adding an icon that indicates the link is to an external site. That can be styled with CSS and a span is added with has text that is only for screen readers to say "external". For FTC Docs we also add "link opens in a new tab" to warn screen reader users that the link will open a new tab.

  The Persona Pages are bad for this. There are grid button links that sometimes take you to a ftc-docs page but often take you to another site with no warning. Ideally all Persona pages should link to ftc-docs pages, some of which might be Gateway Pages to the main FIRST site.

Principle 3 – Understandable

Information and the operation of the user interface must be understandable.

Make text content readable and understandable.

- Insert something about plain language. See https://evolvingweb.com/blog/plain-language-guide-how-write-inclusive-digital-co

Make Web pages appear and operate in predictable ways.

FTC Docs To Do

- Some pages (like the old Field Coordinate System page) had acronyms and excess punctuation that screen readers had trouble with. Revising the text to make it more accessible would make it more readable and understandable for all users. See the Plain Language guide.
- The sidebar is not predicable to an inexperienced user or a visually impaired person. When a link is clicked the sidebar redraws itself and grabs focus. I think the focus should be on the content of the link destination.
- We also have a weird CAPTCHA that pops up unexpectedly and with a complete context switch. I've noticed it in the search box. There is also a CAPTCHA related to the submit feedback form. There may not be much we can do about that except verify it works with screen readers and keyboard only users.

Principle 4 – Robust

Content must be robust enough that it can be interpreted by a wide variety of user agents, including assistive technologies. Help users avoid and correct mistakes.

This success criterion is primarily for Web authors who develop or script their own user interface components. For example, standard HTML controls already meet this success criterion when used according to specification.

FTC Docs To Do

- We probably need to change the fake buttons used in the primary grid pages to real buttons.

Image and Figure Details

This section has detailed information about images and figures and more advice about how to handle them in FTC Docs.

Contents

- *Complex Image Example*

Decorative Images

A wavy line image that is used to separate blocks of content is a decorative image. A photo of persons shaking hands on a page about negotiating contracts might be decorative if it was placed to look pretty. If the image is not directly related to the content and is only there for visual appeal then it is a decorative image.

Note: FTC Docs does not use decorative images. Do not add an image because it looks nice. This is not a marketing web site or a visual design website.

Decorative images are also an accessibility problem. The screen reader has to process them. If the image has alt text it may not be related to the content so it may just cause confusion.

If you really need a decorative image, use the `.. image` directive and include the **:alt:** option with blank text. Blank alt text will cause screen readers to ignore the image, which is appropriate when the image is just there for visual presentation.

Example

```
.. image:: images/handshake.jpg
   :alt:
```

Image Directive

Use the `.. image` directive to include an image that does **not** need a caption. Determine if the image needs a caption. Photo credits need a caption. Complex images probably need a caption. Anything that requires an editorial or illustrative description to tie the image to the content of the page requires a caption.

If no caption is required then the image directive can be used. Add the `:alt:` option and provide text that describes the image. This is all you need for basic images.

```
.. image:: images/butterfly.jpg
   :alt: A blue butterfly sitting on an orange flower.
```

The alt text should have a functional description. A functional description explains exactly what is in an image. These descriptions should be thorough but brief, one or two sentences. If a long description is required, use the *figure directive* and provide a long description.

See *Alt Text* for more information about alt text.

Warning: Sphinx will set alt text set to the path and file name of the image if you don't specify the `:alt:` option. The HTML will look like `<img alt="../_images/GoodStuff.png" src="../_images/GoodStuff.png">`. A screen reader will read out loud the alt text **../_images/GoodStuff.png**.

This is not good accessibility. If you are editting an existing page that has an image or figure directive with no `:alt:` option, please take a moment to add the `:alt:` option with a functional description of the image.

The image directive has many options, but we don't recommend most of them for FTC Docs. This is new guidance, many existing pages specify **width** and **align**. It maybe worth removing those if you are changing the content on that page anyway.

Image Options

The options supported by the `image` directive are:

alt
: [*text*] Alternate text: a short description of the image, displayed by applications that cannot display images, or spoken by applications for visually impaired users

height
: [*length*] The desired height of the image. Used to reserve space or scale the image vertically. When the "scale" option is also specified, they are combined. For example, a height of 200px and a scale of 50 is equivalent to a height of 100px with no scale.

scale
: [*integer percentage (the "%" symbol is optional)*] The uniform scaling factor of the image. The default is "100 %", i.e. no scaling.

width
: [*length or percentage of the current line width*] The width of the image. Used to reserve space or scale the image horizontally. As with "height" above, when the "scale" option is also specified, they are combined. It is often preferable to use *width* over *height* or *scale*.

align
: ["top", "middle", "bottom", "left", "center", or "right"] The alignment of the image, equivalent to the HTML <img> tag's deprecated "align" attribute or the corresponding "vertical-align" and "text-align" CSS properties. The values "top", "middle", and "bottom" control an image's vertical alignment (relative to the text baseline); they are only useful for inline images (substitutions). The values "left", "center", and "right" control an image's horizontal alignment, allowing the image to float and have the text flow around it. The specific behavior depends upon the browser or rendering software used.

target
: [*text (URI or reference name)*] Makes the image into a hyperlink reference ("clickable"). The option argument may be a URI (relative or absolute), or a reference name with underscore suffix (e.g. `a name`_).

The new guidance related to images comes from improving website accessibility. We recommending avoiding the following options.

align
: The accessibility problem comes because the image can float to the new position with text re-flowing around it. This can float the image out of context with its surrounding text. That is a big accessibility issue as images should relate to the text near them. Images can float to the next page in the PDF version, sometimes the image is there all alone on an otherwise blank page.

width, height, scale
: Width is usually used to force the image to not fill the width of the page which usually looks OK in HTML and PDF. However, when viewing in a mobile browser the image can be too small to see easily. For example, a width of 50% will look fine when viewed on a big screen, but in portrait mode in a mobile browser the image will be half the width of the screen. However, on mobile you can usually just use two fingers to zoom the image (as long as you don't have a physical disability with your fingers).

  The bigger accessibility problem with these options is that that Sphinx will insert a link to the image. The idea is that you can click the link to see the full size image. This is an accessibility issue as the link itself has no title. It does not read well in a screen reader. If a visually impaired person followed the link they end up on a page with no text content and no alt text either.

  Sighted persons who want to see the full size image have the option to right click the image and open it in a new tab or window.

  The AprilTag test images have both height and width specified as 5 inches which looks OK on the desktop or in a PDF, but ends up with a squished aspect ratio if viewed in a mobile browser. We'd be better off not specifying set sizes for the HTML and the PDF version of FTC docs. Then provide a separate PDF download that they can print to get accurately sized AprilTags.

If you want to keep the **width** option (perhaps the image size is too big for the page), then for accessibility we recommend you add the **class** option with **no-scaled-link** e.g. `:class: no-scaled-link` This tells Sphinx to not create the link, but the images will have the width you want. Though a better option might be to change the resolution of the image if relevant detail can be preserved.

External Images

It is possible to include images that are external to FTC Docs, but we don't recommend that. There is no way to know if the image will still be there in the future. There is also the issue that external images may be copyrighted and we might not have permission to use.

Including an external image using a web address:

```
.. image:: https://m.media-amazon.com/images/I/51-2PZby7KL.jpg
   :alt: Logitech gamepad
```

Alt Text

Images must have text alternatives that describe the information or function represented by them. This ensures that images can be used by people with various disabilities.

Why is this important?

Images and graphics make content more pleasant and easier to understand for many people, and in particular for those with cognitive and learning disabilities. They serve as cues that are used by people with visual impairments, including people with low vision, to orient themselves in the content.

However, images are used extensively on websites and can create major barriers when they are not accessible. Accessible images are beneficial in many situations, such as:

- People using screen readers: The text alternative can be read aloud or rendered as Braille
- People using speech input software: Users can put the focus onto a button or linked image with a single voice command
- People browsing speech-enabled websites: The text alternative can be read aloud
- Mobile web users: Images can be turned off, especially for data-roaming
- Search engine optimization: Images become indexable by search engines

Alt Text Guidelines

The following guidelines are from: https://www.nngroup.com/articles/write-alt-text/

- Keep it short. Alt text shouldn't be much longer than around 150 characters. Users can't pause and resume the screen reader in the middle of alt text without going back to the beginning. People also can't hold very much information in their working memory. Users will skip alt text if it doesn't immediately seem like it will help them with their task.
- Do not include words like 'image' or 'photo' at the beginning. Screen readers already identify images as images when they encounter them because they are contained within the <img> HTML tag. Identifying an image as a certain type (e.g., infographic, chart, illustration) is appropriate if it will help the user understand the other alt text.
- End alt text with a period, even if it isn't a full sentence. The period ensures that the screen reader pauses after reading the alt text.
- Frontload alt text with the most important words, to help users make a quick and informed decision about whether it's worth listening to the rest of the alt text before moving on.

- Always include an alt attribute (alt=""), even if it will be empty. Otherwise, screen readers might announce the image file name.
- Avoid technical jargon and abbreviations unless users are certain to understand them.
- Never reuse alt text for the same image without reanalyzing the context in which the image is placed.
- Mention identity only if it's relevant. If the race, ethnicity, gender, religion, or cultural identifiers of the people pictured aren't part of the reason the image was included, don't mention them.

The following examples are taken from: https://www.w3.org/WAI/tutorials/images/

Example: Images used to supplement other information

The following image shows a dog wearing a bell. It supplements the adjacent text that explains the purpose of this bell. A short text alternative is sufficient to describe the information that is displayed visually but is not explained in the text; in this case, the text alternative is "Dog with a bell attached to its collar.".

	Off-duty guide dogs often wear a bell. Its ring helps the blind owner keep track of the dog's location

The alt text is added to the HTML img tag.

```
<img alt="Dog with a bell attached to its collar." src="https://www.w3.org/WAI/content-images/tutorials/images/dog.jpg">
```

Note: If the text included an explanation of how the dog wears a bell, the image might be considered redundant and therefore decorative. As this isn't mentioned in the text, the image is deemed to be informative.

Example: Images conveying succinct information

This simple diagram illustrates a counterclockwise direction for unscrewing a bottle top or cap. The information can be described in a short sentence, so the text alternative "Push the cap down and turn it counterclockwise (from right to left)" is given in the alt attribute.

The alt text is added to the HTML img tag.

```
<img alt="Push the cap down and turn it counterclockwise (from right to left)" src="https://www.w3.
↪org/WAI/content-images/tutorials/images/counter-clockwise.jpg">
```

Note:

1. An alternative technique would be to provide the instructions within the main content rather than as a text alternative to the image. This technique makes all information available in text for everyone while providing an illustration for people who prefer to view the information visually.
2. If more information than that of the diagram is intended to be conveyed by the image, it may be better to follow one of the approaches described in *Complex Image Example*. For example, if the fact that this diagram appears on a bottle or if the shape and size of the bottle were relevant pieces of information, use a more detailed alternative text.

Alt Text and Captions

The previous examples did not require captions. If you also require a caption use the RST figure directive with a caption and alt text. Ensure the caption does not repeat the alt text. That's because a screen reader will read both.

While both the alt attribute and the figcaption element provide a way to describe images, the way we write for them is different.

- alt descriptions should be functional;
- figcaption descriptions should be editorial or illustrative.

If the caption is just a functional description of the image, maybe you don't need a caption and can use the image directive instead.

See more examples of alt text and captions on this Thoughbot blog post.

Figure Directive

Use the `.. figure` directive when the image requires a visible text caption or a long description.

Photo credits are an example of when you need a caption. You should also use a caption when you need editorial or illustrative text to highlight something about the image to the reader or to connect the image to the surrounding text content.

If a caption is not required, just use the *Image Directive*.

Please create alt text for screen readers even though there is a caption. The alt text and caption should be different because a screen reader will read both. One way to think about this is the alt text should be functional and the caption editorial or illustrative. In the following example, the alt text describes the image, and the caption serves to connect the image with a travel article about Machu Picchu.

```
.. figure:: images/martha.jpg
   :alt: A closeup of a llama's face looking off to the side on a mountain.

   Martha is one of the many domesticated llamas that roam freely around the grounds of Machu␣
↪Picchu.
```

Note that the `:alt:` line and caption are both indented 3 spaces after the directive. A blank line is required between the `:alt:` and the caption.

We don't want the `:alt:` line to be blank for a figure. A screen reader will have probably spoken that there is a figure, without alt text the screen reader will skip over announcing the image and read the caption leaving the user wondering what the caption is referring to.

The `.. figure` directive supports all options of the `.. image` directive. These options (except align) are passed on to the contained image.

- **`:align:` "left", "center", or "right".**
  The horizontal alignment of the figure, allowing the image to float and have the text flow around it. The specific behavior depends upon the browser or rendering software used.

  Avoid using **align**. In PDF it tends to float the figure to another area of the page, sometimes to the next page where the image is no longer in context.

There is an optional legend that can be included after the caption. This might be useful for charts and maps and other complex imagery. The legend paragraph is a good place for a long description of the image to go.

Legends looks like:

```
.. figure:: map.png
   :alt: map to buried treasure

   This is the caption of the figure (a simple paragraph).

   The legend consists of all elements after the caption.  In this
   case, the legend consists of this paragraph and the following
   table:

   +-----------------------+-----------------------+
   | Symbol                | Meaning               |
   +=======================+=======================+
   | .. image:: tent.png   | Campground            |
   +-----------------------+-----------------------+
   | .. image:: waves.png  | Lake                  |
   +-----------------------+-----------------------+
   | .. image:: peak.png   | Mountain              |
   +-----------------------+-----------------------+
```

There must be blank lines before the caption paragraph and before the legend. To specify a legend without a caption, use an empty comment ("..") in place of the caption.

A table might be useful for charts or complex images that might need descriptions of various parts of the image.

> **Caution:** We do not recommend using the ascii art form of a table as shown above, use a list style table instead.

Complex Image Example

FTC Docs has some images that are very complex. Usually there is some surrounding text that relates to the image. However, we usually don't actually describe the image as we assume the reader is not impaired visually.

If you are editing an existing page, you may need to adjust what text surrounds the image, and what text describes the image. If the surrounding text describes part of the image, it should probably be moved into the long description of the image.

The FTC Docs guidance is to use the legend of a `.. figure` directive when a long description is required. This paragraph should be placed after after the caption, leaving a single blank line in between. Instead of a paragraph, you can include a table or list if that would better describe the image.

The following example is how we might describe a complex diagram. We use a `.. figure` directive with alt text, caption and long description. This diagram is located on the Control System Introduction page.

```
.. figure:: images/REVExpansionHubLayout.jpg
   :alt: Rev expansion hub with various devices connected.

```

(continues on next page)

(continued from previous page)

```
   Expansion Hub and Phone

   The Expansion Hub has the following devices connected.

   - a Robot Controller phone via a USB connection;
   - A 12 volt battery with on/off switch;
   - A three wire servo connects to one of six servo ports;
   - An analog sensor connects to one of two analog sensor ports;
   - An I2C sensor connects to one of four I2C ports;
   - Two motors are connected to the Expansion hub. Each motor has a power connection and an␣
↪encoder connection. There are four motor ports on the Expansion Hub.
```

The alt text is a summary of the functional description of the image (which follows the caption). The caption indicates that this is an example of an Expansion Hub and phone and relates to the prior paragraphs on the Control System Introduction page which talk about possible configurations of the Expansion Hub. In this case the long description is basically a listing of the devices connected to the Expansion Hub.

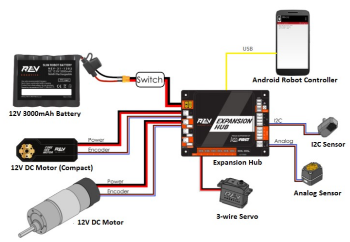


Fig. 3: Expansion Hub and Phone

The Expansion Hub has the following devices connected.

- a Robot Controller phone via a USB connection;
- A 12 volt battery with on/off switch;
- A three wire servo connects to one of six servo ports;
- An analog sensor connects to one of two analog sensor ports;
- An I2C sensor connects to one of four I2C ports;
- Two motors are connected to the Expansion hub. Each motor has a power connection and an encoder connection. There are four motor ports on the Expansion Hub.

Using a Figure with Caption and Legend is good for accessibility because Sphinx will generate a HTML Figure tag and Figcaption tag. This clearly associates the text with the image for screen readers.

This HTML is from the square field image of the Field Coordinate System page.

```
<figure class="align-default" id="id2">
  <img alt="A square field with X, Y and Z axes shown" src="../../_images/image3.jpg">
  <figcaption>
    <p><span class="caption-text">The Cascade Effect game field</span><a class="headerlink" href="
↪#id2" title="Permalink to this image"></a></p>
    <div class="legend">
      <p>In a square field configuration the two Alliances face each other across the field.
      The field is oriented such that the Red Wall is on the right as seen
      from the audience, and the blue wall will be on the left.
      The Y axis points across the field from the Red Wall to the blue wall.
      The X axis points away from the audience to the rear of the field.</p>
    </div>
  </figcaption>
</figure>
```

A screen reader encountering the HTML above will normally announce that there is a *figure*. It would then speak the alt text and indicate there was an *image*. It would continue by reading the caption followed by the long description. Finally the screen reader would announce *end figure*.

Using the `.. figure` directive for complex images nicely surrounds the image with a complete text description that is useful for all users as we clearly indicate what the image is about and what is important.

Note: If you ever find that you need a long description, but the image really does not need a visible text caption, you can enter a blank comment as the caption. This looks like two periods `..` placed where the caption would be and at the same indent as the long description.

```
.. figure:: images/image3.jpg
   :alt: A square field with X, Y and Z axes shown.

   ..

   In a square field configuration ...
```

30.3 FTC Docs Workflows

Note: Please note that this flowchart is meant only for reference for *maintainers* of the FTC Docs repository. For those looking only to contribute to the FTC Docs documents please refer to the *Contributing to FTC Docs* document.

30.3.1 Overview

The following diagram shows the various GitHub repositories and the actions and flow between them and the build artifacts. Pull Requests to the FTC Docs repository result in GitHub actions that build HTML pages and PDF files. The HTML pages ultimately end up on the ftc-docs.firstinspires.org website and the PDF files in AWS S3 file storage.

In a web browser this diagram can be zoomed and panned by using a mouse. Use the scroll wheel to zoom in and out. Right click and hold then drag to pan. The diagram is not keyboard accessible. A screen reader will read the various nodes and actions in the diagram and starts in the Translation section of the diagram.

30.4 Tutorials

These are a couple tutorials that will walk you through the process of creating and editing in FTC Docs.

30.4.1 Overview

Below is an overview of the process of contributing to FTC Docs.

Warning: Using **Codespace** and working **Locally** are two different ways to contribute to FTC Docs. The steps for each are similar, but there are key differences in setup. After reading step 2, you should choose whether you want to use Codespaces or work locally. For the purposes of this guide, Local and Codespaces use will be mutually exclusive.

Key

One Time Only	Repeat	Codespaces	Local	Information
Step only needs to be done once	Step must be repeated for each set of changes	Step specific to Codespaces users	Step specific to Local users	General information that provides context

2. *Intro to Codesapces* Information
   - This will provide an overview of Codespaces and how to use them.
3. *Getting to know the GitHub Repository* Information
   - This will provide an overview of the FTC Docs repository and how it is organized. This is important to understand before you start working on a contribution.
4. *Fork the repository* One Time Only Codespaces Local
   - This will create a copy of the FTC Docs repository in your GitHub account.
5. *Update the repository* Repeat Codespaces Local
   - This will update your fork with the latest changes from the FTC Docs repository. This is important to do before you start working on a new contribution.
6. *Set up your environment* One Time Only Local
   - This will set up your local environment to work on FTC Docs. This step can be skipped for Codespaces users.
7. *Create a new branch* Repeat Codespaces Local
   - This will create a new branch for your change. You should create a new branch for each change you work on.
8. *Create a new codespace* Repeat Codespaces
   - This will create a new Codespace for your change. You should create a new Codespace for each change/branch you work on.
9. *Switch to your branch* Repeat Local
   - This will switch to the branch you created in step 7. You should switch to the branch you created for each change you work on.
10. *VS Code Tasks* Information
    - This will provide an overview of the tasks for FTC Docs available in VS Code. This is important to understand before you start working on a contribution.
11. *Make your changes* Repeat Codespaces Local
12. *Setup Git Credentials* One Time Only Local
    - This will set up your Git credentials so you can push your changes.
13. *Submit your changes* Repeat Codespaces Local
    - Commit your changes and submit a pull request to the FTC Docs repository.

30.4.2 Codespaces

Information

What is Codespaces?

Codespaces is a cloud-hosted development environment that you can access from anywhere. It comes with a pre-configured Visual Studio Code environment and a containerized runtime that matches your development environment. You can use Codespaces to develop in a browser, or by using the Visual Studio Code Remote - Codespaces extension in your local Visual Studio Code instance.

Why use Codespaces?

Codespaces is a great way to develop in a consistent environment, no matter where you are. It also allows you to develop on a device that might not have the resources to run your development environment locally. It also allows you to sidestep the need to install and configure your development environment on a new machine.

Who should use Codespaces?

Codespaces is our recommended development environment for all developers. It is especially useful for developers who need to work on multiple machines, or who are not comfortable with setting up their development environment and the troubleshooting that comes with it.

Downside of Codespaces

Codespaces is a paid service, and you will be billed for the resources you use. However, GitHub currently offers GitHub Free tier users **120 Core hours** and GitHub Pro tier users **180 Core hours** per month for free. To learn more about Codespaces pricing, visit GitHub Documentation.

30.4.3 Getting to know the GitHub Repo

Information

Note: It is assumed that you have already created a GitHub account.

GitHub is a website where people and organizations can store projects in what's known as a "repository". Some repositories can be public, meaning anyone can see the content within it, and some are private. The ftcdocs repository you're working on is a public repository, which means that anyone can access it.

Understanding the ftcdocs Repo

The GitHub `ftcdocs` repository can be found on GitHub at the following address:

- https://github.com/FIRST-Tech-Challenge/ftcdocs/

The GitHub repository is where all of the source files for this website are kept. Figure 1 shows what the main code page for the `ftcdocs` repository looks like.

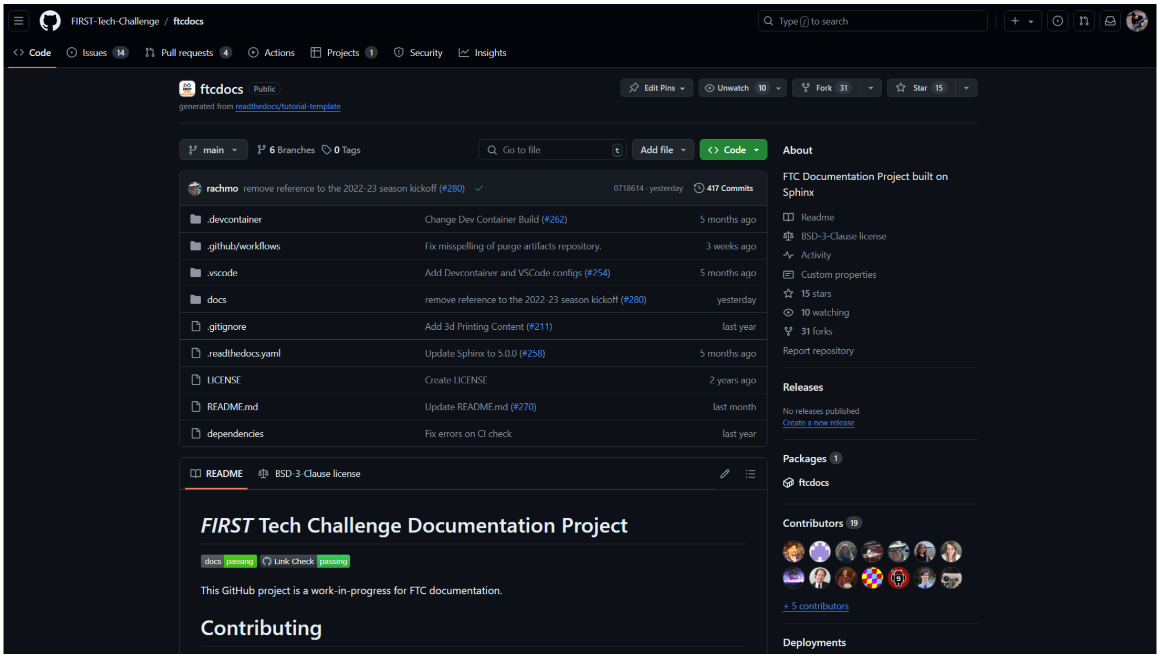


Fig. 4: Figure 1: ftcdocs GitHub Repository

This main **code page** is where you'll do most of your work. It's called a **code page** because by default the `< > Code` tab of the repository is selected, and this is the page that we're currently viewing. For software projects, the **code page** is where code is stored; for us, this is where our *content* is stored. There are several different tabs, but we only really care about the first four:

1. `< > Code` - The **Code page** shows us the file structure of our repository and also allows us to view and edit files.
2. `Issues` - The **Issues page** shows us "issues" that any user can submit. These issues are generally feature request (like "Please add emojis to the document workflow") or bug reports (like "When I use dropdowns, my document errors out."). Issues are not meant to be discussions, but very specific tasks that need to be addressed.
3. `Pull Requests` - The **Pull Requests page** shows us "Pull Requests"; for this project, these will be requests to merge changes into the main branch. Don't worry about this page just now, we'll cover **Pull Requests** in more detail later.
4. `Discussions` - The **Discussions page** is where users can visit and ask questions or get help on topics. This is meant to be an open discussion area for the repository. This area is similar to a forum, but specifically for `ftcdocs`.

The `< > Code` tab will be the tab that we will spend most our time in, as this is where we manage *branches*, view and edit files, and perform most of our basic functions.

30.4.4 Forking FTC Docs

One Time Only Codespaces Local

Note: This following assumes you have already created a GitHub account. If you have not, please create one

Forking a repository is a simple process. It creates a copy of the repository in your account, and you can make changes to it without affecting the original repository. You can also submit a pull request to the original repository to propose changes to the original repository. It also allows you to easily keep your forked repository up to date with the original repository.

Steps

1. Go to the repository you want to fork. In this case, it is the FTC Docs repository.

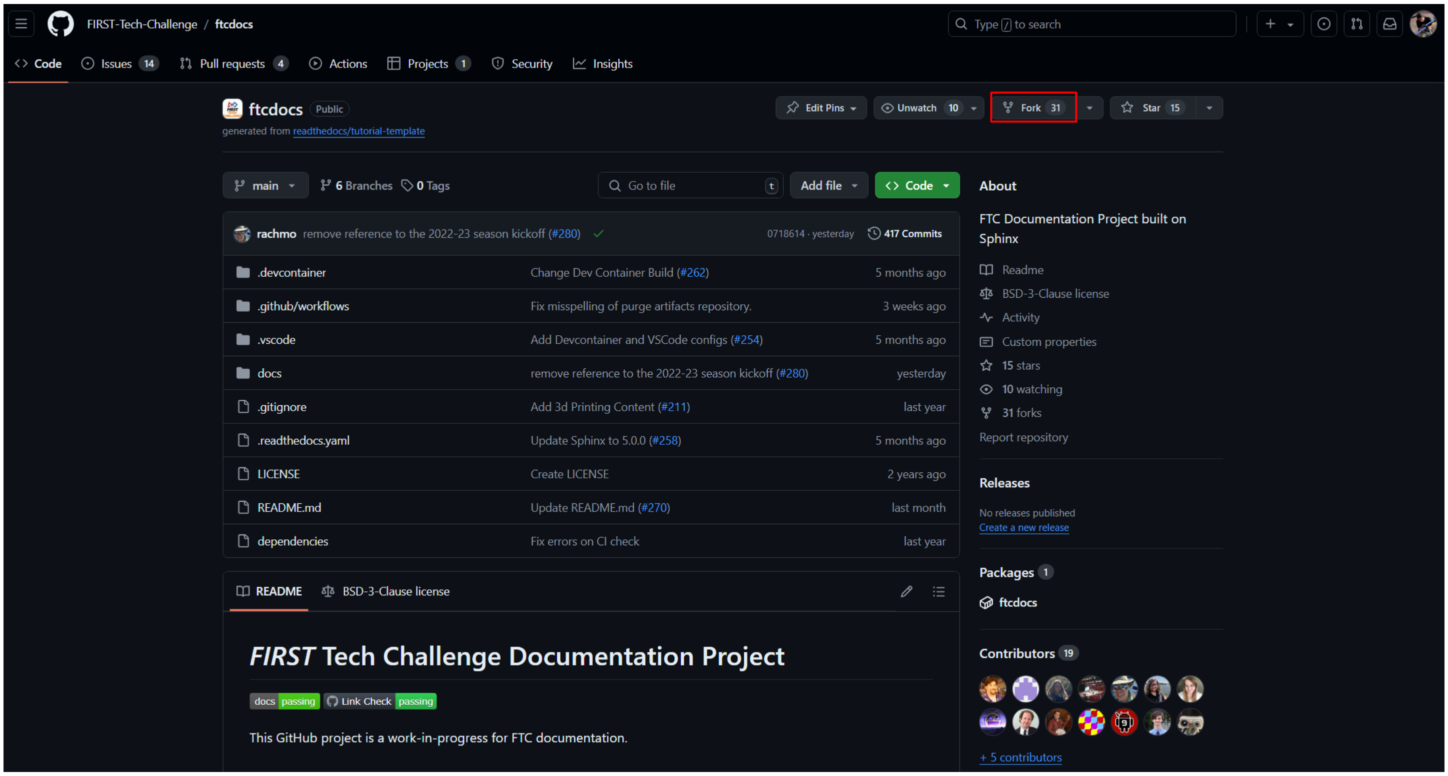


Fig. 5: The Official FTC Docs GitHub Repository

2. Follow the steps shown below to fork the repository.

For more information on forking a repository, visit the GitHub Documentation.

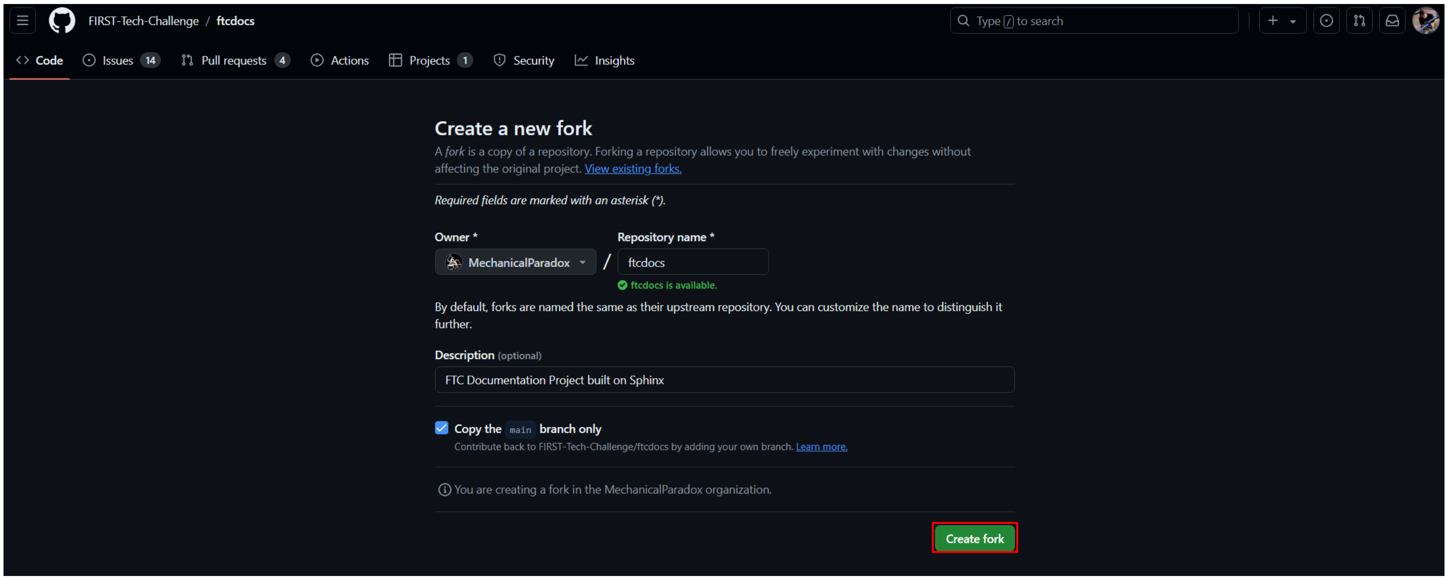


Fig. 6: Click the “Create Fork” button to create a fork of the repository.

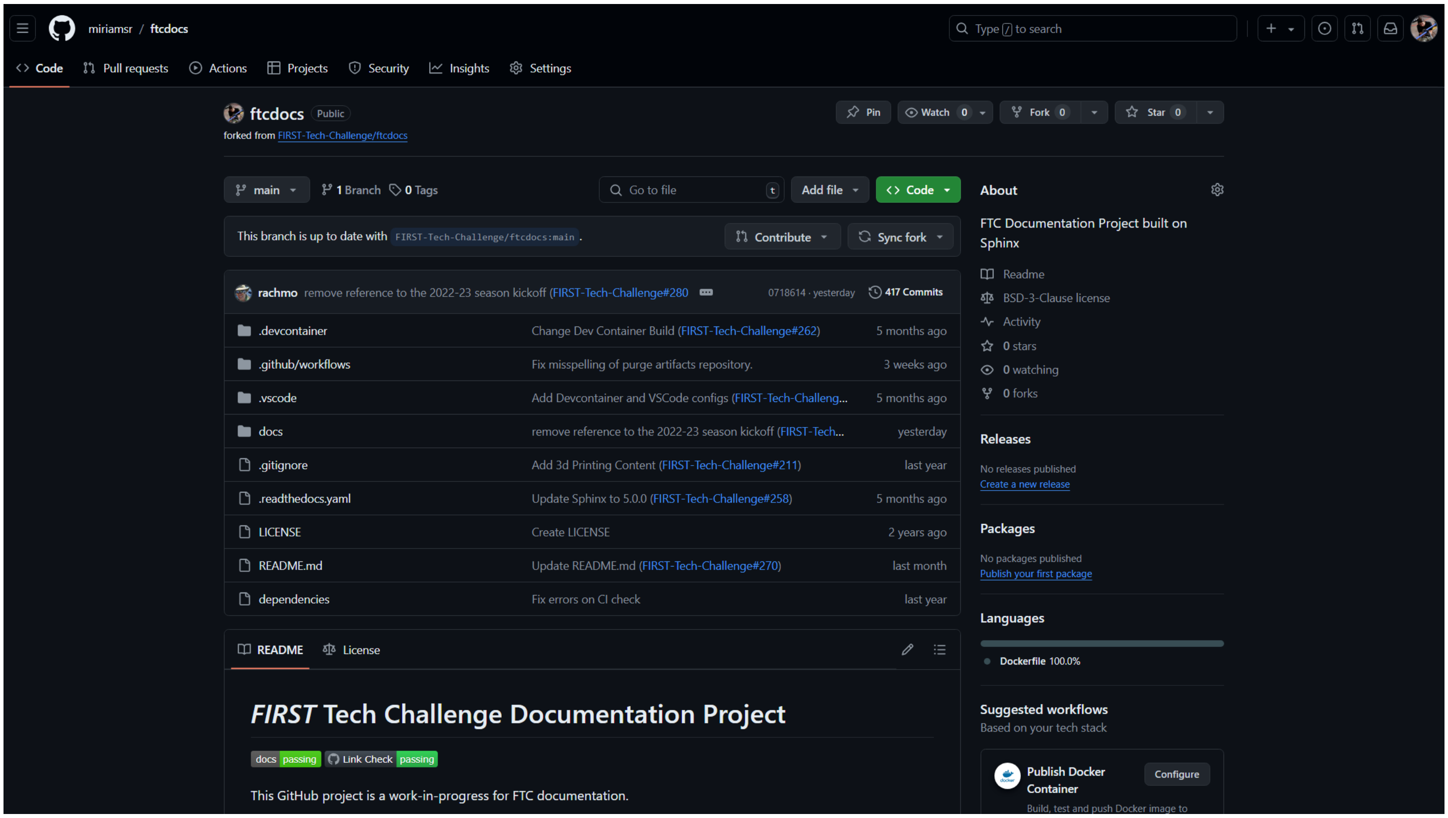


Fig. 7: Forked Repository

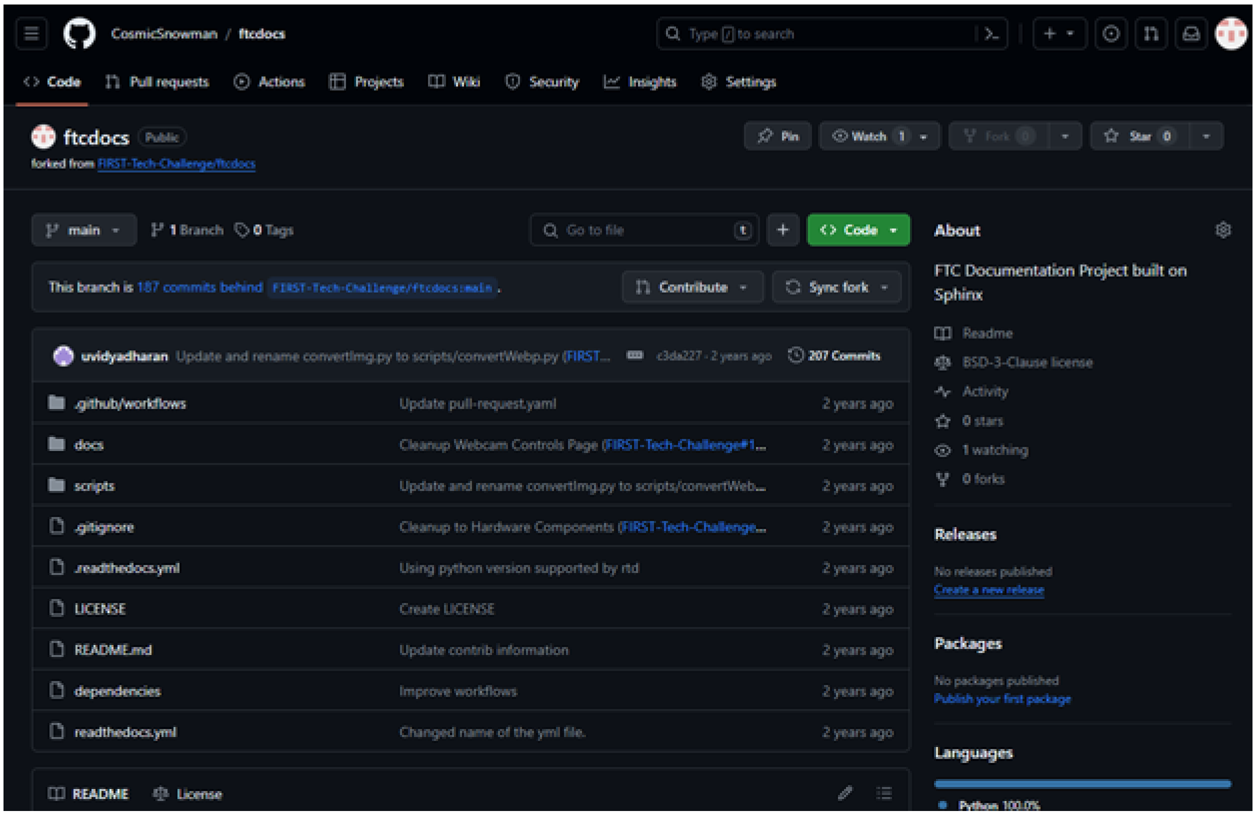
30.4.5 Updating Fork

Repeat Codespaces Local

Your fork, while linked to the original repository, does not serve as a mirror. When changes are made "upstream" (in the original repository), you will need to update your fork to reflect these changes. You will want to do this **before** you start working on a new feature or bug fix.

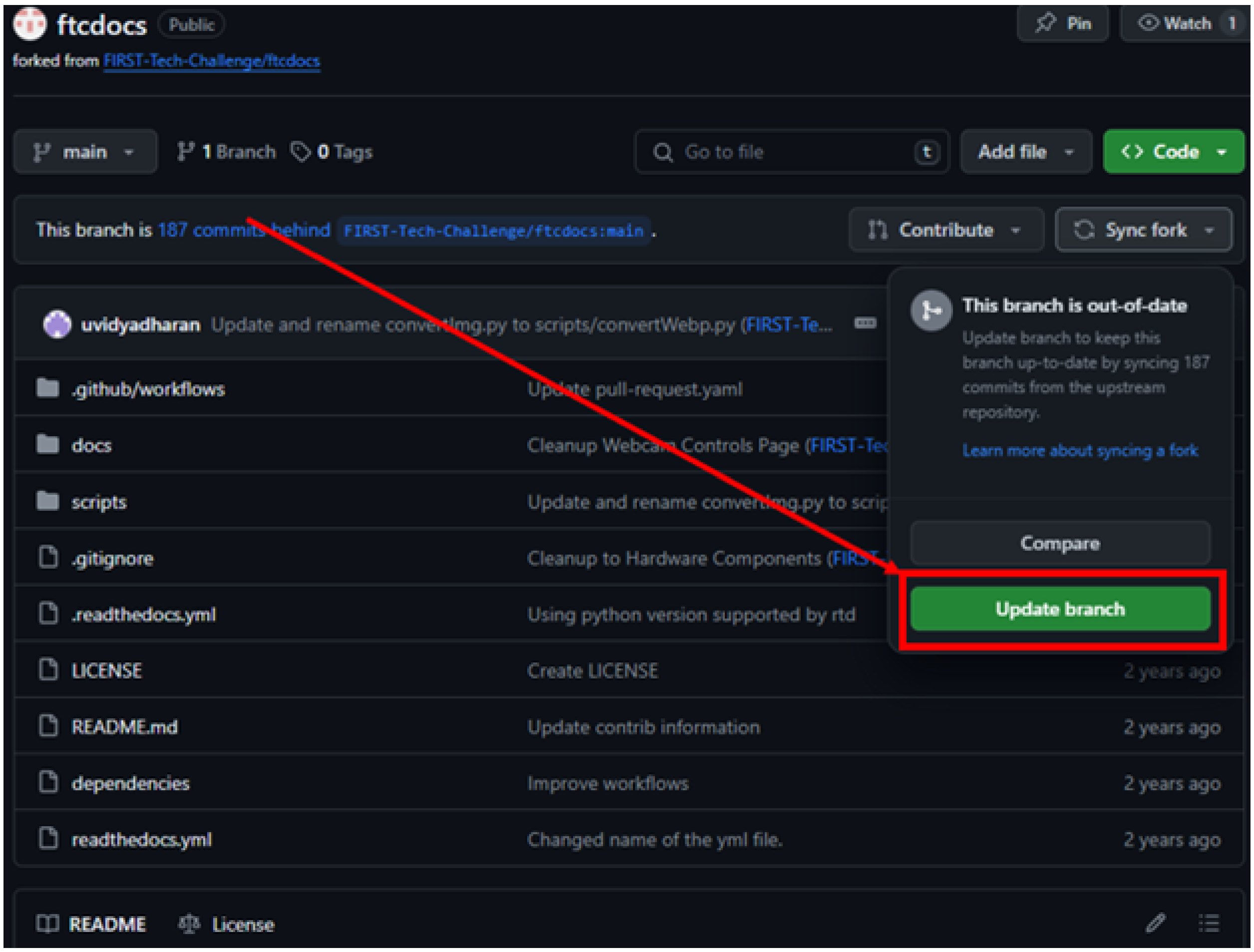
Steps

1. Navigate to your fork on GitHub



2. Follow the steps outlined below

More information on the process can be found on GitHub Documentation.


ftcdocs
Public
forked from FIRST-Tech-Challenge/ftcdocs
Pin
Watch 1
main
1 Branch
0 Tags
Go to file
Add file
Code
This branch is 187 commits behind FIRST-Tech-Challenge/ftcdocs:main .
Contribute
Sync fork
uvidyadharan Update and rename convertImg.py to scripts/convertWebp.py (FIRST-Te...
This branch is out-of-date
Update branch to keep this branch up-to-date by syncing 187 commits from the upstream repository.
Learn more about syncing a fork
Compare
Update branch
.github/workflows
Update pull-request.yaml
docs
Cleanup Webcam Controls Page (FIRST-Te
scripts
Update and rename convertImg.py to scrip
.gitignore
Cleanup to Hardware Components (FIRST-
.readthedocs.yml
Using python version supported by rtd
LICENSE
Create LICENSE
2 years ago
README.md
Update contrib information
2 years ago
dependencies
Improve workflows
2 years ago
readthedocs.yml
Changed name of the yml file.
2 years ago
README
License

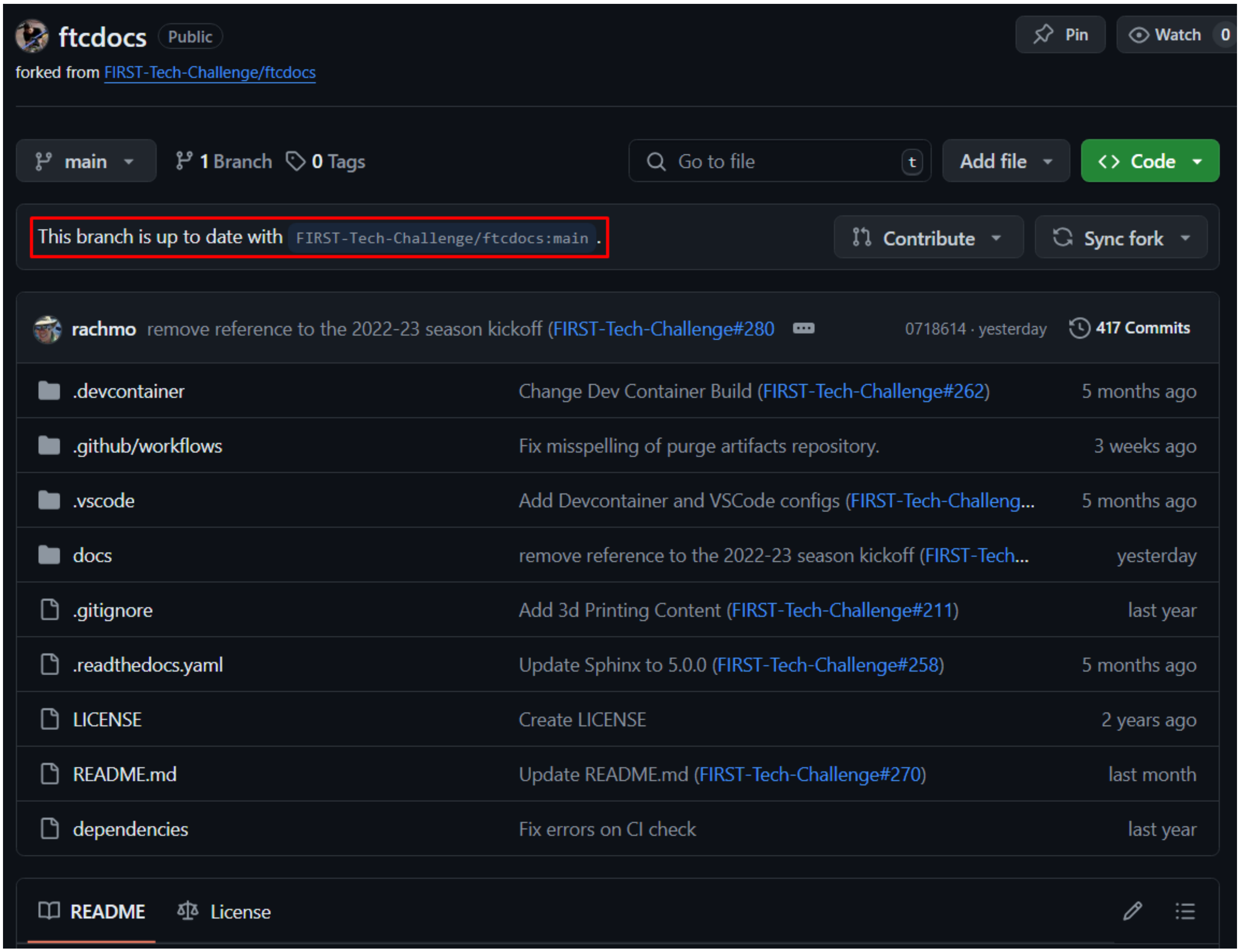


Fig. 8: Note the new message indicating that your fork is up to date with the original repository.

30.4.6 Setting Up Your Development Environment

One Time Only Local

> **Warning:** Only complete these steps if you have chosen to develop the site locally. If you are using **GitHub Codespaces** you should skip this section.

FTC Docs has a few dependencies that you'll need to install before you can start developing. Full build features are only available on Linux. This will only effect those who are looking to build PDFs locally.

Remember, this step should **ony be done for Local Development**. If you are using **GitHub Codespaces** you should skip this step. Also note that these steps should only be done **once**.

Steps

> **Warning:** Make sure you have forked the repository before starting this process. If you have not, please do so now.

> **Warning:** In some cases, you may need to restart your computer or terminal instance between or after installing these dependencies for the changes to take effect.

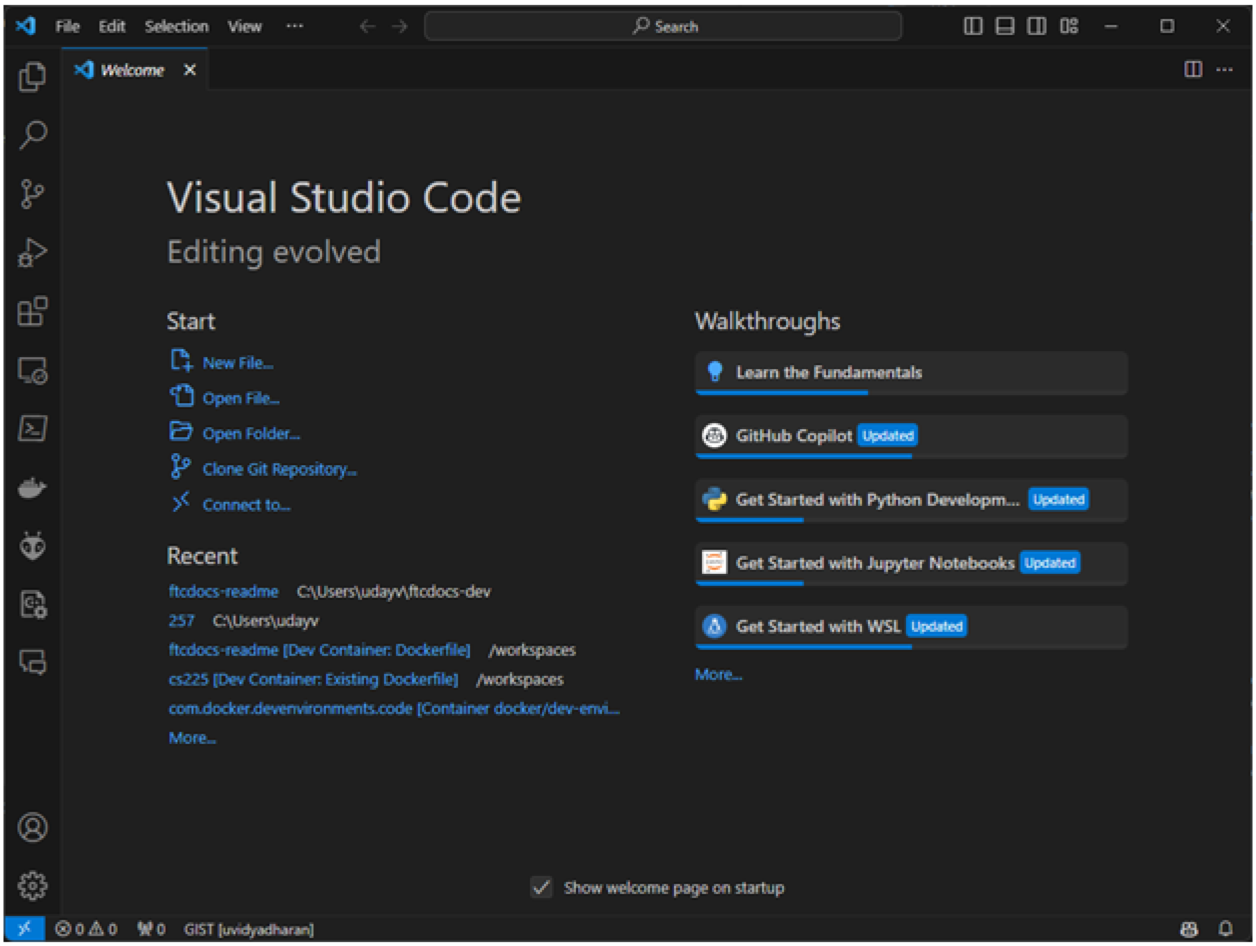
Windows

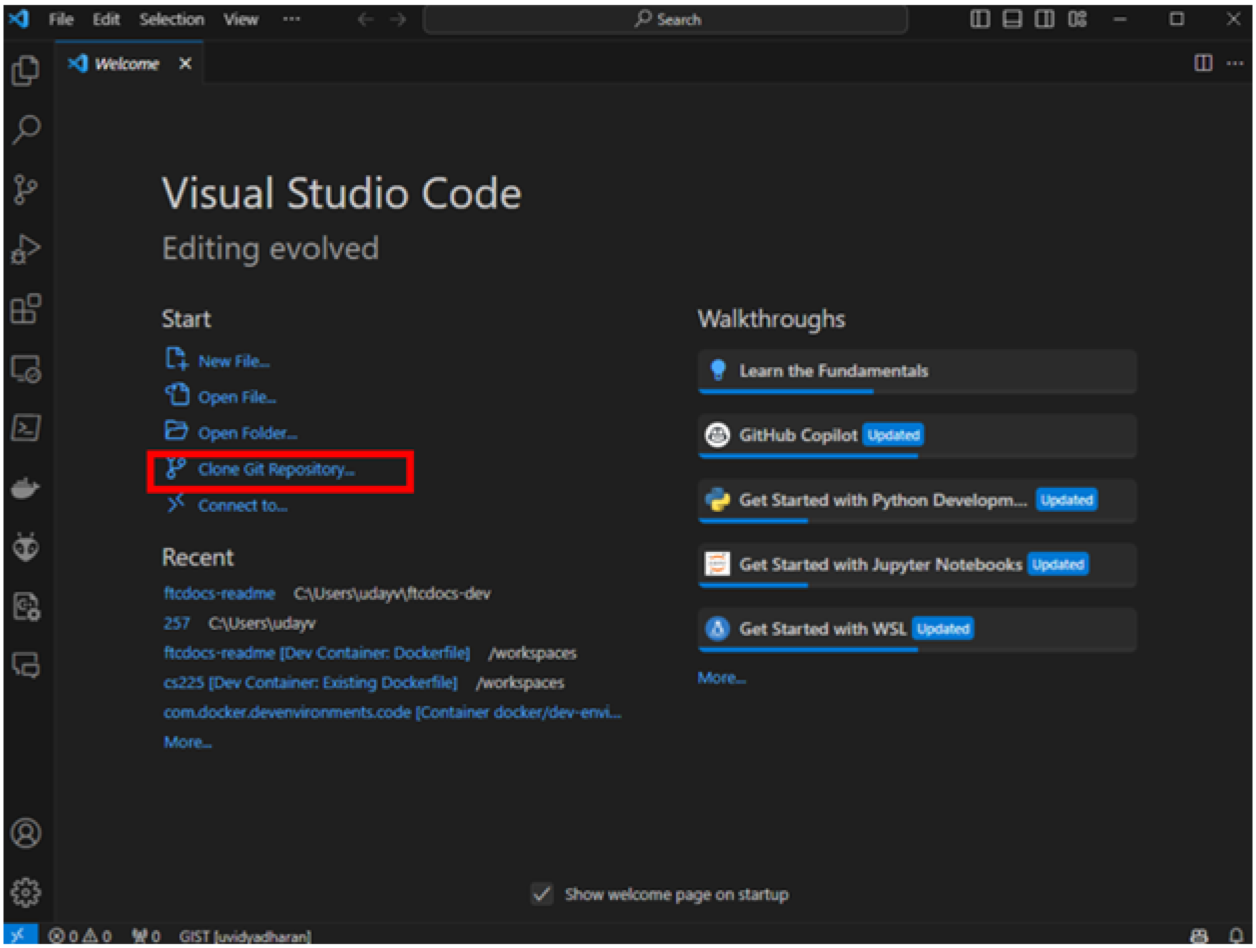
1. Install Chocolatey.
2. Install Python 3.9 or later from the Python website. **Make sure to check the box that says "Add Python to PATH".**
3. Install Pip. `python -m ensurepip`
4. Install Git. `choco install git`
5. Install Make. `choco install make`
6. Install the lastest version of VS Code. `choco install vscode`

Linux/Mac

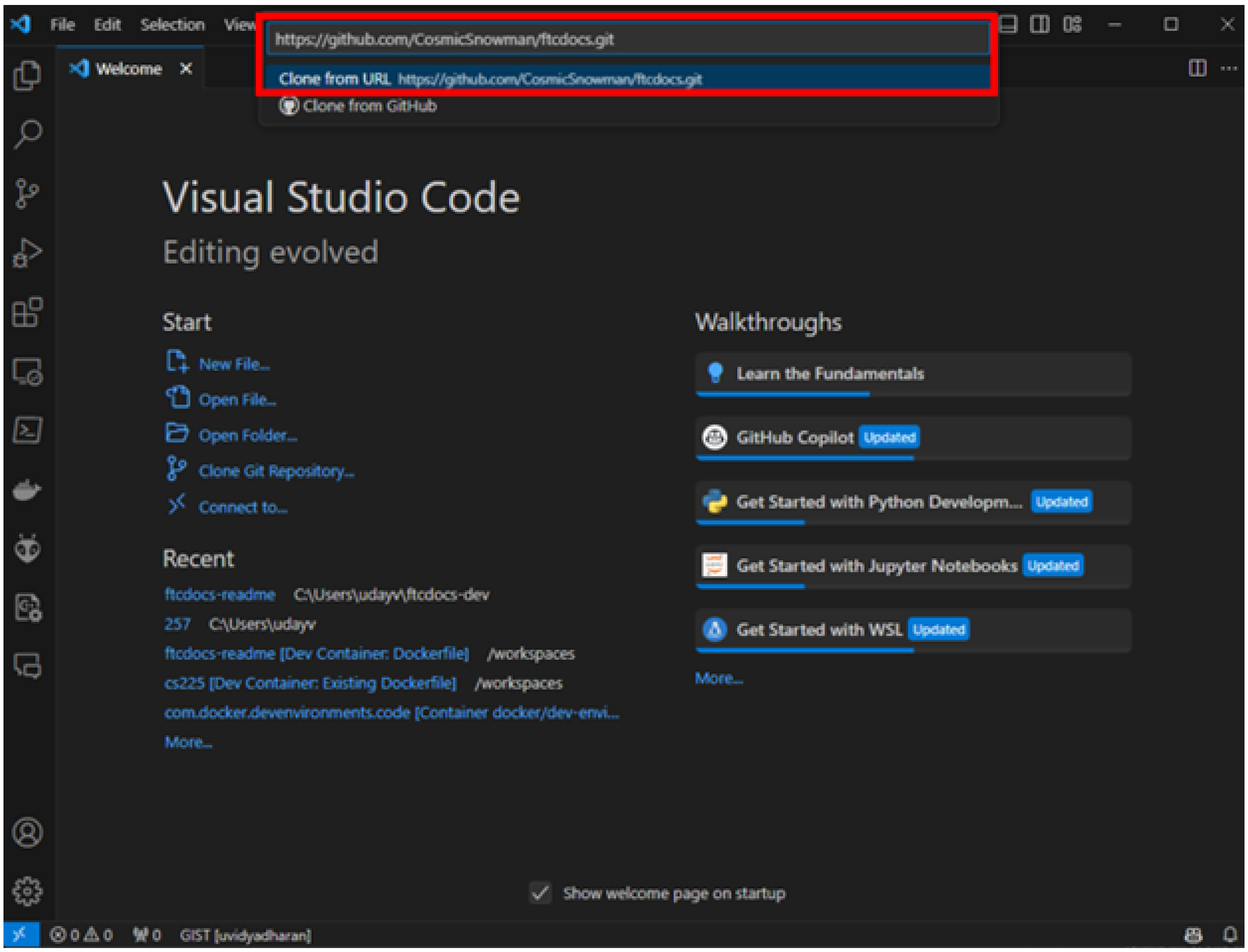
1. Install Python 3.9 or later. You can download it from the Python website.
2. Install the latest version of Pip.
3. Install Git from the Git website.
4. Install Make .
5. Install the lastest version of VS Code.

1. Open VS Code
2. Under Start in the welcome screen, click on "Clone Repository"
3. Enter the URL of your forked repository and click "Clone Repository". This will take the format of *https://github.com/<NAME>/ftcdocs.git* replacing <NAME> with your GitHub username. If you changed the name of your fork to something other than `ftcdocs`, replace `ftcdocs` with the name of your fork.
4. Chose a location on your computer to save the repository and click "Select Repository Destination".


File Edit Selection View
Search
Welcome
Visual Studio Code
Editing evolved
Start
New File...
Open File...
Open Folder...
Clone Git Repository...
Connect to...
Recent
ftcdocs-readme C:\Users\udayv\ftcdocs-dev
257 C:\Users\udayv
ftcdocs-readme [Dev Container: Dockerfile] /workspaces
cs225 [Dev Container: Existing Dockerfile] /workspaces
com.docker.devenvironments.code [Container docker/dev-envi...
More...
Walkthroughs
Learn the Fundamentals
GitHub Copilot Updated
Get Started with Python Developm... Updated
Get Started with Jupyter Notebooks Updated
Get Started with WSL Updated
More...
Show welcome page on startup
GIST [uvidyadharan]



File Edit Selection View
Search
Welcome
Visual Studio Code
Editing evolved
Start
New File...
Open File...
Open Folder...
Clone Git Repository...
Connect to...
Recent
ftcdocs-readme C:\Users\udayv\ftcdocs-dev
257 C:\Users\udayv
ftcdocs-readme [Dev Container: Dockerfile] /workspaces
cs225 [Dev Container: Existing Dockerfile] /workspaces
com.docker.devenvironments.code [Container docker/dev-envi...
More...
Walkthroughs
Learn the Fundamentals
GitHub Copilot Updated
Get Started with Python Developm... Updated
Get Started with Jupyter Notebooks Updated
Get Started with WSL Updated
More...
Show welcome page on startup


File Edit Selection View
https://github.com/CosmicSnowman/ftcdocs.git
Clone from URL https://github.com/CosmicSnowman/ftcdocs.git
Clone from GitHub
Welcome
Visual Studio Code
Editing evolved
Start
New File...
Open File...
Open Folder...
Clone Git Repository...
Connect to...
Recent
ftcdocs-readme C:\Users\udayv\ftcdocs-dev
257 C:\Users\udayv
ftcdocs-readme [Dev Container: Dockerfile] /workspaces
cs225 [Dev Container: Existing Dockerfile] /workspaces
com.docker.devenvironments.code [Container docker/dev-envi...
More...
Walkthroughs
Learn the Fundamentals
GitHub Copilot Updated
Get Started with Python Developm... Updated
Get Started with Jupyter Notebooks Updated
Get Started with WSL Updated
More...
Show welcome page on startup
GIST [uvidyadharan]

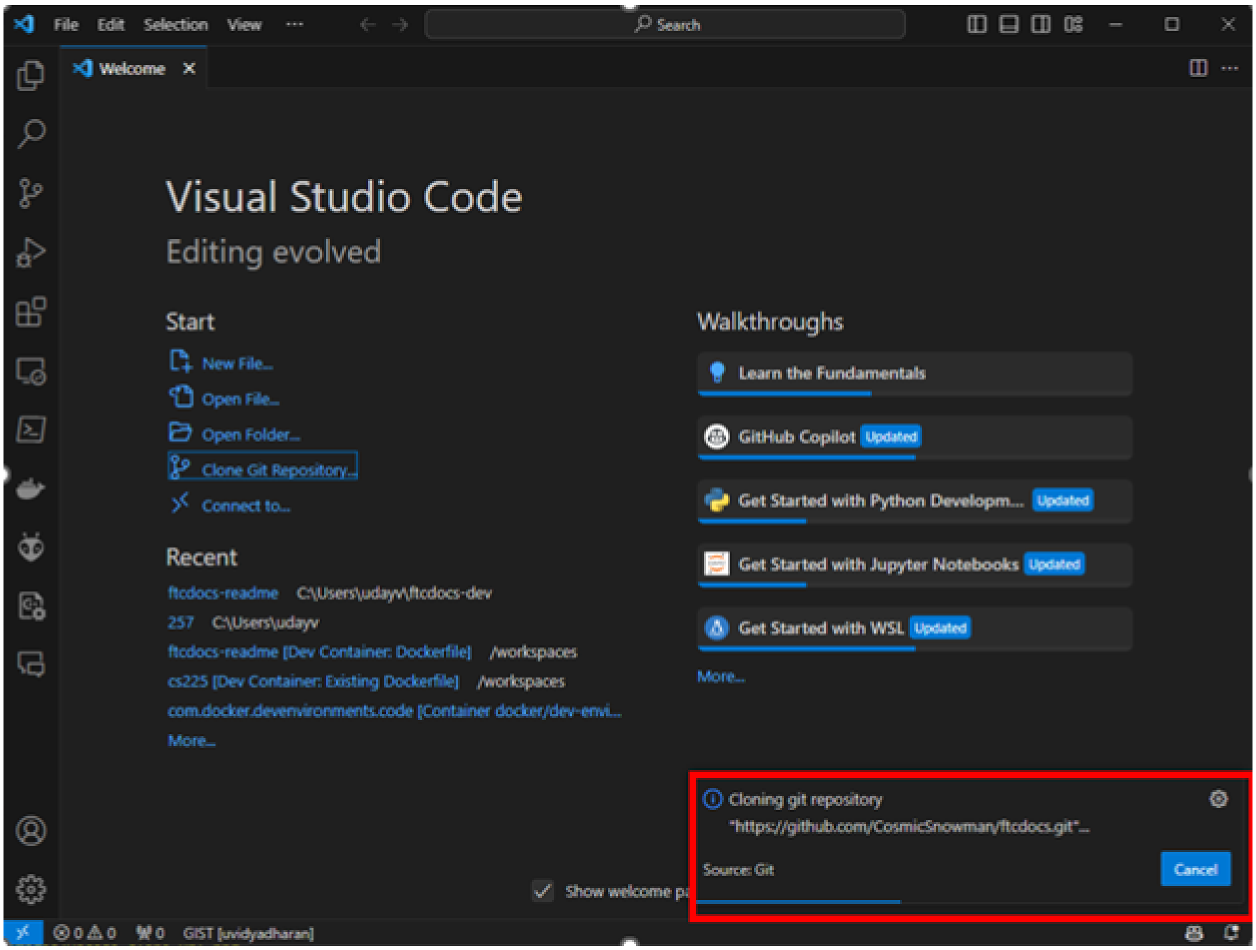


Fig. 9: You will then see a loading bar while the repository is cloned to your computer.

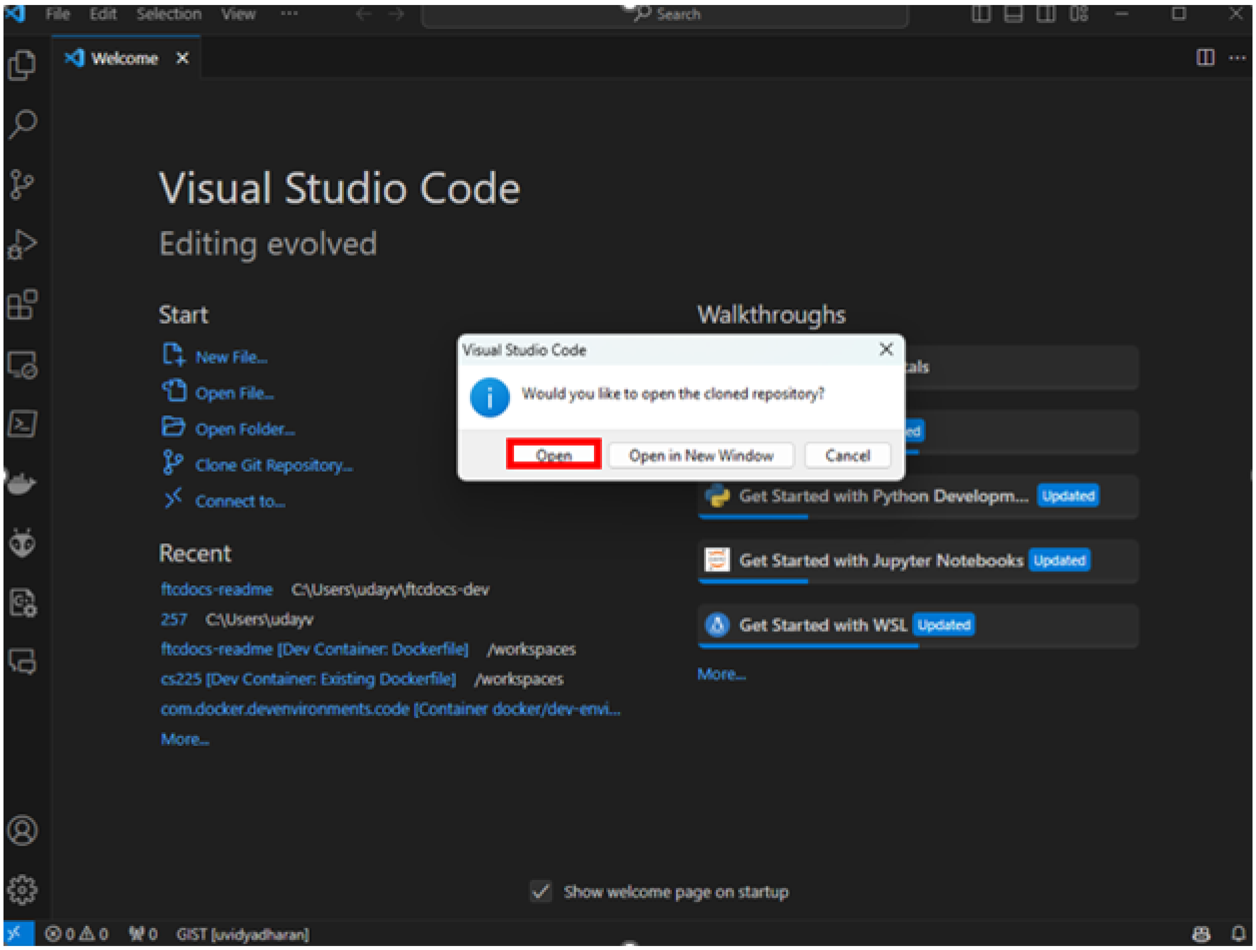
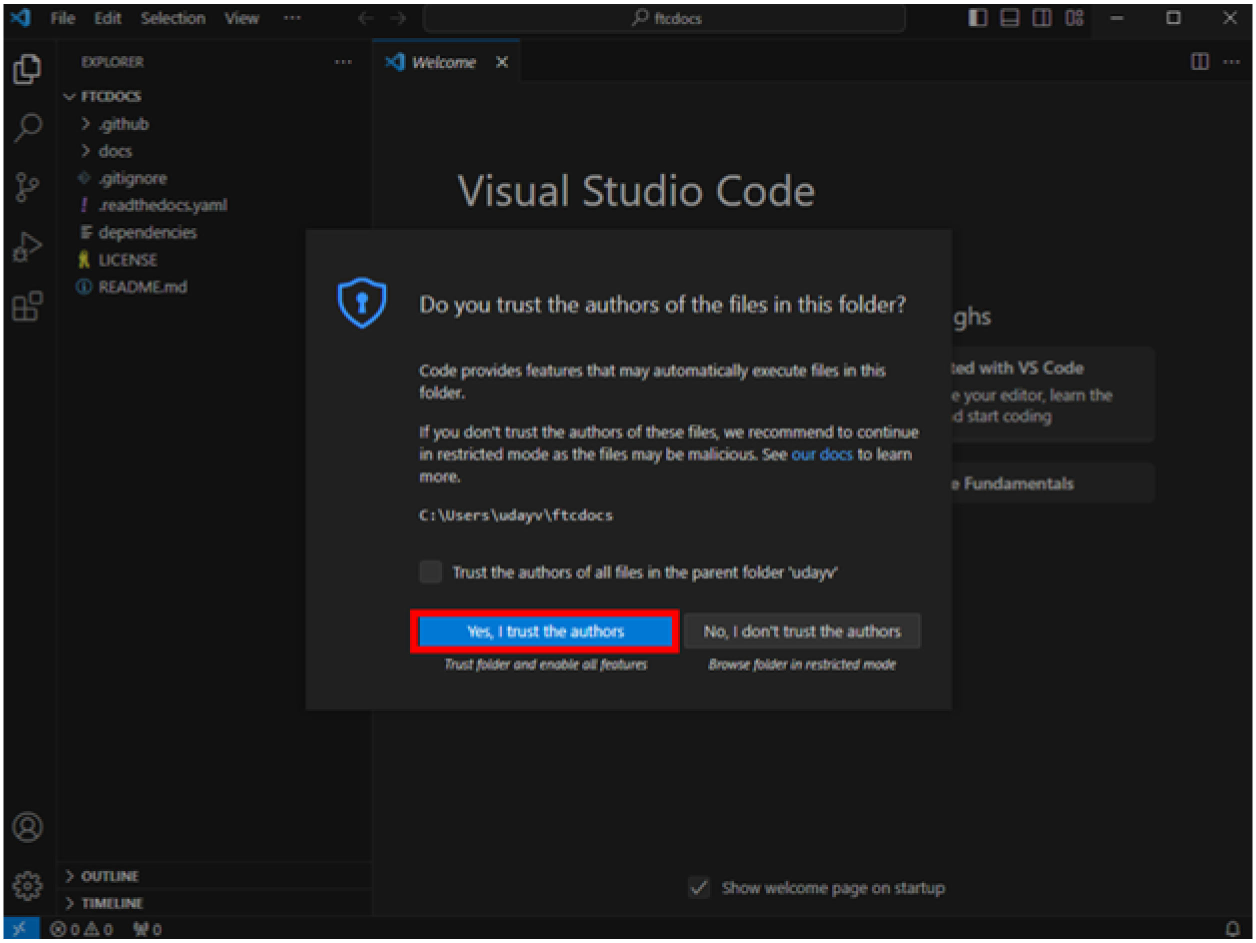


Fig. 10: Select "Open" to open the repository in VS Code.

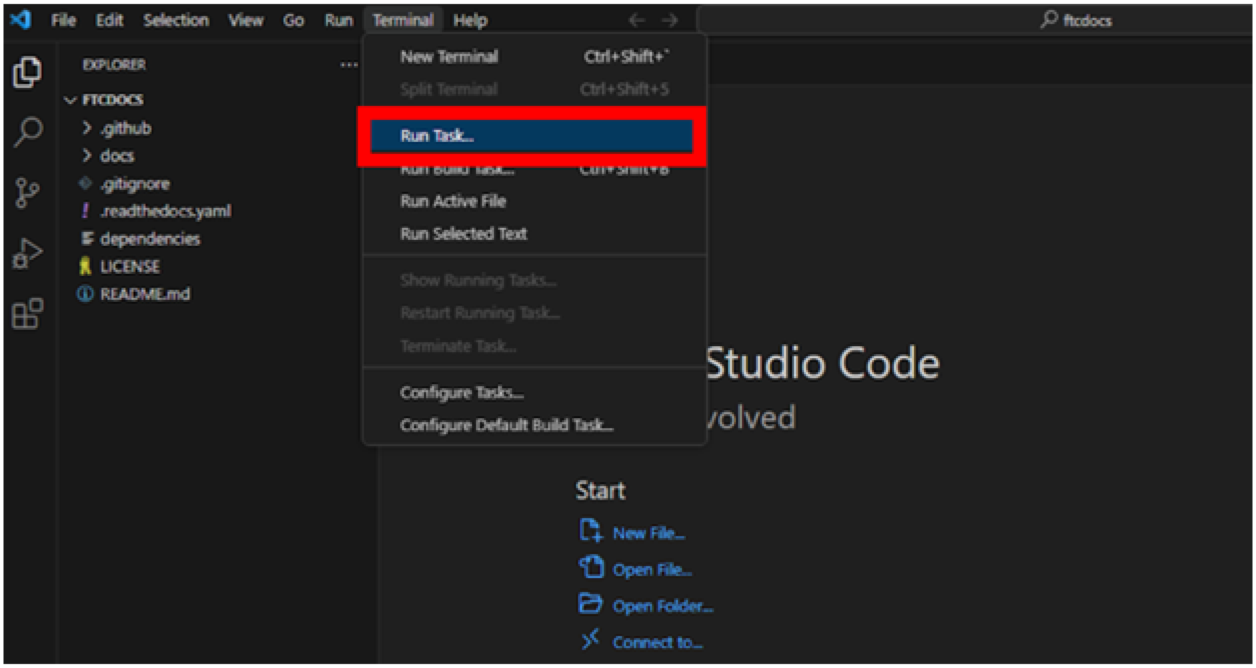
5. Select "Yes, I trust the authors"

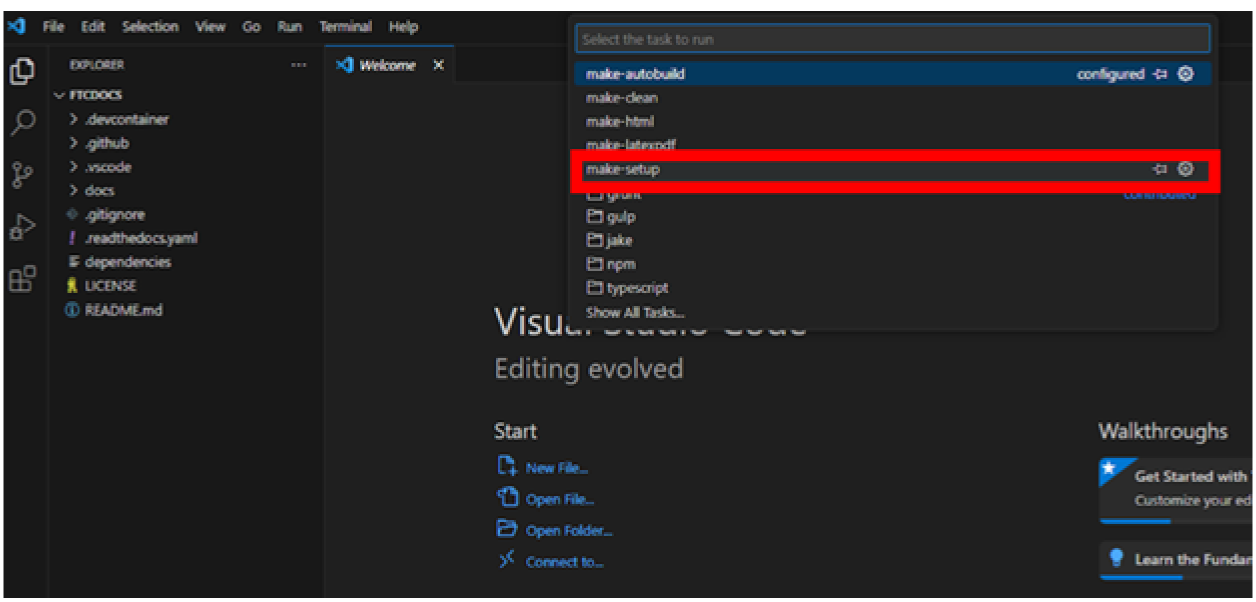


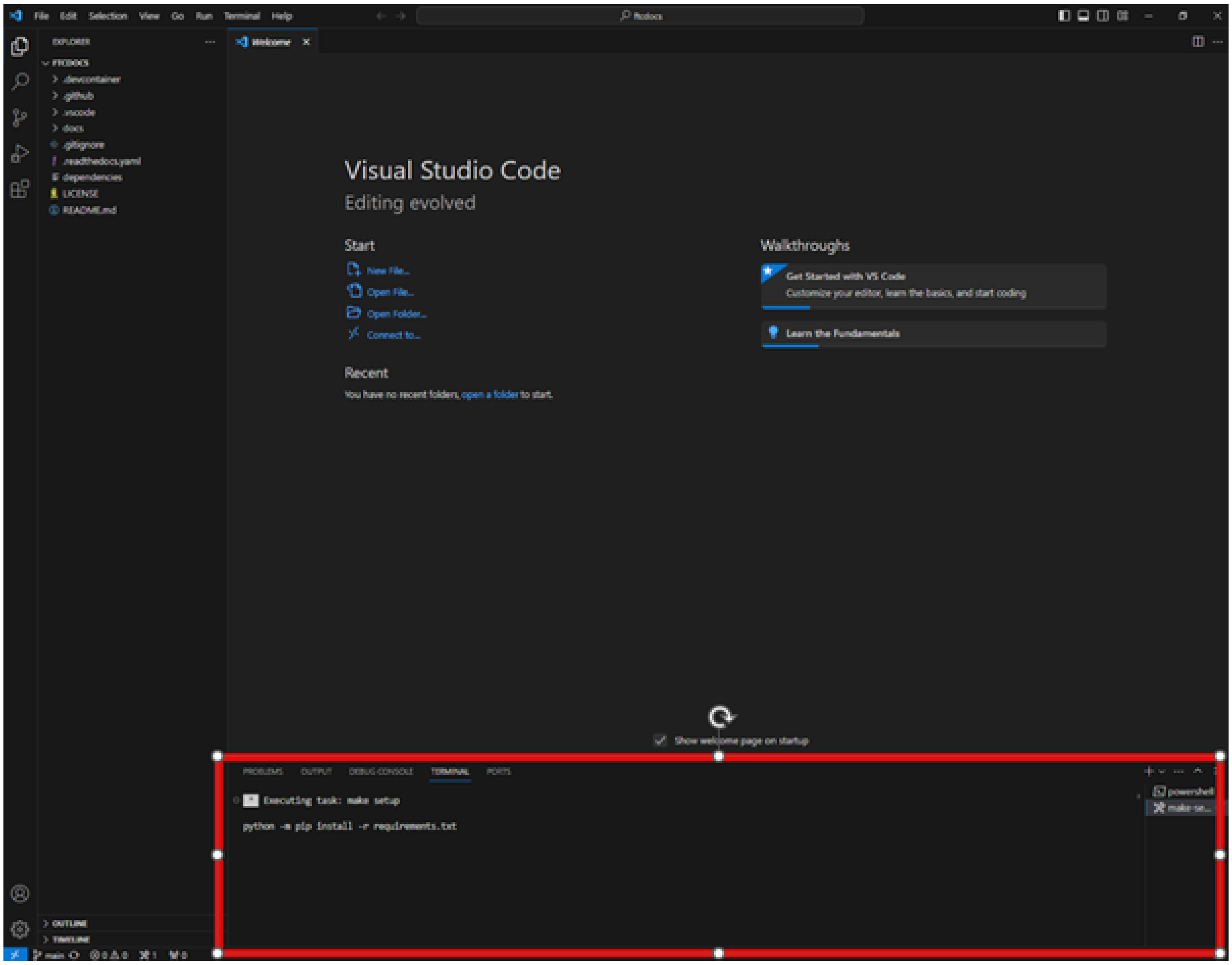
6. On the top ribbon of VS Code, click on "Terminal" and then "Run Task…"
7. On the new menu click on "make-setup". This task will only need to be run once per environment.
8. You will see a terminal window open and run a series of commands. This will take a few minutes to complete.
9. Once you see the message "Terminal will be reused by tasks, press any key to close it." you can move to the next step.
10. To test that everything is working, press *Ctrl + Shift + B* to build the site. You should see a terminal window open and run a series of commands. This will take a few minutes to complete.
11. **Once the build is complete, you will see "build succeeded" in the terminal window.**

    You can now click on the url `http://127.0.0.1:7350` to view the site.

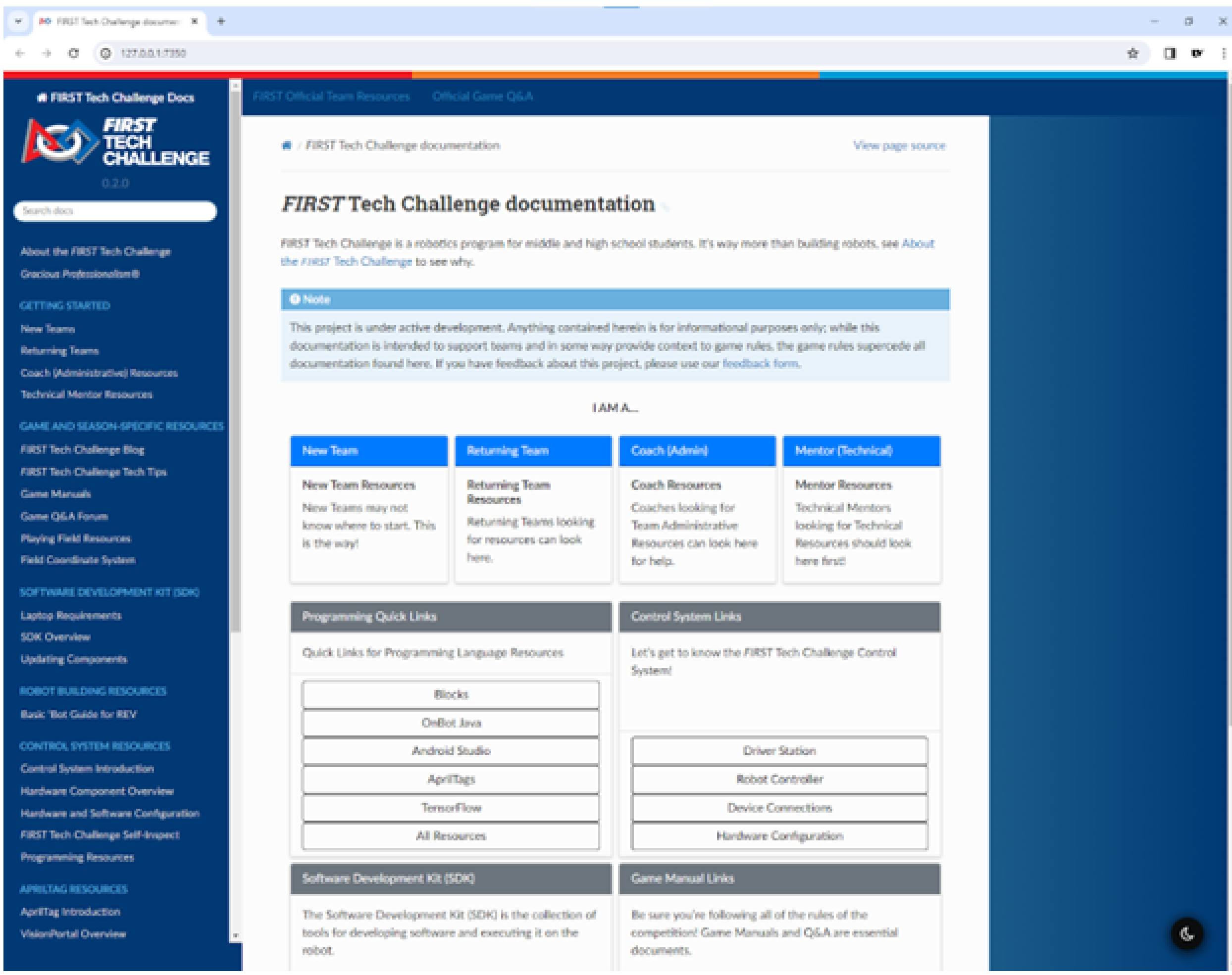
You are now ready to start developing! This version of the site will automagically update as you make changes to the source files. To stop the server, press *Ctrl + C* in the terminal window. To restart the server, press *Ctrl + Shift + B*.


File Edit Selection View Go Run Terminal Help
ftcdocs
EXPLORER
FTCDOCS
.github
docs
.gitignore
.readthedocs.yaml
dependencies
LICENSE
README.md
New Terminal Ctrl+Shift+`
Split Terminal Ctrl+Shift+5
Run Task...
Run Active File
Run Selected Text
Show Running Tasks...
Restart Running Task...
Terminate Task...
Configure Tasks...
Configure Default Build Task...
Studio Code
volved
Start
New File...
Open File...
Open Folder...
Connect to...


File Edit Selection View Go Run Terminal Help
EXPLORER
Welcome
FTCDOCS
.devcontainer
.github
.vscode
docs
.gitignore
.readthedocs.yaml
dependencies
LICENSE
README.md
Select the task to run
make-autobuild configured
make-clean
make-html
make-latexpdf
make-setup
gulp
jake
npm
typescript
Show All Tasks...
Editing evolved
Start
New File...
Open File...
Open Folder...
Connect to...
Walkthroughs


Visual Studio Code
Editing evolved
Start
Walkthroughs
Get Started with VS Code
Learn the Fundamentals
Recent
Executing task: make setup
python -m pip install -r requirements.txt


PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
build succeeded.
[I 240212 19:14:59 server:335] Serving on http://127.0.0.1:7350
[I 240212 19:14:59 handlers:64] Start detecting changes


FIRST Tech Challenge Docs
FIRST TECH CHALLENGE
0.2.0
About the FIRST Tech Challenge
Gracious Professionalism®
GETTING STARTED
New Teams
Returning Teams
Coach (Administrative) Resources
Technical Mentor Resources
GAME AND SEASON-SPECIFIC RESOURCES
FIRST Tech Challenge Blog
FIRST Tech Challenge Tech Tips
Game Manuals
Game Q&A Forum
Playing Field Resources
Field Coordinate System
SOFTWARE DEVELOPMENT KIT (SDK)
Laptop Requirements
SDK Overview
Updating Components
ROBOT BUILDING RESOURCES
Basic 'Bot Guide for REV
CONTROL SYSTEM RESOURCES
Control System Introduction
Hardware Component Overview
Hardware and Software Configuration
FIRST Tech Challenge Self-Inspect
Programming Resources
APRILTAG RESOURCES
AprilTag Introduction
VisionPortal Overview
FIRST Official Team Resources
Official Game Q&A
FIRST Tech Challenge documentation
View page source
FIRST Tech Challenge documentation
FIRST Tech Challenge is a robotics program for middle and high school students. It's way more than building robots, see About the FIRST Tech Challenge to see why.
Note
This project is under active development. Anything contained herein is for informational purposes only; while this documentation is intended to support teams and in some way provide context to game rules, the game rules supercede all documentation found here. If you have feedback about this project, please use our feedback form.
I AM A...
New Team
Returning Team
Coach (Admin)
Mentor (Technical)
New Team Resources
New Teams may not know where to start. This is the way!
Returning Team Resources
Returning Teams looking for resources can look here.
Coach Resources
Coaches looking for Team Administrative Resources can look here for help.
Mentor Resources
Technical Mentors looking for Technical Resources should look here first!
Programming Quick Links
Quick Links for Programming Language Resources
Blocks
OnBot Java
Android Studio
AprilTags
TensorFlow
All Resources
Control System Links
Let's get to know the FIRST Tech Challenge Control System!
Driver Station
Robot Controller
Device Connections
Hardware Configuration
Software Development Kit (SDK)
The Software Development Kit (SDK) is the collection of tools for developing software and executing it on the robot.
Game Manual Links
Be sure you're following all of the rules of the competition! Game Manuals and Q&A are essential documents.

30.4.7 Creating a New Branch

Repeat Codespaces Local

What are branches and why do we need them?

Branches can be thought of as parallel versions of a project. This is useful for deveopment because it allows you to work on a feature or bug fix in an isolated environment without affecting the main project. Once you've made the changes you want to make, you can merge your branch back into the main branch to publish your changes. In the case of the FTC Docs, the main branch of your fork should always be a copy of the main branch of the main repository which serves as a reference point to create new branches from. After we are satisfied with the changes we've made in our branch, we will create a Pull Request to merge our changes back into the main branch of the *Main Repository*. After the Pull Request is approved, the changes will be merged into the main branch of the Main Repository and will be published to the FTC Docs website. After the changes are merged, we will use the process described in the *Updating Fork*

On the < > `Code` tab, we can see the Branch information below:

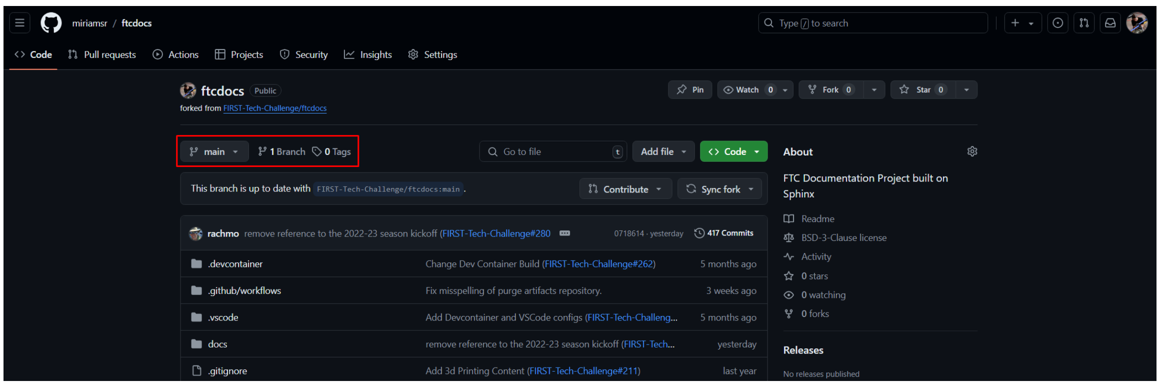


Fig. 11: Figure 3: Example Fork branch information

Here we can see, circled in red, the Branch information for the repo. We can see that the branch we're currently viewing is the `main` branch, and there is only one branch in the repo (there *always* has to be a `main` branch, so if there's only one branch then it's the `main` branch).

If we click on the branch drop-down (that currently says, "main") we will see a list of all the branches, and a search/create box. You can click on a branch name to switch the current view to a different branch, but if there are too many in the list you can just type in the name of the branch to filter the list. If you want to create a new branch, you can just type in the new name for the new branch and click the "Create branch <NAME> from 'main'" item that will appear. This is the "quick branch" way of creating branches.

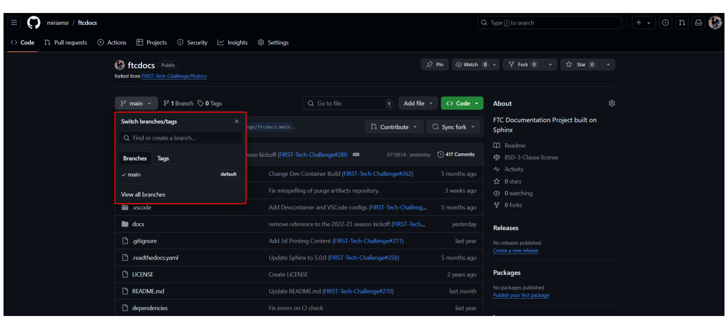


Fig. 12: Figure 4: Example Fork branch selection

In order to make any changes to the content, we need to create ourselves a new branch so that we have our own workspace in which to make changes. To create a new branch duplicated from `main`, we need to ensure `main` is the currently selected branch (if there's only one branch then it's got to be `main`) and then we can type a new name in the box and click the "create" selection that will appear. Let's call this new branch `demo`. For real work you should use a more descriptive name, like `feature-<FEATURE_NAME>`, `bugfix-<BUG_NUMBER>`, or `<YOUR_NAME>-<FEATURE_NAME>`.

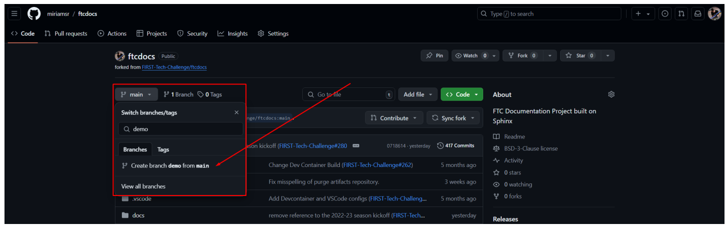


Fig. 13: Figure 5: Example Fork create new branch

After clicking the "Create branch" button from Figure 5, your new branch should be created and the branch is automatically selected in the branch selection drop-down. You will also see the number of total branches increase by one. If you want to see all branches, you can click the "View all branches" link within the branch drop-down, or you can also click the "X Branches" link next to the branch selector. This will show you all of the current branches along with which one(s) are owned by you.

Figure 6 shows the "All Branches" view. This is yet another way of managing your branches, and may or may not turn into your favorite way of viewing and managing branches - everyone has their own personal opinion. From within this view you can create new branches by clicking the green "New Branch" button, rename *YOUR* branches by clicking on the pencil icon to the right of each branch, or delete *YOUR* branches by clicking the "*Trash Can*" icon to the right of each branch. You can only perform actions on *YOUR* own branches unless you're an administrator (like me). You can also switch to another branch by clicking on the name of the branch.

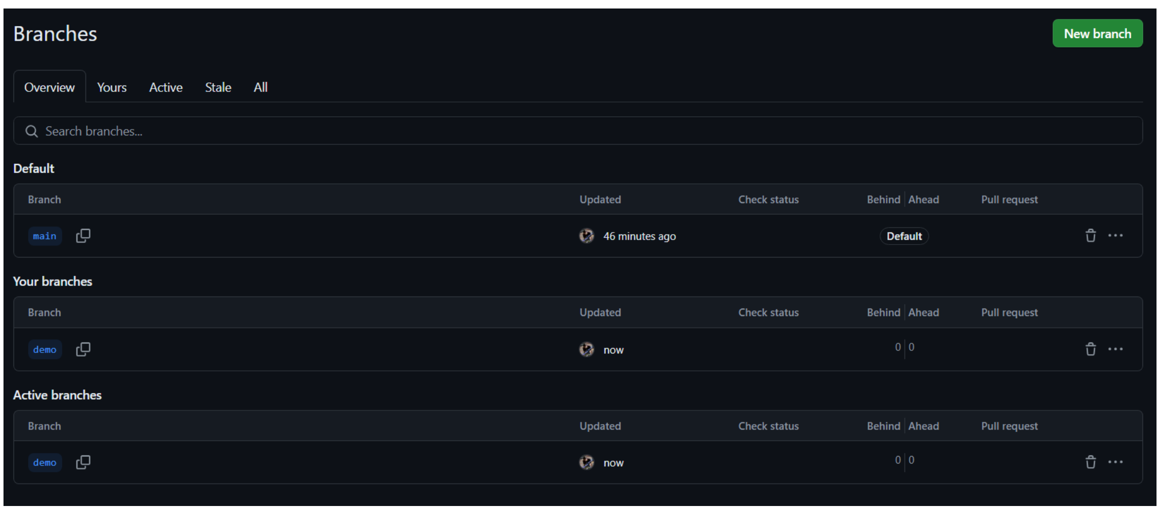


Fig. 14: Figure 6: Example Fork view all branches

Congratulations, you've created a new branch!

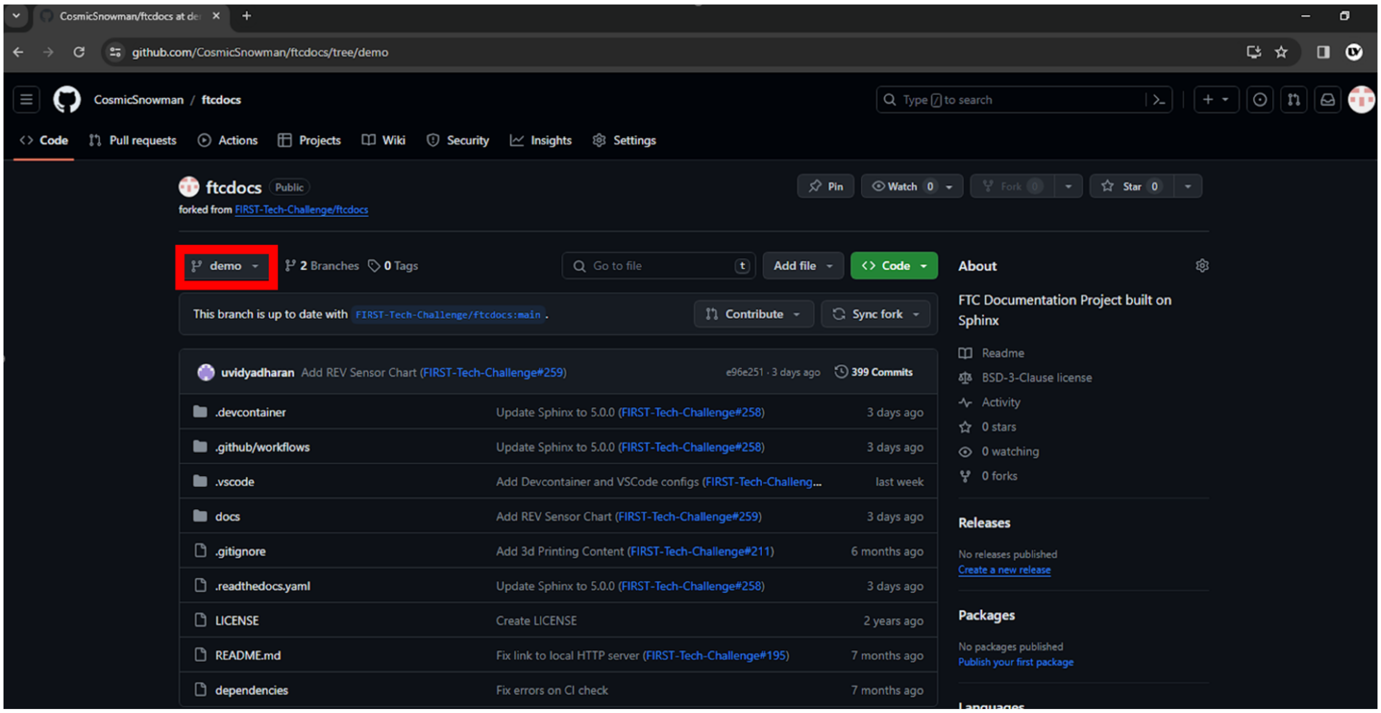
30.4.8 Creating a Codespace

Repeat Codespaces

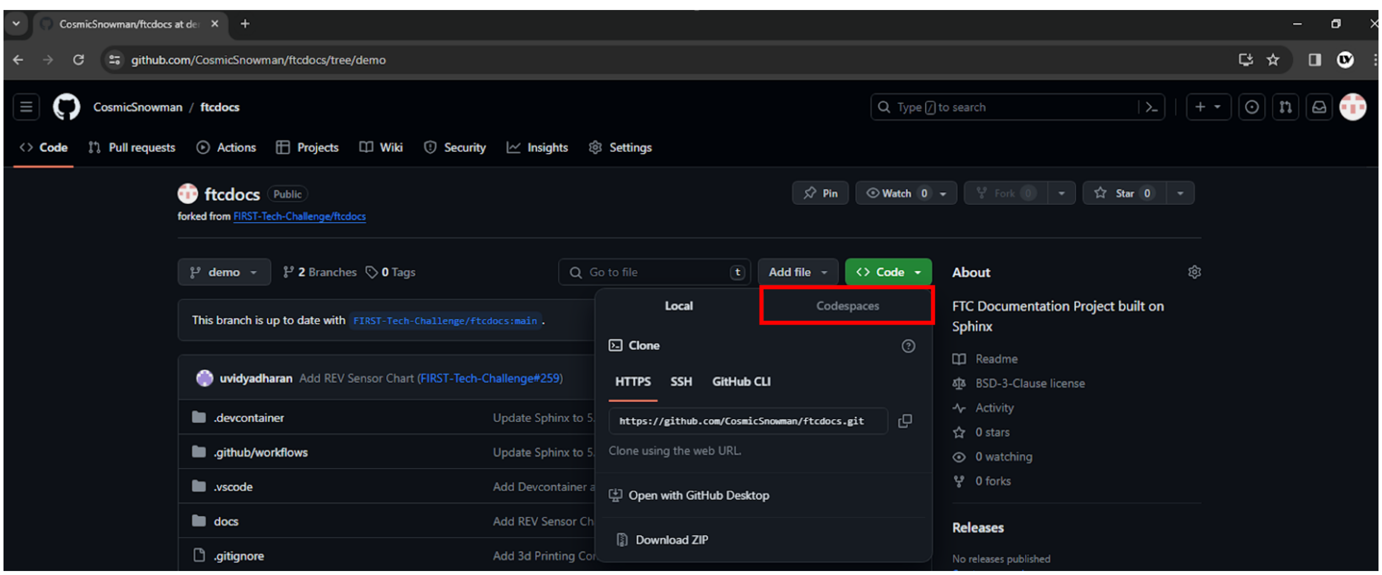
For every new branch you make in your repository, you must create a new codespace. This is a virtual environment that will allow you to run your code and test it before merging it into the main branch. It may take a few minutes to create the codespace, but once it is created, you can access it from the browser and subsequent access will be much faster.

Steps

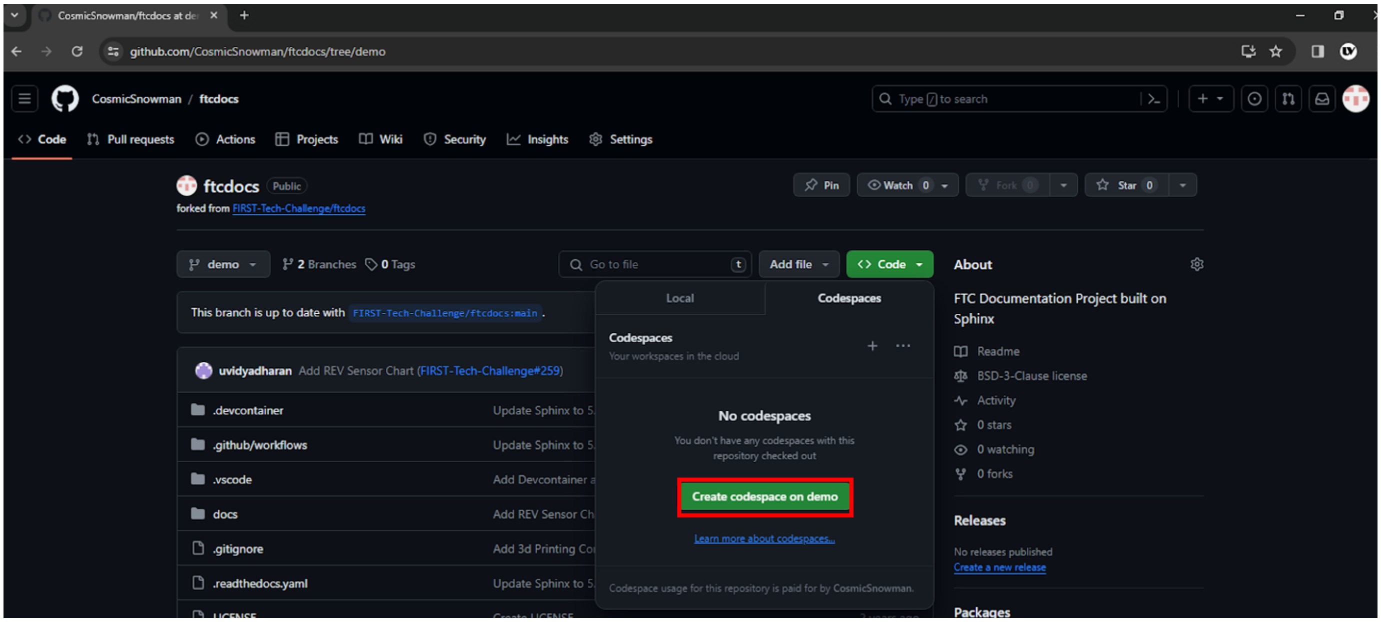
1. Open your **forked** repository in GitHub.
2. On the left side of the page select the branch you want to work on.



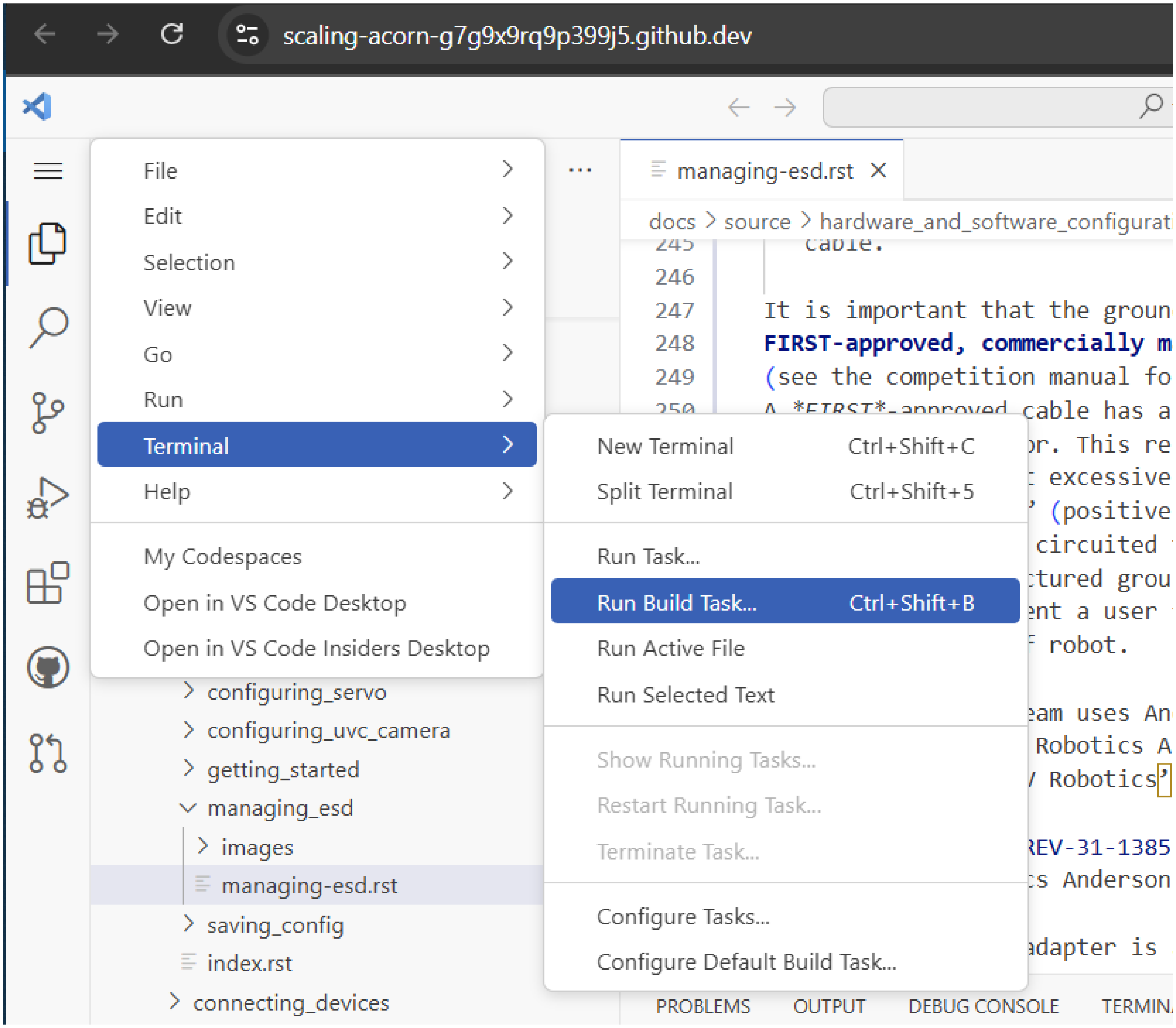
3. Click on the green Code button which will display a Local and a Codespaces tab, then click the Codespaces tab.



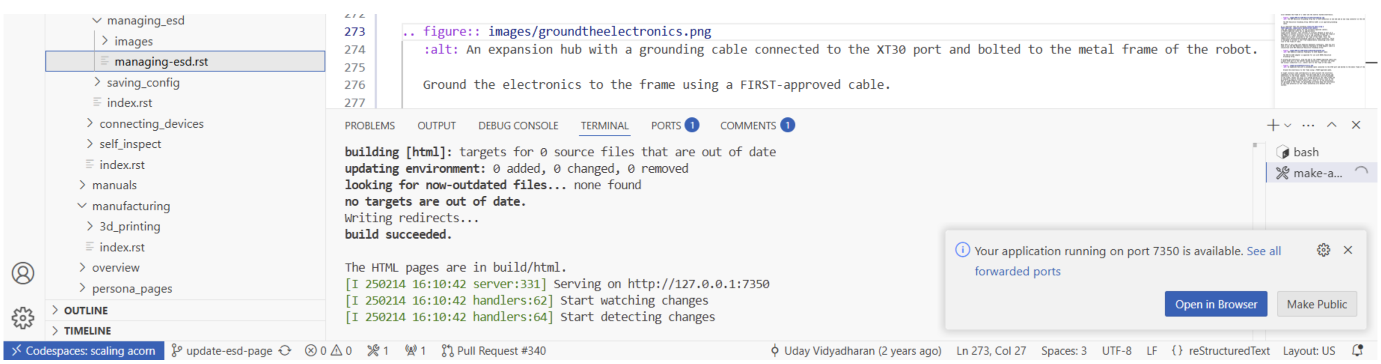
4. Click on the green button that says “Create codespace on”.



5. Wait for the codespace to be created. This may take a few minutes.
6. Once the codespace is created, you will be taken to the codespace in your browser. This is a browser based version of VS Code that can be used to make your changes and build the HTML pages to review your changes.
7. Enter `CTRL + SHIFT + B` to build the project. You can also run the build task from the Terminal menu.



8. The build messages will be displayed. Look to see a "build succeeded" message. Then a pop up will indicate an application is running.



9. If you click the Open in Browser button a new tab will open displaying the HTML pages you just built.
10. You can now make your changes. See the section on *Learning to create ReStructured Text (.rst)*.

30.4.9 Switching Branches

Repeat Local

> **Warning:** Only complete these steps if you have chosen to develop the site locally. If you are using **GitHub Codespaces** you should skip this section.

This step is necessary to change which branch you are working on. If you are working on a branch and want to switch to another branch, you can use the following command:

```
git checkout <branch_name>
```

Make sure to replace `<branch_name>` with the name of the branch you want to switch to and that it matches the name of the branch you want to switch to that you created in the previous step.

Troubleshooting

Branch Not Found

`error: pathspec '<branch-name>' did not match any file(s) known to git`

This error occurs when the branch you are trying to switch to does not exist. Make sure you have created the branch you are trying to switch that there are no typos. However, it can also occur when your local repo is not up to date with the remote repo. To fix this, you can run the following command:

```
git fetch
```

This command will update your local repo with the remote repo. After running this command, try switching branches again.

Uncommitted Changes

`error: Your local changes to the following files would be overwritten by checkout:`

This error occurs when you have uncommitted changes in your working directory. You can either commit your changes, stash them, or delete them. To commit your changes, you can use the following command:

```
git commit -m "Your commit message"
```

This command will commit your changes with the message you provide.

To stash your changes, you can use the following command:

```
git stash
```

Stashing allows you to save your changes for later without committing them. After stashing your changes, you can switch branches. To apply your stashed changes, you can use the following command:

```
git stash apply
```

It is best to use `git stash` when you are not ready to commit your changes but need to switch branches.

To delete your changes, you can use the following command:

> **Warning:** This command will delete all uncommitted changes in your working directory. It is not recommended to use this command unless you are sure you want to delete your changes.

```
git reset --hard
```

30.4.10 Intro to VS Code Tasks

Information

In order to simplify the process of building and running FTC Docs we have created a set of tasks for Visual Studio Code. These tasks are defined in the *.vscode/tasks.json* file and can be run via the *Terminal -> Run Task…* menu.

Tasks

`make-setup`

This task will install the necessary dependencies for building FTC Docs. This task should only need to be run once.

`make-html`

This task will build the HTML version of the FTC Docs. The output will be placed in the *html* directory. This will not automatically create a local server to view the output. In most cases we would instead recommend using the *make-autobuild* task.

`make-latexpdf`

This task will build the PDF version of the FTC Docs. The output will be placed in the *latex* directory. In the backend it generates the LaTeX files and then runs *pdflatex* to generate the PDF. This build functionality is not supported in this tutorial though Codespace users can use this task. The PDF output is stored as `docs/build/latex/ftcdocs.pdf`. Note that this task takes a long time to run and is not recommended for frequent use.

`make-autobuild`

This task will build the HTML version of the FTC Docs and then start a local server to view the output. It will also automatically rebuild the HTML version of the FTC Docs whenever a file is changed. This is the recommended way to view the FTC Docs while working on them. It is also hotkeyed to *Ctrl+Shift+B* for easy access. You can stop this task by pressing *Ctrl+C* in the terminal.

make-clean

This task will remove all of the build artifacts. This is useful if you want to start from a clean slate. This is sometimes necessary if you are not seeing your changes reflected in the output as will sometimes happen with images and other static files.

30.4.11 Learning to create ReStructured Text (.rst)

ReStructured Text Overview

This overview serves to describe the overall purpose and functionality of reStructured Text (rST) files.

Overview

From the outset, let me say that "reStructured Text" is probably a bit of a misnomer. It's more like "Text" that uses certain consistent patterns (also known as *markup*) in order to instruct a tool on how that text should ultimately look or feel - for example, whether or not a passage should be *italicized*, **bolded**, or if an image should be displayed in that location in the final document. ReStructured Text is used to generate *minimally-interactive web pages*, PDF documents, and other forms of output in such a way that no actual web programming or PDF publishing experience is necessary AND is still human-readable. In this way, reStructured Text can literally be written once and then exported to lots of different formats and platforms. Comparable web programming and PDF authoring tools are not, in the opinion of this author, readable by the average person and cannot be easily translated between output formats whereas "reStructured Text" is.

History

Now a very brief history lesson: In the early 1990's the first "Structured Text" markup languages were developed - these languages were used to help develop different kinds of program documentation for the quickly developing software industry. This created a universal standard for adding symbols and directives to plain text so that it could be processed by generic tools. This way the documentation could be read and formatted any way a user or company wanted - it could be converted into printer commands for printing onto paper, it could be rendered onto a screen, it could be read aloud by a text-to-voice system, or it could be used in lots of other ways. Unfortunately, as is usual in this business, the tools quickly became too complicated or not flexible enough for the needs of its users. In the early 2000's, elements of "Structured Text" were "revised, reworked, and reinterpreted" into a new format, which ultimately became called "reStructured Text" (rST).

Example

As a quick example, consider the following text:

```
The quick brown fox jumped over the lazy dog.
```

Let's assume we want to **bold** the nouns in the sentence, so in rST format we would add two asterisks (**) to the front and end of each word to make them **bold**. Let's do that by modifying the text to show the following:

```
The quick brown **fox** jumped over the lazy **dog**.
```

This would ultimately output text that looks like:

> The quick brown **fox** jumped over the lazy **dog**.

Now let's make the word `quick` be *italicized*. In rST format, we *italicize* words or sections by adding a single asterisk (*) to the front and end of each section we want to *italicize*. It would now look like so:

```
The *quick* brown **fox** jumped over the lazy **dog**.
```

This would output text that looks like:

> The *quick* brown **fox** jumped over the lazy **dog**.

And finally, let's add a link on the word "over" to the dictionary.com reference of the word over. To do this, we use a slightly more complex piece of markup that includes "back ticks" (`) and ends in an underscore (_), like this:

```
The *quick* brown **fox** jumped `over <https://www.dictionary.com/browse/over>`_ the lazy **dog**.
```

And this sentence with several different kinds of *markup* finally would be displayed as:

> The *quick* brown **fox** jumped over the lazy **dog**.

This was just a quick example of what can be done with rST and how it is accomplished. Move on to the other elements of this tutorial to learn more about rST.

Creating rST Documents

Note: It is assumed that you have created a GitHub account and been given the proper permissions to work within this repository for training. If you can read this, you probably have, but double-check with your friendly FTC Support person.

Now that we have created our branch, now we can begin making our own special changes to our branch - i.e. adding or modifying content. Remember, our branch is our place to make any kinds of changes we want, we just have to understand *where* and *how* to make those changes.

Understanding the ftcdocs Structure

The ftcdocs repository is a "map" of the files and how they'll be presented to users on the website. Filenames and where they're located in the folder structure of the repository define the ultimate web experience users will have interacting with your final documentation.

Quick GitHub Folder Listing Intro

For those unfamiliar with the GitHub folder (also known as "directories") structure listing, here's a quick introduction:

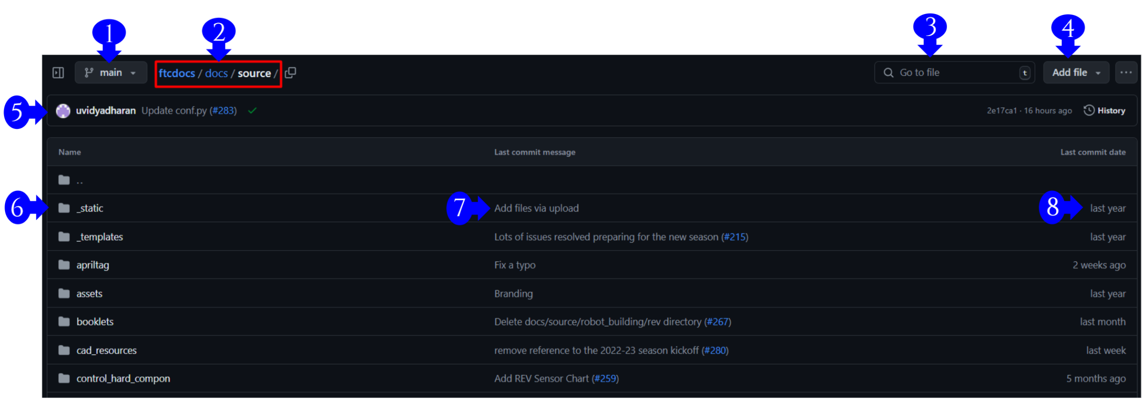


Fig. 15: GitHub Folder Listing Info

1. Branch Indicator and Selection - Here you can see which branch you're viewing, and select another branch if you want to view a different one. Test 123
2. Repository Path Location of the currently viewed folder - This helps you understand where you are within the Repository. On the main page of the repository there is no path shown, but on all other locations within the repository the path indicates where you are.
3. File Search Tool - If you can't find a file by browsing through the folder, or you want to "quick-find" a particular file (useful for listing all of the "index.rst" files especially) this tool can help you out.
4. New File Creation Tool (can also create folders) - this can help you create new files (or files within new folders) via the web client.
5. Last user to commit a change, and description of the change - this is helpful to understand what the last changes were to the repository you're looking at. This is mostly useful when trying to remember what change was last made to a particular branch.
6. Folder/File Name - Within the row, this shows you a given folder or file for fast navigation.
7. Last Commit Description for changes within the Folder/File - new users often mistake this for a description of the folder or file that this row is referring to, but this is actually the description of the last commit made to the folder or file and generally has absolutely nothing to do with what content is within the folder or file. This can be really confusing for new users, FYI.
8. How long ago the last commit was made - this way you can see how long ago changes were made to folders or files. This information shows you how potentially outdated content may be.

Repository Main Folder

We're going to call the first folder in the repository the "**repository main folder**". This is the main folder where viewers of the repository can get important information regarding what the repo is for, what license the repository is managed by, and so on. As a contributor of this repository, most of the content here will not be important to you.

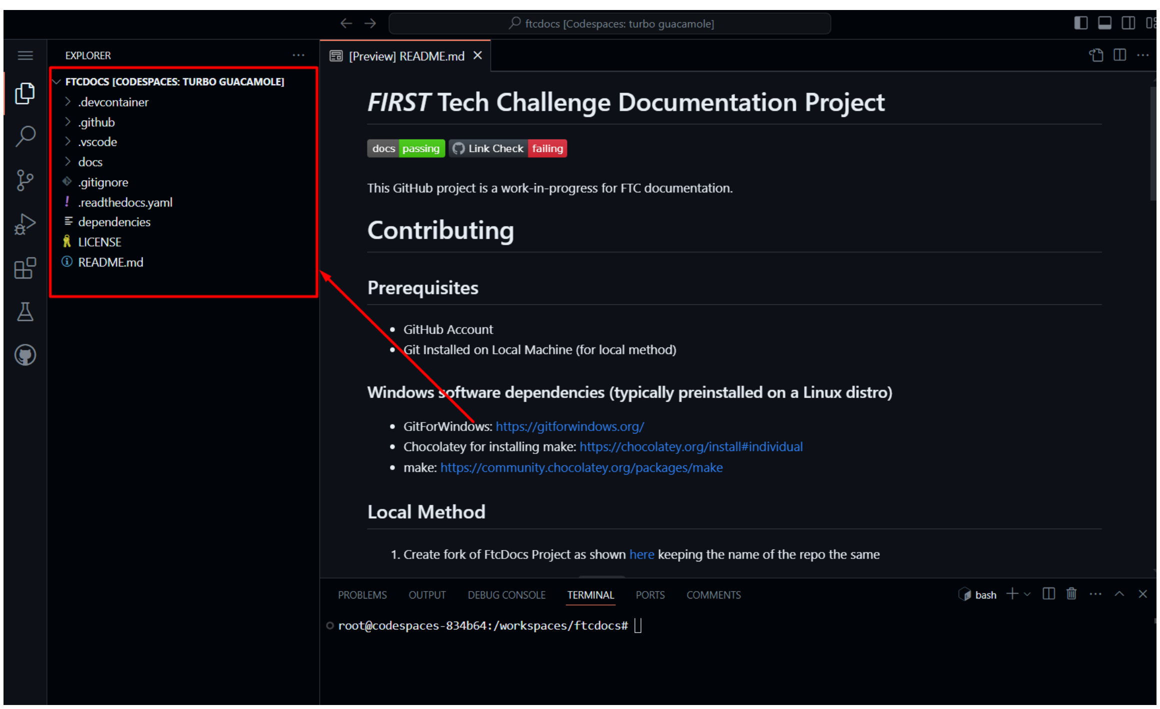


Fig. 16: Repository Main folder for rst-primer the quick brown fox

Docs Folder

Within the **repository main folder** will almost always be a folder that contains the actual project files for the project - for the *rST-Primer* repository, that folder is the `docs/` folder. The `docs/` folder is where the top-level build definitions are for building the documentation.

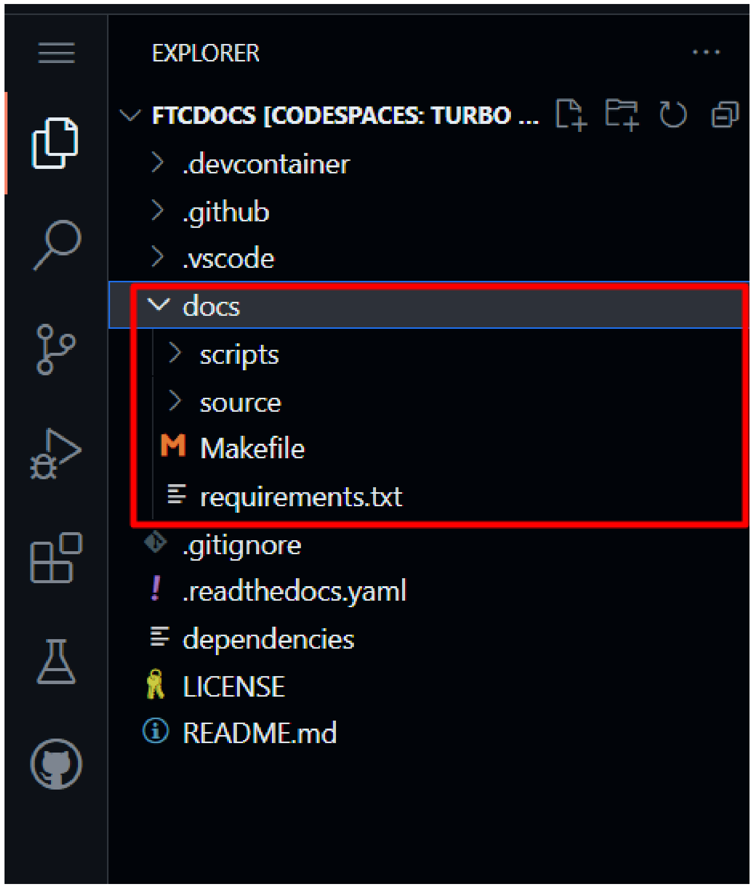


Fig. 17: Docs Folder

Again, as a documentation contributor the contents of this folder will likely not be of interest to you right now, but for project builders and maintainers this is where the "magic happens." The important element here is that this is where the `source/` folder is located. The `source/` folder is the actual starting point for the documentation and the rST-primer website.

Source Folder

Within the `docs/` folder you will find the `source/` folder. Please take careful notice that above the folder listing we can see the full repository *path* of where these files and folders are within the repository; we are currently viewing the current contents of `rst-primer/docs/source`.

When we work on documentation, it's important to understand that there are TWO related paths that we must keep track of:

Repository Location
: This is the location of the file within the repository. This is conveniently shown in the path within GitHub. This location is only used as a reference within GitHub, where the full *repository location* is sometimes needed.

Document Location
: This is the location within the final website where the document will exist. The document location for files is the same as the *repository location* minus the `rst-primer/docs/source`. For example, a file with a *repository loca-*

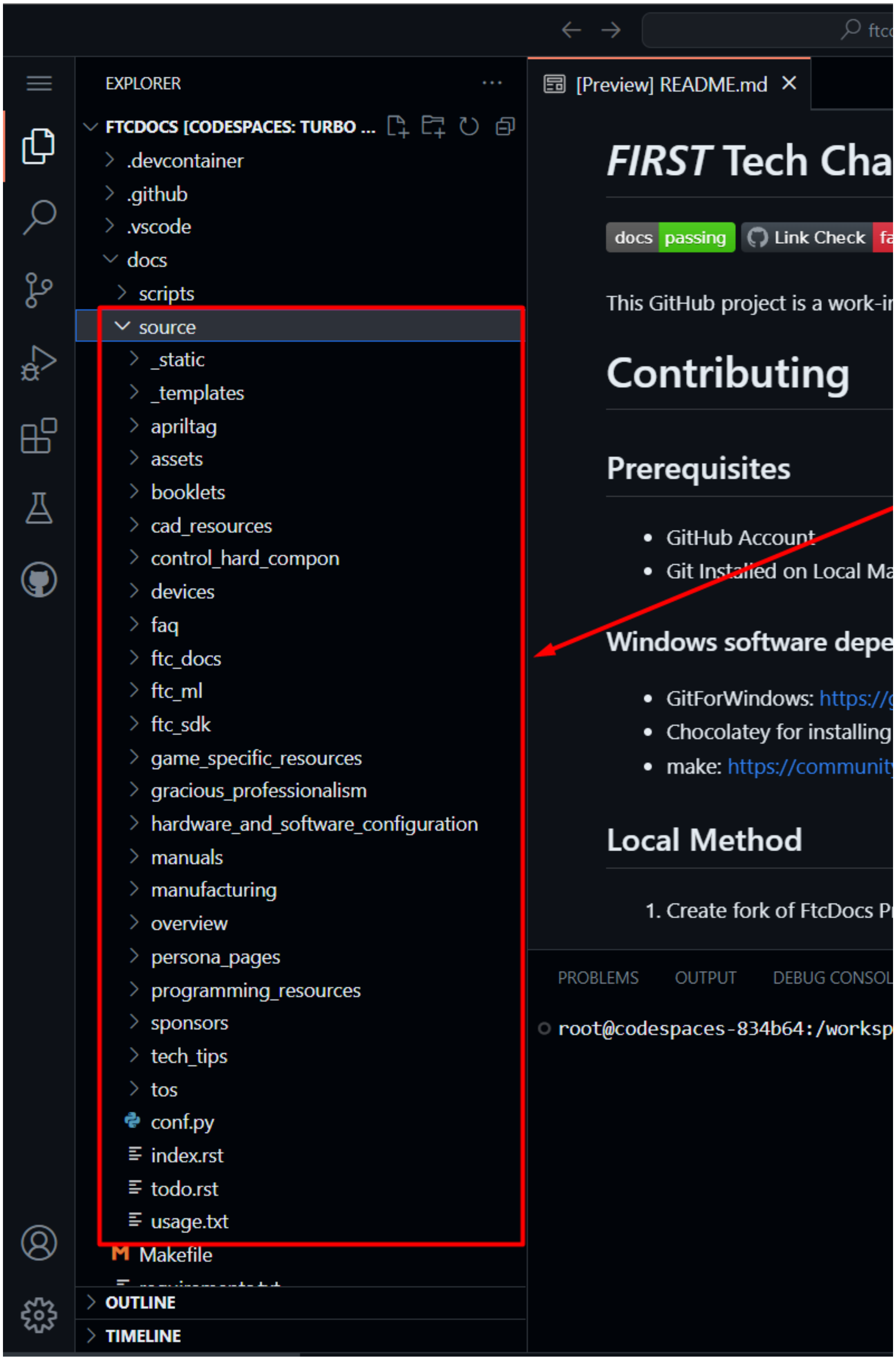


Fig. 18: Source Folder

tion of `rst-primer/docs/source/tutorial/overview/overview.rst` would have a *document location* of `/tutorial/overview/overview.rst`. We will use this *document location* within our documents.

We will refer to these different document locations within this and future documents.

As previously stated, this `rst-primer/docs/source` repository location is the main starting point for the documentation in all forms that it can take - e.g. this is the "root" location for the site's PDF document, is the "root" location for the html website, and so on. This "root" location has some very special files and folders:

_assets
The *_assets* folder contains asset files that are used site-wide, such as the *FIRST* Tech Challenge logos and icon. Assets that could be included on all pages of the site, regardless of the content on the page, can be placed here for easy inclusion within templates and static content. However, any assets that are specific to content on the site should be placed with the content documentation instead. This folder starts with an `_` underscore symbol, meaning it's designated as a special folder whose contents should only be modified with the help of site administrators.

_static
The *_static* folder, a *special folder* signified by the underscore prefix in its name, contains HTML-specific code that is necessary to format the documentation in HTML. This code should never need to be edited or updated by a document contributor, but if you feel you do then please coordinate any changes with an administrator.

_templates
The *_templates* folder, a *special folder* signified by the underscore prefix in its name, is used by Sphinx's templating engine, Jinja, to override the way Sphinx manages its basic theme to allow customizations of the page layout, the header content, and the footer content (among others). Again, this isn't likely something a document contributor should need to edit (other than possibly adding content if you feel the need) but please work with an administrator to coordinate any updates to the HTML site templates.

conf.py
The **conf.py** file is a special file for sphinx that defines all of the modules, parameters, and tools used to build the documentation in all forms (HTML, PDF, etc…). This file does not have an underscore prefix, however this file should never be modified without first consulting an administrator.

index.rst
The **index.rst** file here in the "root" documentation location is the primary organization and layout file for the HTML version of the documentation. This **index.rst** defines the *Table of Contents* (typically seen on the left-hand pane of the HTML website) as well as the content on the main page of the website. There may be multiple *index.rst* files within the website, but **this** particular file is the master index file for the website.

todo.rst
The **todo.rst** file is an auto-filled file by the "To-Do" module. This will be discussed in a much later tutorial, but for now we're just going to pretend that this file doesn't exist.

Understanding Content Structure

Once you understand the rst-Primer repository structure, now we're ready to understand how to add *content* to this structure. A basic example of the content structure can be found in the `/tutorial` folder structure. The general rules are defined as:

- Each content page is defined by a reStructuredText file with an **.rst** extension.
- Each **.rst** file is defined within its own folder.
- Each **.rst** file generally has supporting files (images, included files, etc…) and so it will have subfolders that include those files (`/images`, `/includes`, etc…)
- Content categories can have category-specific landing pages, and those landing pages are `index.rst` files. These category pages don't generally contain content themselves, except as an introduction to the content linked from those pages (similar to the master index `index.rst` file).
  - Content Categories is used heavily within ftcdocs, but we will likely cover this topic in a much later tutorial.
- **.rst** files must be referenced from at least **one** `index.rst` page.

The best way to describe these rules are to show an example of these rules in practice.

Simple Content Example

For a simple example, let's take a look at the folder hierarchy of the *Gracious Professionalism* content on the ftcdocs website. The *Gracious Professionalism* content is a single page on the site devoted to describing *GP* to site visitors and honoring Dr. Flowers.

Here is a view of all of the folders and files that are involved within the structure of the *Gracious Professionalism* content:

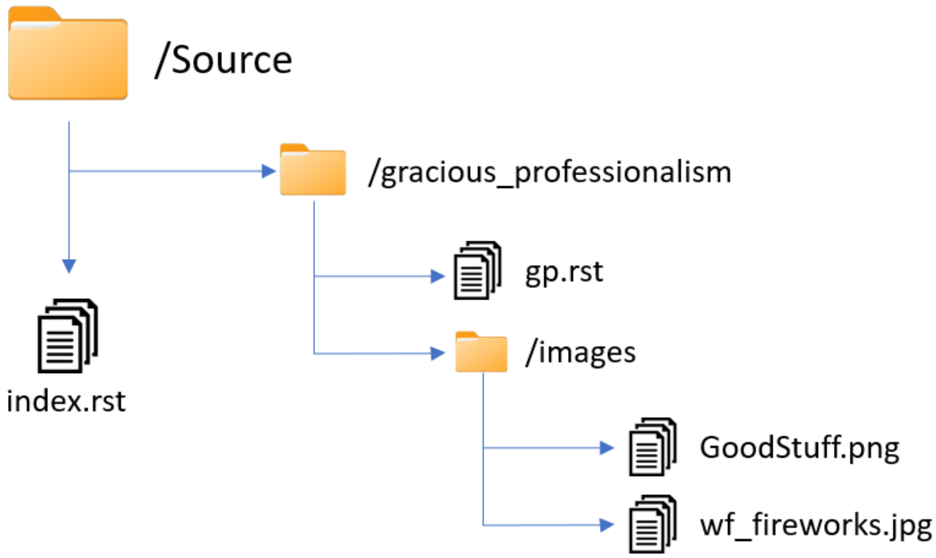


Fig. 19: ftcdocs structure for Gracious Professionalism content

Let's pick this apart a little to understand the structure of the content better. We will learn later *why* this structure is important once we start creating our own content.

1. The site's `/source` root for `ftcdocs`, similar to `rST-Primer`, is in the `ftcdocs/docs/source` repository folder.
2. The main content file is named `gp.rst` and lives within the `/gracious_professionalism` fol der within the `/source` folder. Files and folder names cannot include spaces, so the general site-wide style is to use underscores _ for folders and dashes - for filenames where spaces would otherwise be used. You can see this is more of a "guideline" rather than a rule, especially when dealing with externally-sourced files.
3. The `gp.rst` file references/uses two locally-stored images, and so those two images are stored within a `/images` folder located immediately within the same folder that the **.rst** file is stored.
4. Because the content is a single page with high-level content, it is included within the main site `index.rst` file (we'll see *how* it's included a little later).

Feel free to browse the content folder on the ftcdocs github repo.

Creating a New Content Document

Note: The process of creating a new content document is the same whether you're doing it locally or via codespaces as they both use vscode as the editor.

We're now going to create our first document. Let's have our document follow a similar structure as the *Gracious Professionalism* document above. Let's follow this structure:

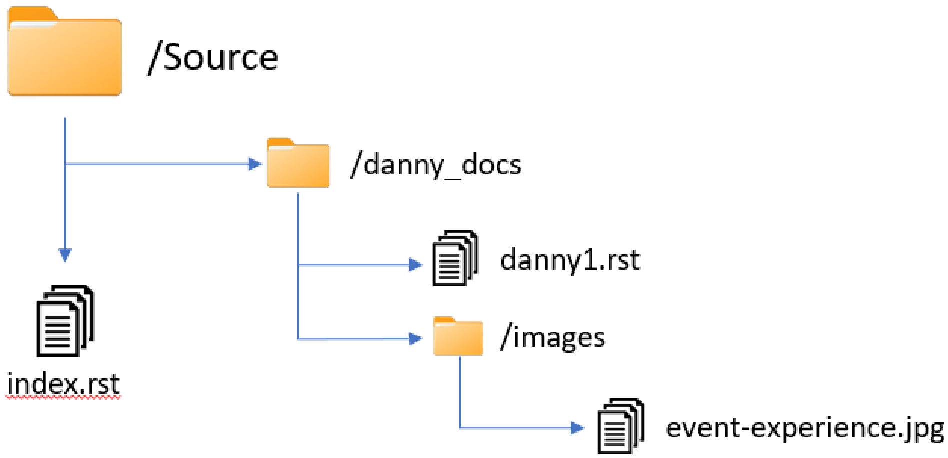


Fig. 20: Folder Structure for Danny Content

Of course it's best to create your own folder and document names, so that you don't overlap other people's work. This is being done within your own branch so even if others use the same folder structure you won't collide, but we do eventually want to merge your branch back to the `main` branch, and we don't want thing colliding there. So just like your branch name, use your name as the unique key for the folder name at least.

Step 1: Create the initial content file

Our first step in creating content is to create the core document. We're just going to create a "stub" file initially that we'll edit later.

1. Navigate to the `/rst-primer/docs/source` folder in VS Code.
2. Create a new folder with your name (or a unique name) within the `/source` folder. You can do this by right-clicking on the `source` folder and selecting "New Folder" and then typing in your name. E.g. "Demo".
3. Create a new file within the folder you just created. You can do this by right-clicking on the folder you just created and selecting "New File" and then typing in the name of the file. E.g. "7350.rst".

Step 2: Upload image file to use

Next let's upload an image file that we're going to use in our document. reStructured Text allows you to use images in your documents, and those images can be local images (saved/stored in the repository) or remote images (using HTTP links). We're going to use both in our document, but we need an image to use. I downloaded this file locally:

https://www.firstinspires.org/sites/all/themes/first/assets/images/2020/ftc/event-experience.jpg

1. Navigate to the folder you created in the previous step.
2. Create a new folder within your folder called `images`. E.g. `/source/Demo/images`. This can be done by right-clicking on your folder (*Demo*) and selecting "New Folder" and then typing in "images".
3. Add the image file you downloaded to the `images` folder you just created. You can do this by opening the downloaded image file in your file explorer, and then dragging and dropping the file into the `images` folder you just created.

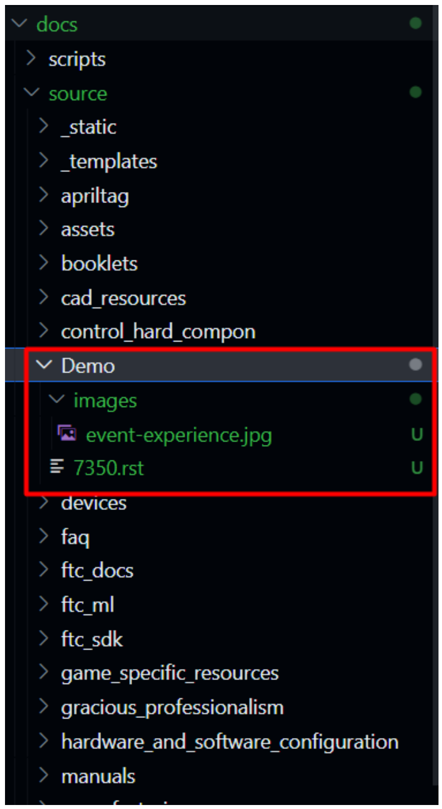


Fig. 21: Completed Demo Folder

Congrats! Now we're ready to add content to our reStructured Text document!

Basic rST Content

This section should be used as a reference for building basic documents using ReStructured Text. When building a document, you generally have the actual content that the text is meant to include; if you ONLY included this content then your document is just one long text document - boooring! In order to add formatting and pizzazz to your document, authors usually add formatting such as titles, links, text formatting, bulleted and numbered lists, and so on. This section introduces the basics of creating text and many of the simple add-ons that are common for web documents.

It's also important to note that all of these basic formatting commands are all supported by the basic preview within GitHub. More complex formatting commands (admonitions, drop-downs, advanced formatting, and so on) are not supported by the basic preview and requires full rendering in order to preview. Those commands will be covered by intermediate and advanced content tutorials.

This tutorial covers these basic building blocks of simple documents (these are quick links within the document):

Okay, so now let's cover these topics one by one.

Simple Table of Contents

The first thing you'll want to do when creating a document is to add or update the Table of Contents (TOC). The TOC is what creates the connections between all the documents in a project. If you are creating a subcategory that resides in the *docs/source/contrib* directory, you will need to add the document to the *toctree* directive in the *docs/source/index.rst* file. Generally speaking, the toctree that you will add the document to is the one that is one layer above the document you are creating.

Adding a document to the TOC is done by simply adding the relative path to the document in the *toctree* directive as shown below:

Before

```
.. toctree::
   :maxdepth: 2
   :caption: Contents:

   previous-doc/source.rst
```

After

```
.. toctree::
   :maxdepth: 2
   :caption: Contents:

   previous-doc/source.rst
   Demo/7350.rst
```

Simple Paragraph Content

ReStructured Text is based on the Python programming language (yes, the name was inspired after reading scripts from "Monty Python's Flying Circus", a comedy series from the 1970's). In Python, text indentation is incredibly important - normal text should start in the first column of the line, and anything that is indented (meaning whitespace precedes the text on a line) has a specific meaning based on what was written on the preceding lines. Blank lines and whitespace is also very important, and rules for these must be followed explicitly. It will initially seem like there are a TON of rules to creating documents - and there are - but once you start making documents yourself hopefully most of the rules will "just make sense" and they will fly out your fingertips without thinking about them.

When writing basic text, there are things to keep in mind:

1. Where text starts is important. Main text should start on the first column of the line, SO DO NOT INDENT the beginnings of paragraphs like your teacher in grade school taught you to do. Anything indented "has special meaning" and introduces special formatting to the text - the indentation rules will be explained later when introducing those formatting elements.

2. Lines and paragraphs do not necessarily end with newlines. All text within a paragraph continues until a completely blank line is reached. What this means is that your paragraph content is allowed to be on as many consecutive lines as you want. For example, the following content:

   ```
   This is a sentence, it is a normal sentence of normal length.
   This is the next sentence, it is shorter.
   This is a third sentence, you can see how the text wraps.

   This is a sentence after a blank line.
   ```

   would be ultimately rendered on a webpage as:

   > This is a sentence, it is a normal sentence of normal length. This is the next sentence, it is shorter. This is a third sentence, you can see how the text wraps.
   >
   > This is a sentence after a blank line, it starts a new paragraph.

   Notice how the lines are combined into a single paragraph ignoring any newlines at the end of each sentence. In order to start a new paragraph, you must separate paragraph content with blank lines (no text).

3. Because of (2), we generally tend to create documents with a "text width" of 80 - meaning lines in the source document are kept to about 80 characters, give or take. It is less important to keep to 80 characters as it is to be consistent within your document. If this doesn't make sense, just ask anyone who has made a few documents and they can explain it - once you understand, you understand.

4. Remember we talked about indentation? Indenting paragraphs requires 3 or more whitespaces of indentation from the last beginning column of the previous paragraph. Let's look at a practical example.

   ```
   This is the first paragraph. It might
   be one or more lines.

      Here's an indented paragraph. It must start 3 or more
      whitespace characters from the last paragraph. Lines can
      continue as long as they are at the same indentation as
      the new paragraph, like these 4 lines.
   ```

(continues on next page)

(continued from previous page)

```
      Here's a second indentation.
      Note that everything is indented nicely.
      Also note a blank line between each paragraph level.

   This is aligned with the first indented paragraph. It is
   at the same indentation. Note blank lines between levels.

This is a second main paragraph. This is at the
same level as the first main paragraph.
```

This will be rendered as:

This is the first paragraph. It might be one or more lines.

Here's an indented paragraph. It must start 3 or more whitespace characters from the last paragraph. Lines can continue as long as they are at the same indentation as the new paragraph, like these 4 lines.

Here's a second indentation. Note that everything is indented nicely. Also note a blank line between each paragraph level.

This is aligned with the first indented paragraph. It is at the same indentation. Note blank lines between levels.

This is a second main paragraph. This is at the same level as the first main paragraph.

Transitions

You might notice the "separator lines" between sections in this document - those are not done automatically, those are special elements knows as "transitions." Transitions are simply four or more punctuation characters between blank lines (meaning there's a blank line before and after the transition). Any punctuation characters can be used, but it's recommended to keep to a consistent style in your documentation. Hence, it's recommended to use dashes (-).

For example:

```
----
```

Creates a line transition separator in a document.

Document Title

When you're editing a file, your document title header should be on the first line of your document. GitHub is really nice in that it provides line numbers for all your document lines. Documents are NOT required to have title headers, however, so when we do use a title header we put special characters underneath the titles to indicate that the text should be formatted as a Title. As a general rule of thumb, we use Equal Signs (=) as our document title indicator. A document title has the following rules:

1. Title Text should be on the first line, and must start in column 1 (no indentation).
2. Special Characters - in this case equal signs (=) - must be used on the next line, and must be as long or longer than the Title Text.
3. There should be a blank line after the special character line.

So below we can see an example of creating a document title. Note that the title text is on the first line, the special characters are below the title and start and extend the full width of the title, and there is a blank line after the special characters. Then we can start a new line to start the actual text of the document (remember, you can click on any image to see the full resolution image):

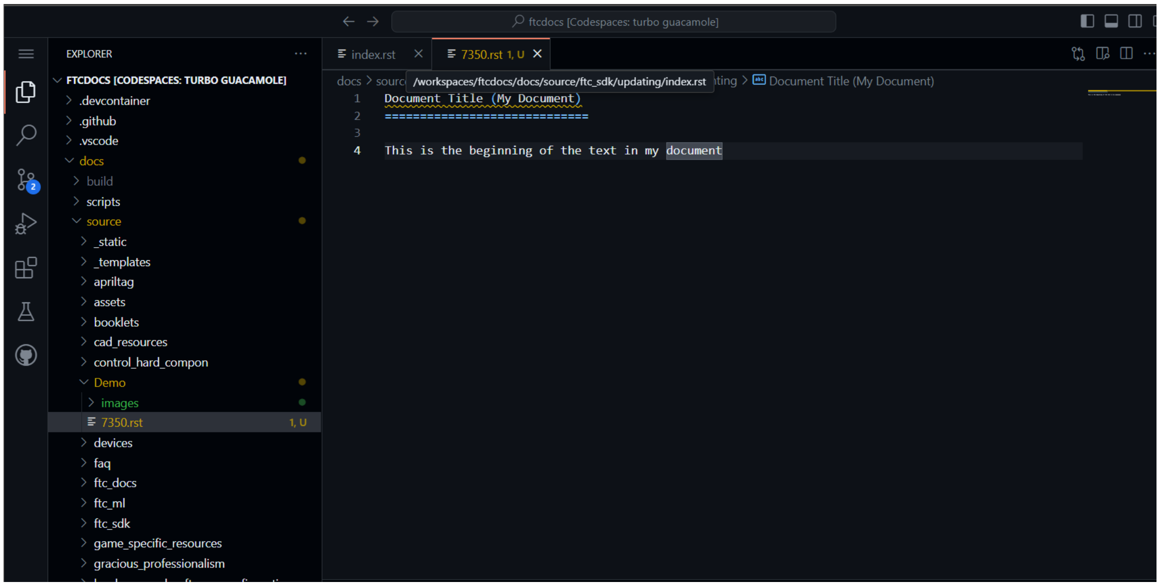


Fig. 22: Example of Editing a Document with a Title

Great! So what will this document ACTUALLY look like on the web or in print? This is where the auto-rendering of ReStructured Text comes in. When you press *ctrl-shift-b* in the VS Code editor, the autobuild will render the document. You only need to press *ctrl-shift-b* once, and all future changes to any documents will automatically render when you save the document. To see what the document will look like, you can navigate *http://127.0.0.1:7350/* in your browser for local users or click the button shown below for codespaces users.

Warning: Remember that inorder to see the rendered document, you must have the document added to the *toctree* directive in the *index.rst* file. If you don't add the document to the *toctree* directive, the document will not be rendered in the preview.

Note: Realize that the special character you use could be almost any special character, but we generally tend to standardize on using an Equal Sign (=) for titles. Once you use a special character in a document to define a heading (like a Title, a Section, a SubSection, and so on) that character will be used to define the heading throughout the document. It's important to be consistent, which is why we have recommended special character progressions.

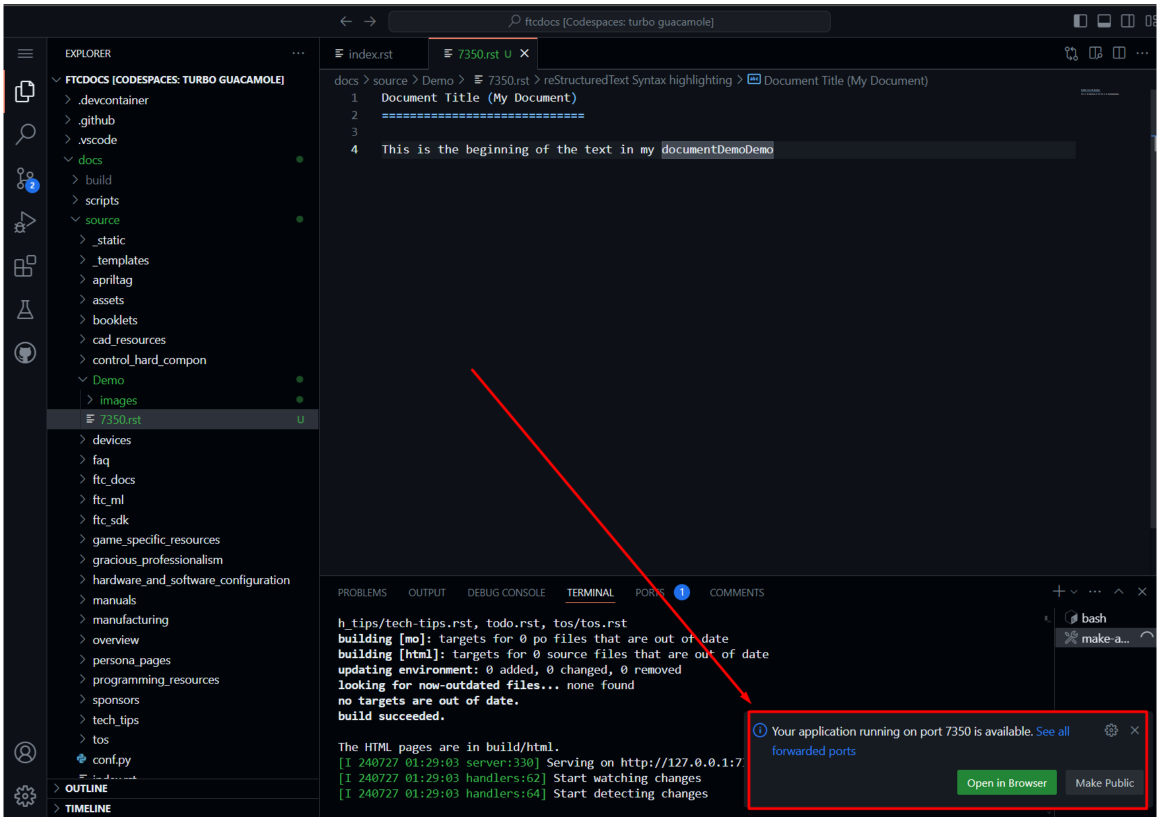


Fig. 23: Example of Previewing a Document with a Title

Section Headers

Section Headers are just like titles; they're actually both related, and are treated the exact same way. Section Headers do follow a progression - titles use the largest heading font size, sections use the next smallest heading font size, subsections use the next smallest heading font size, and so on. This progression is not changeable - as in you cannot "skip" a font size - each new heading type you use will just use the next smallest size font.

In order to create a new Section, SubSection, SubSubSection, and so on, we just use a special character that we will define for each level. The standard special characters used in Python are:

```
Titles
====== (Equals)

Sections
-------- (Dash)

SubSection
^^^^^^^^^^ (Carrot)

SubSubSection
""""""""""""" (Double Quotes)

SubSubSubSection
++++++++++++++++ (Plus Sign)
```

This is what should be used for different levels of sections. Additional special characters that can be used beyond these levels (in case they are needed) are Pound Signs (#) and Asterisks (*). Here's what using Sections looks like:

```
11.3 Titles

11.3.1 Sections

11.3.1.1 SubSection

11.3.1.1.1 SubSubSection

11.3.1.1.1.1 SubSubSubSection
```

The great thing about sections is that each section gets an automatic anchor that can be used to reference that section within the document. Just hover over a section and you'll see a "link" icon show up, and if you click on the "link" icon the URL in the browser will reflect the anchor you can use to direct someone specifically to this section of the document.

> **Warning:** Titles/Sections/SubSections/etc. must all be uniquely named within the same document. In the advanced quickstarts you'll be shown how to reference sections within documents, even within documents between projects, and these require unique headers per document (you can have the same title/header in different documents, just not within the same document).

Text Formatting

You can add simple text formatting - like **Bold**, *italics*, and `literals` really simply in ReStructured Text using simple inline markup. The caveat is that these Text Formatting *do not stack*, meaning you cannot have "Bold Italics" or "Italiczed Literal". You'll find that virtually none of the inline markup styles (Including Text Formatting, External Links, and so on) can stack, so having things like the italicized word *FIRST* in link text requires really inventive and complex procedures in order to make happen (sometimes it's not possible at all). Another caveat is that underlining is not natively supported by ReStructured Text, in order to have underlining you must mess with style sheets and pdf layout definitions in order to do (YUCK!).

The standard Text Formatting Markup is quite simple - use:

- One Asterisk: *text* for emphasis (italics) - like *text*
- Two Asterisks: **text** for strong emphasis (boldface) - like **text**
- Two Backquotes: ``text`` for literals - like `text`

There are a few important restrictions to be aware of:

- You cannot nest/stack inline markup
- Content may not start or end with whitepace: For example, * text* is wrong
- You must separate inline markup from surrounding text by non-word characters, like spaces. For example, *This text is italicized* will look like *This text is italicized*. However, * This text is not* will not render as expected because of the space between the first asterisk and the word or phrase that is expected to follow.
  - One way to avoid this is using a "forced whitespace character", or "" without the quotes (you can tell rST to "force" a character by preceding the character with a backslash (\). For example, we can have Bold and Italics right next to each other without requiring spaces by "injecting" the forced whitespace character that will remain unseen - **Alien**\ *Nation* will be seen as **Alien***Nation*.

It's REALLY nice that the standard GitHub editing window provides some in-line features to show that you're using text formatting - for instance if you bold something, the text will appear bold and the same goes for italicize. This gives you context of what is happening as you do it.

Here are a few examples:

```
This is **bold** text in **this document**.

This is *italic* text in *this document*.

This is ``literal`` text in ``this document``.

This is a fun way using "\\ " to smash styles without
spaces: \*\*Bold\*\*\\ \*Italic\* will render as **Bold**\ *Italic*.
```

This is **bold** text in **this document**.

This is *italic* text in *this document*.

This is `literal` text in `this document`.

This is a fun way using "\ " to smash styles without spaces: **Bold**\ *Italic* will render as **Bold***Italic*.

Internal Links

It is important to note that this is for internal links within the same document. For creating document links between documents, see the :doc: and :ref: commands.

Links with Anchors

Internal Links using Anchors are ways to "jump to" various places within a single document. If you're familiar with HTTP anchors, this follows the exact same concept. You can create an anchor anywhere in text that WILL NOT be displayed to users in the following way:

.. _anchor:

- A blank line must come before the Anchor and after the Anchor.
- This requires two periods at the beginning of the line.
- Followed by a space
- Followed by an underscore
- Followed by a descriptor for the anchor
- Followed by a colon

Note: This is the first time we've introduced what's known as a *directive*, which is a *block* that begins with two periods and a space. Often directives help format special blocks of text in very specific ways - in this case, the underscore prior to the label identifies this as an *anchor directive*. Most often directives have TWO colons following the directive name, but in this case a single colon identifies this as a *simple directive* versus an *external directive*, and helps rST determine how to process the directive.

The name `anchor` can be replaced by any descriptor, like for example: `.. first-example:`

Then, you can create an internal link anywhere in your text that directs to that anchor, in the following way:

anchor_

- This is simply the anchor descriptor
- Followed by an underscore

Therefore, in order to reference the `.. first-example:` anchor, you would use `first-example_` as the link text in your content.

Links with Anchors Example

In plain text, this could look like:

```
This is a link to an anchor_.
This reference can be anywhere before or after the anchor.

.. _anchor:

This is text following the anchor.
The anchor helps to reference this text.
```

And this would be rendered as:

This is a link to an *anchor*. This reference can be anywhere before or after the anchor.

This is text following the anchor. The anchor helps to reference this text.

Referencing Sections

This one is really simple - every title, section, subsection, subsubsection, and so on already has an anchor built-in. The name of the anchor is the name of the section. In order to use a section reference, just wrap the section name in ` (back tick) characters and then follow with an underscore.

Referencing Sections Example

In plain text, a section can be created and referenced like:

```
Section Name
------------

This is a link to `Section Name`_
```

As another real-life example within this document, I can jump to the top of this document really easily by using the name of the document title.

In plain text, this is what it would look like:

```
`Basic rST Content`_
```

And here's how it would be rendered (click the link to jump to the top):

Basic rST Content

Pretty easy!

External Links

External Links provide a way to link to external websites. These are just another form of inline markup, with a few caveats.

Simple External Link

A simple External Link is simply the http address.

- Just put the name of the URL and rST will detect it and provide an auto-link for you.

For example, simply writing:

```
*FIRST* Website: https://www.firstinspires.org
```

creates the rendered text with the link embedded:

FIRST Website: https://www.firstinspires.org

Embedded URI

An embedded URI can provide user-readable text with a link while hiding the actual URL. An embedded URI looks a lot like a Section Reference link, with an added URL component.

An embedded URI follows this example:

- `Description<URL>`_
  - It starts with a ` (back tick) symbol
  - Followed by a description of the URL.
  - Then it defines the URL within angle brackets (< >)
  - Followed by another ` (back tick) symbol
  - Followed by an _ (underscore) character

For example:

```
`Microsoft <https://microsoft.com/en-us>`_
```

would be rendered as:

Microsoft

As another example:

```
`REV Robotics Website <https://www.revrobotics.com>`_
```

would be rendered as:

REV Robotics Website

Embedding External Images

Embedding Images using the image directive

There are multiple ways to embed external images. The most common way is to use the `.. image::` directive. This directive can use a local path within your document project, or it can use an external URI.

Two examples of using the image directive:

```
Including an inline image using a URI
.. image:: https://m.media-amazon.com/images/I/51-2PZby7KL.jpg

Including an inline image using a file path
.. image:: images/myimage.png
```

Directives can include options, which change parameters used with the directive. Options for directives are defined IMMEDIATELY AFTER the directive (on the next line), indented AT LEAST three spaces, with one option on each line. Options on these lines are defined by having a colon before and after the option, followed by the value of the option.

The options supported by the `image` directive are:

alt
: [*text*] Alternate text: a short description of the image, displayed by applications that cannot display images, or spoken by applications for visually impaired users

height
: [*length*] The desired height of the image. Used to reserve space or scale the image vertically. When the "scale" option is also specified, they are combined. For example, a height of 200px and a scale of 50 is equivalent to a height of 100px with no scale.

scale
: [*integer percentage (the "%" symbol is optional)*] The uniform scaling factor of the image. The default is "100 %", i.e. no scaling.

width
: [*length or percentage of the current line width*] The width of the image. Used to reserve space or scale the image horizontally. As with "height" above, when the "scale" option is also specified, they are combined. It is often preferable to use *width* over *height* or *scale*.

align
: ["top", "middle", "bottom", "left", "center", or "right"] The alignment of the image, equivalent to the HTML <img> tag's deprecated "align" attribute or the corresponding "vertical-align" and "text-align" CSS properties. The values "top", "middle", and "bottom" control an image's vertical alignment (relative to the text baseline); they are only useful for inline images (substitutions). The values "left", "center", and "right" control an image's horizontal alignment, allowing the image to float and have the text flow around it. The specific behavior depends upon the browser or rendering software used.

target
: [*text (URI or reference name)*] Makes the image into a hyperlink reference ("clickable"). The option argument may be a URI (relative or absolute), or a reference name with underscore suffix (e.g. \`a name\`_).

Examples of using these options:

```
.. image:: https://m.media-amazon.com/images/I/51-2PZby7KL.jpg
   :width: 80%
   :alt: This is alternate text for the image

.. image:: images/picture.jpeg
   :height: 100px
   :width: 200 px
   :scale: 50 %
```

(continues on next page)

(continued from previous page)

```
   :alt: alternate text
   :align: right
```

It is important to remember that directives must have a blank line before the directive and must have a blank line after the directive (and all its options).

> **Warning:** Images with the extension `.gif` and `.svg` are not supported in PDF format. For documentation that will be used in PDFs, do not externally link to files with these extensions. It is possible to use these files when using a file path, as long as supported versions of the files exist. For example, if you have both `picture.svg` and `picture.png`, you can command the HTML to use one version and the PDF to use *any supported version* through using a * (asterisk) in the file extension, like so:
>
> ```
> .. image:: images/picture.*
> ```

Embedding Images using the figure directive

The `.. figure::` directive is very similar to the `.. image::` directive, as a matter of fact the `figure` directive *contains* an `image` directive but also allows for an optional caption (a single paragraph) and an optional legend (with arbitrary body elements).

The `figure` directive supports all of the options of the `image`. These options, except `:align:`, are passed on to the contained image. The following options are important for the `figure`:

align
: [*"left", "center", or "right"*] The horizontal alignment of the figure, allowing the image to float and have the text flow around it. The specific behavior depends upon the browser or rendering software used.

Figures are probably the best way of showing images as they allow captions to help describe and label images. Some examples of using figures are:

```
.. figure:: images/picture.png
   :width: 80%
   :alt: Map to buried Treasure

   This is the caption of the figure (a simple paragraph). Note that the
   intentation for everything below the ``.. figure::`` line is the same
   and 3 or more spaces, which indicates that everything belongs to the
   figure.

.. figure:: https://m.media-amazon.com/images/I/51-2PZby7KL.jpg
   :width: 80%
   :alt: Alternate Text

   Simple Caption for Figure
```

You can see more about the `figure` directive at the Figure Directives Link.

Lists

There are five kinds of lists:

- *Bulleted Lists*
- *Enumerated Lists*
- *Definition Lists*
- Field Lists
- Option Lists

We'll discuss the first three only - for Field and Option lists, click the links to go to the specifications for those types of lists.

Bulleted Lists

Bulleted lists, also known as "unordered lists", are simple text blocks between blank lines. A text block that begins with an asterisk (*), plus (+), dash (-), or bullet (•), followed by whitespace, is a bullet list item. However, stick to a consistent character to use for list items. List item bodies must be left-aligned and indented relative to each other like paragraphs; the text immediately after the bullet determines the indentation.

For example, consider the following simple bulleted list example:

```
- This is the first bullet list item. It is required that
  there be a blank line above the first list item; blank
  lines between list items is optional. Note that each
  subsequent line is indented to group them together.

- This is the first paragraph in the second item in the list.

  This is a second paragraph in the second item in the list. The
  blank line above this paragraph is required. The left edge
  of this paragraph lines up with the paragraph above, both
  indented relative to the bullet.

  - This is a sub-list. The bullet lines up with the left edge
    of the text blocks above. A sublist is a new list, so it requires
    a blank line above and below.

  - This is the second item in the sub-list.

- This is the third item in themain list.

This is a new paragraph, not part of the list.
```

Here are examples of **incorrectly** formatted bullet lists:

```
- This first line is fine.
A blank line is required between list items and paragraphs, so this is bad.

- The following line appears to be a new sublist, but it is not:
  - This is a paragraph continuation, not a sublist (since there's
    no blank line).
   This line is also incorrectly indented.
```

Enumerated Lists

Enumerated lists are similar to bulleted lists, except they can use enumerators. An enumerator consists of an enumeration sequence member and formatting, followed by a whitespace.

The following enumeration sequences are recognized:

- Arabic numerals: 1, 2, 3, … (and so on, no upper limit).
- Uppercase alphabet characters: A, B, C, …, Z.
- Lowercase alphabet characters: a, b, c, …, z.
- Uppercase Roman numerals: I, II, III, IV, …, MMMMCMXCIX (4999).
- Lowercase Roman numerals: i, ii, iii, iv, …, mmmmcmxcix (4999).
- Hashtag (#) - this is known as an auto-enumerator, and uses arabic numerals beginning with 1.

The following formatting is recognized:

- Suffixed with a period: "1.", "A.", "a.", "I.", "i.", etc…
- Surrounded by parenthesis: "(1)", "(A)", "(a)", and so on.
- Suffixed with a right-parenthesis: "1)", "A)", "a)", and so on.

The following situations creates new lists:

- An enumerator with a different format produces a new list (e.g. "1.", "(a)" produces two separate lists).
- Enumerators not in sequence produces a new list (e.g. "1.", "3." produces two separate lists)

Here is an example of an enumerated list:

```
1. This is item #1
2. This is item #2

   (a) This is item 1 in the sub-list
   (b) This is item 2 in the sub-list

3. This is item #3.
```

Here is the same list using an auto-enumerator:

```
#. This is item #1
#. This is item #2
#. Auto-enumerators are useful when adding things
   into lists without having to manually re-number
   or re-order lists.

   (a) This is item 1 in the sub-list

#. This is the fourth item in the main list.
```

Here is an example of a nested enumerated list:

```
1. This is Item 1

   a) Item 1a
   b) Item 1b

2. a) Item 2a
   b) Item 2b
```

(continues on next page)

(continued from previous page)

```
3. This item won't be correct, because
this line after is not indented properly
```

Definition Lists

Definition lists are really useful in several ways:

- As a dictionary or glossary.
- To describe program variables, or other items

Each definition list item contains a term, optional classifiers, and a definition.

- A *term* is a simple one-line word or phrase. If this *term* leads with a hyphen, use an escape () character before the leading hyphen to prevent recognition as an option list item.
  - Optional classifiers may follow the term on the same line, each after an inline " : " (space, colon, space). Inline markup is parsed in the term line before the classifier delimiters are recognized. A delimiter will only be recognized if it appears outside of any inline markup.
- A *definition* is a block indented relative to the term, and may contain multiple paragraphs and other body elements. There may be no blank line between a term line and a *definition* block (this distinguishes *definition* lists from block quotes). Blank lines are required before the first and after the last *definition* list item, but are optional in-between.

Example:

```
term 1
    Definition for Term 1
term 2
    Definition for Term 2, first paragraph.
    This is a continued line for the first paragraph.

    Second paragraph for definition of term 2.
Term 3 : classifier
    Definition for term 3.
Term 4 : classifier one : classifier 2
    Definition for Term 4
\-term 5
    Without escaping, this would be an option list item.
```

Once rendered, this looks like:

term 1
Definition for Term 1

term 2
Definition for Term 2, first paragraph. This is a continued line for the first paragraph.

Second paragraph for definition of term 2.

Term 3
[classifier] Definition for term 3.

Term 4
[classifier one][classifier 2] Definition for Term 4

-term 5
Without escaping, this would be an option list item.

30.4.12 Setup Git Credentials

One Time Only Local

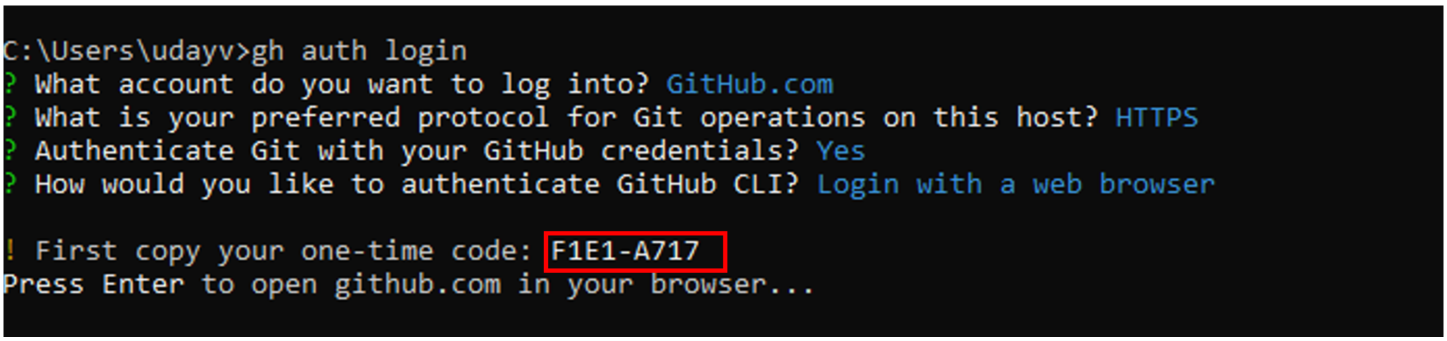
Note: This is a one-time setup that is only necessary for local development. Codespace users may skip this step.

This step enables you to push changes to your forked repository. It is necessary inorder for GitHub to authenticate you as an authorized user.

1. Install GitHub CLI
2. Enter the following command in your terminal to authenticate with GitHub:

```
gh auth login
```

3. Follow the prompts to authenticate with GitHub.



30.4.13 Submitting Your Changes for Review

Repeat Codespaces Local

Workflows

The following diagram shows the workflow for submitting changes to the FTC Docs repository. For first time users of Git/GitHub, do not worry if this seems confusing. First focus on understanding the steps and then the workflow will make more sense.

Local changes start in your local copy of the project. They move from the working directory to local Git staging and are then committed to the local repository. From there they are pushed to your personal FTC Docs repository.

Codespaces are part of your personal FTC Docs GitHub project. In a similar manner to local changes, Codespace changes are staged and committed to your personal FTC Docs repository. Once any changes are in your personal FTC Docs repository they can be submitted as a Pull Request to the FTC Docs repository.

In the FTC Docs repository changes arrive as Pull Requests that are reviewed and approved. They are then merged with the Main branch and the changes are published.

In a web browser this diagram can be zoomed and panned by using a mouse. Use the scroll wheel to zoom in and out. Right click and hold then drag to pan. The diagram is not keyboard accessible. A screen reader will read the various nodes and actions in the diagram and starts in the Local section of the diagram.

Overview

Before getting started, it is important to understand what Git tries to accomplish. As Git is an incredibly powerful(and complex) tool, this can be difficult and overwhelming. This tutorial will attempt to simplify this by focusing on the most common use cases.

What is Git?

Simplified Local Git Repository Workflow

Git is a version control system (VCS) that allows you to track changes to your files within a repository. A repository is a collection of files that are being tracked by Git. You can think of a repository as a folder that contains all of the files that you are working on.

However, Git does not track every change you make to a file. This is because it would be inefficient to track every change and often distracting. Instead Git tracks changes in snapshots called commits. Each commit is a snapshot of the changes made to the files in the repository. A commit does not contain the entire file but only the changes made to the file. This allows Git to be efficient and fast. You can then think of each commit as a "Git save".

Before you can commit your changes, we must indicate which files we want to be updated in the commit. This is done by a process known as staging. Why don't we just commit all of the changes? Sometimes you may have changes that you do not want to commit. For example, maybe you deleted a file that you did not mean to delete. In addition, you may not want to commit build files or other temporary files that are not necessary for the repository. Note that we have configured Git to ignore build files so you do not have to worry about them.

After you have staged and committed your changes, you can push them to the remote repository. This is the repository that you see on GitHub. This allows others to see your changes and collaborate with you. You have full control over what changes you want to push to the your *fork* of the *main repository*. In order for your changes to be reflected in the main FTC Docs website you will need to add your changes to the main repository. This is done by creating a *pull request*.

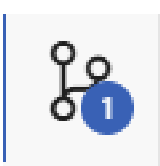
Steps

Note: All of the following commands are typed and executed in the terminal. This can be found on the bottom of the screen in VS Code.

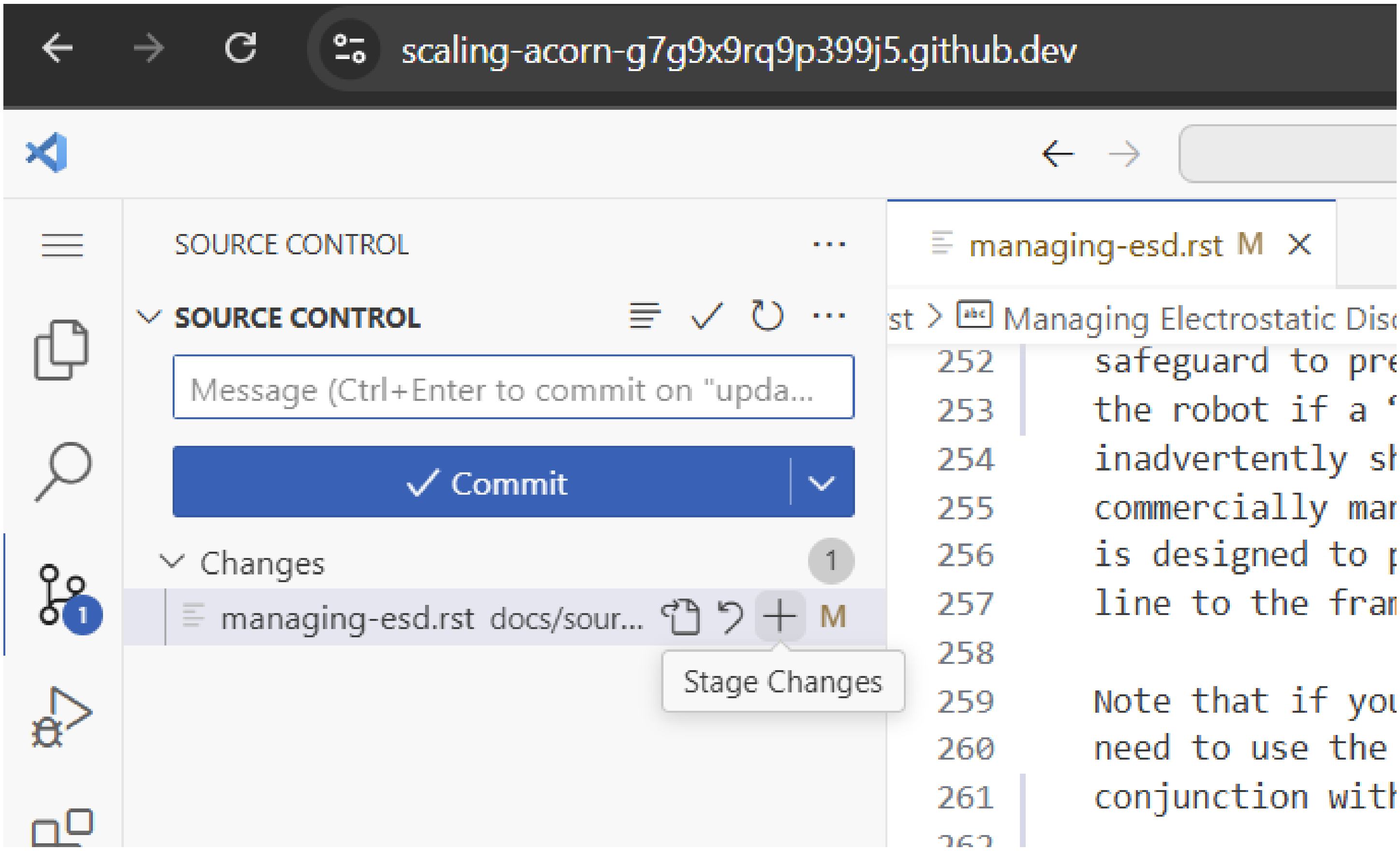
Staging Your Changes

As a reminder, staging is the process of indicating which files you want to be included in the next commit ("Git Save"). This is done by using the Local command `git add <file>`. You can add multiple files by separating them with a space.

In Codespaces click on the source control icon which will display the source control panel.



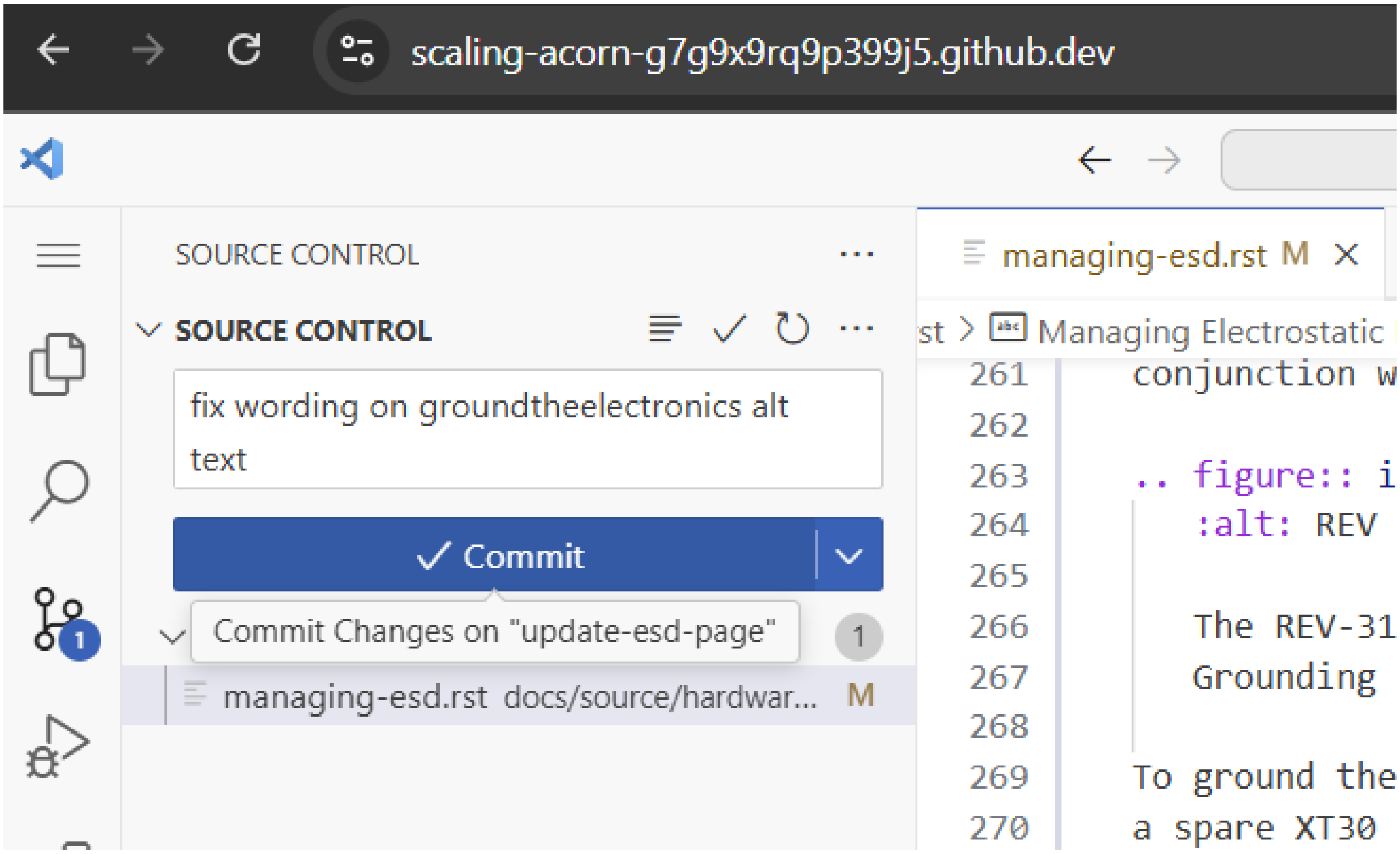
Click on the + symbol next to each changed or new file in the source control panel to stage that file.



Committing Your Changes

Once you have staged your changes, you can commit them. This is done by using the Local command `git commit -m "Your commit message"`. You can think of a commit as a snapshot of your changes. Each repository is a collection of commits each describing incremental changes relative to the previous commit.

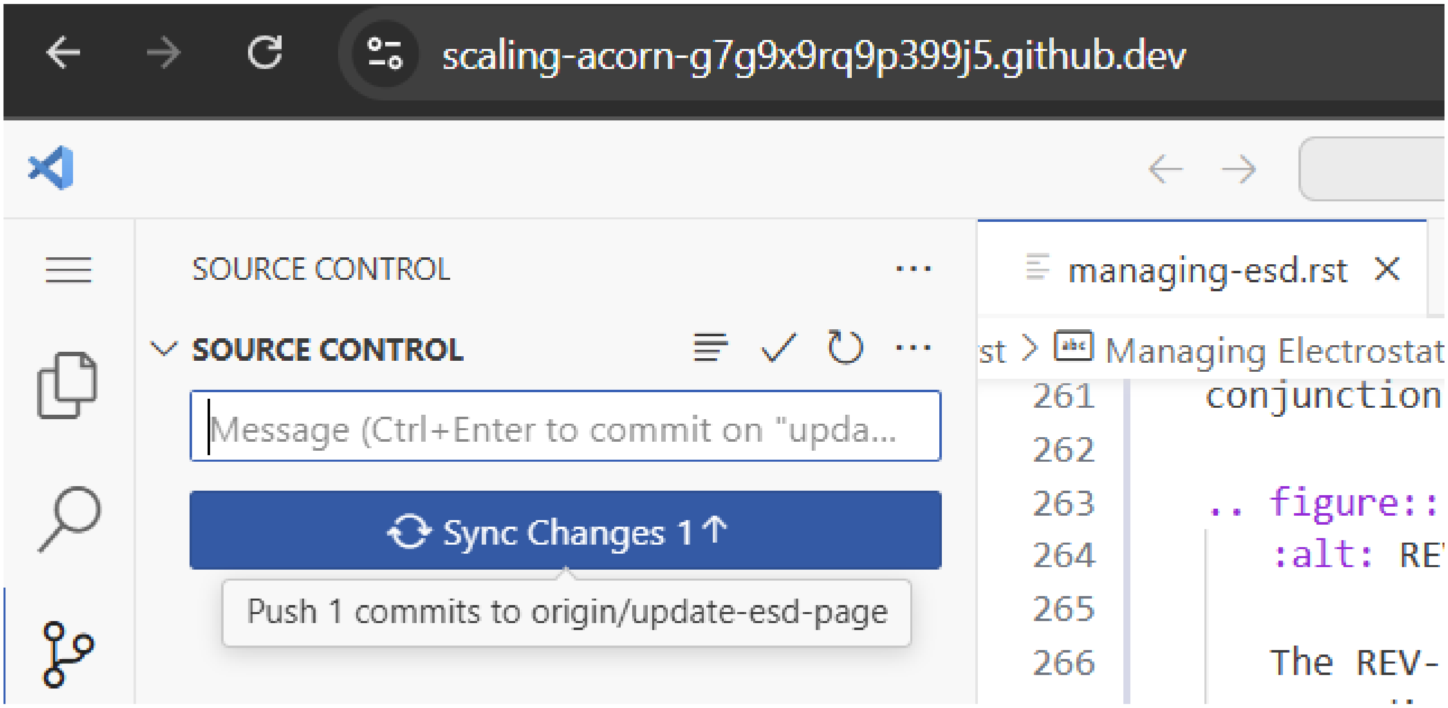
In Codespaces enter a commit message in the source control panel and click the Commit button.



Pushing Your Changes

Once you have committed your changes, you can push them to your fork of the repository. This is done by using the Local command `git push origin <branch>`. This will push your local changes to the remote repository. This means it will be accessible to others. After this change is pushed, you can create a pull request.

In Codespaces click the Sync Changes button.



Creating a Pull Request

Now that you have pushed your changes to your *fork* of the repository the next step is propose these changes to the *main repository*. Why don't we just push our changes to the main repository? This is because the pull request allows the maintainers of the main repository to review your changes before they are merged. This is important as it allows comments and feedback to be given on your changes. For details on how to create a pull request, see the next section.

Example

The following is a Local example of all the Git commands. The pull request commands are done in GitHub starting at step 7.

If you are using Codespaces just skip to step 7.

1. Navigate to the directory of the repository on your local machine.

```
cd docs/source
```

2. Ensure that you are on the correct branch. In this case we are on the branch "demo". If you are not on the correct branch, follow the *previous step*.

```
git branch
```

3. Stage your changes. In this case we need to stage the new file, the images we use in it, as well as the changes to the table of contents.

```
git add demo_section/demo-section.rst
git add demo_section/images/demo.png
git add index.rst
```

4. For the **first time** you commit changes, you will need to set your name and email.

```
git config --global user.email "<INSERT YOUR EMAIL HERE>"
git config --global user.name "<INSERT YOUR NAME HERE>"
```

5. Commit your changes. In this case we are adding a new section called “Demo” so we will use the commit message “Add demo section”.

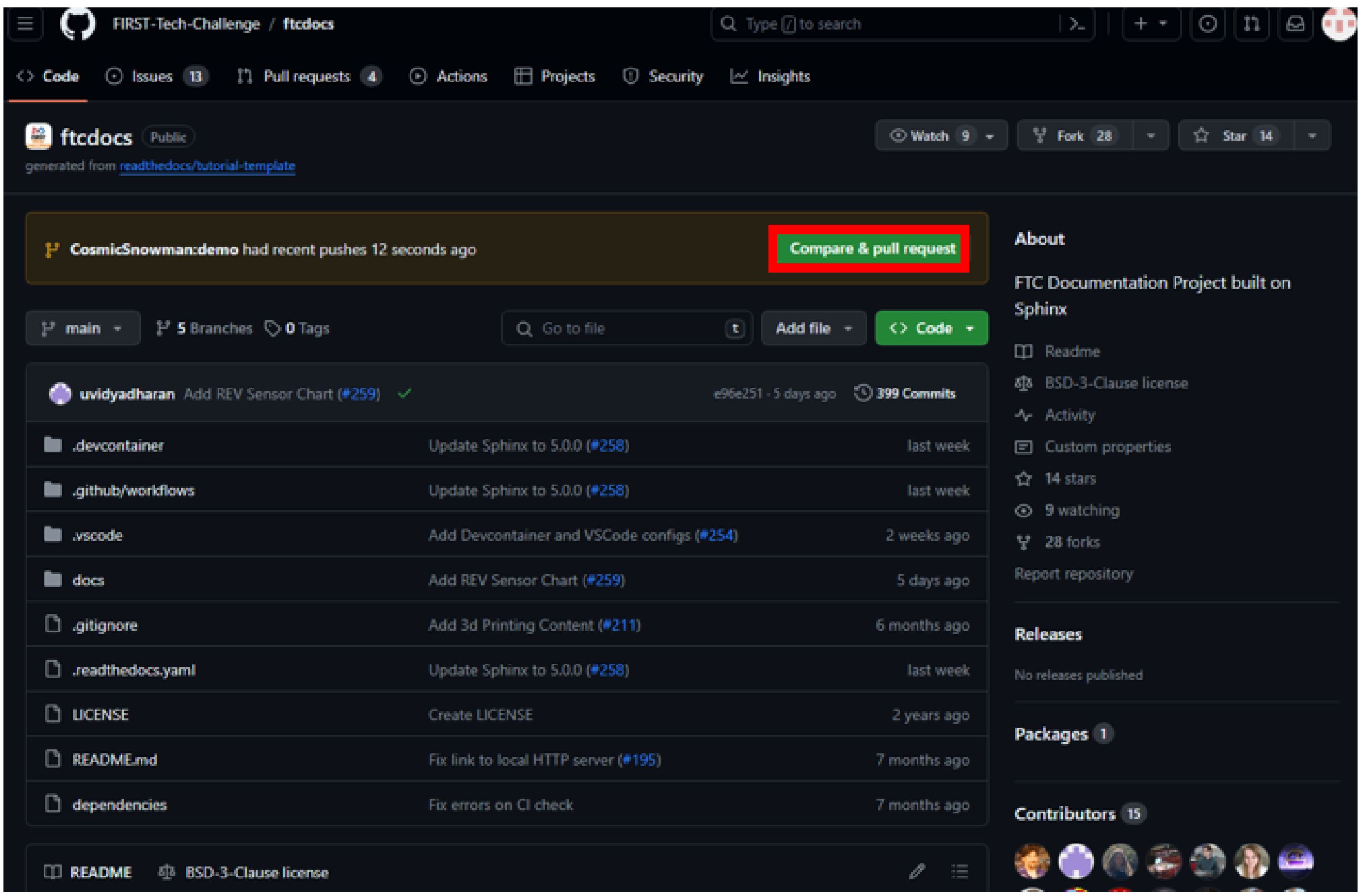
```
git commit -m "Add demo section"
```

6. Push your changes to your fork of the repository.

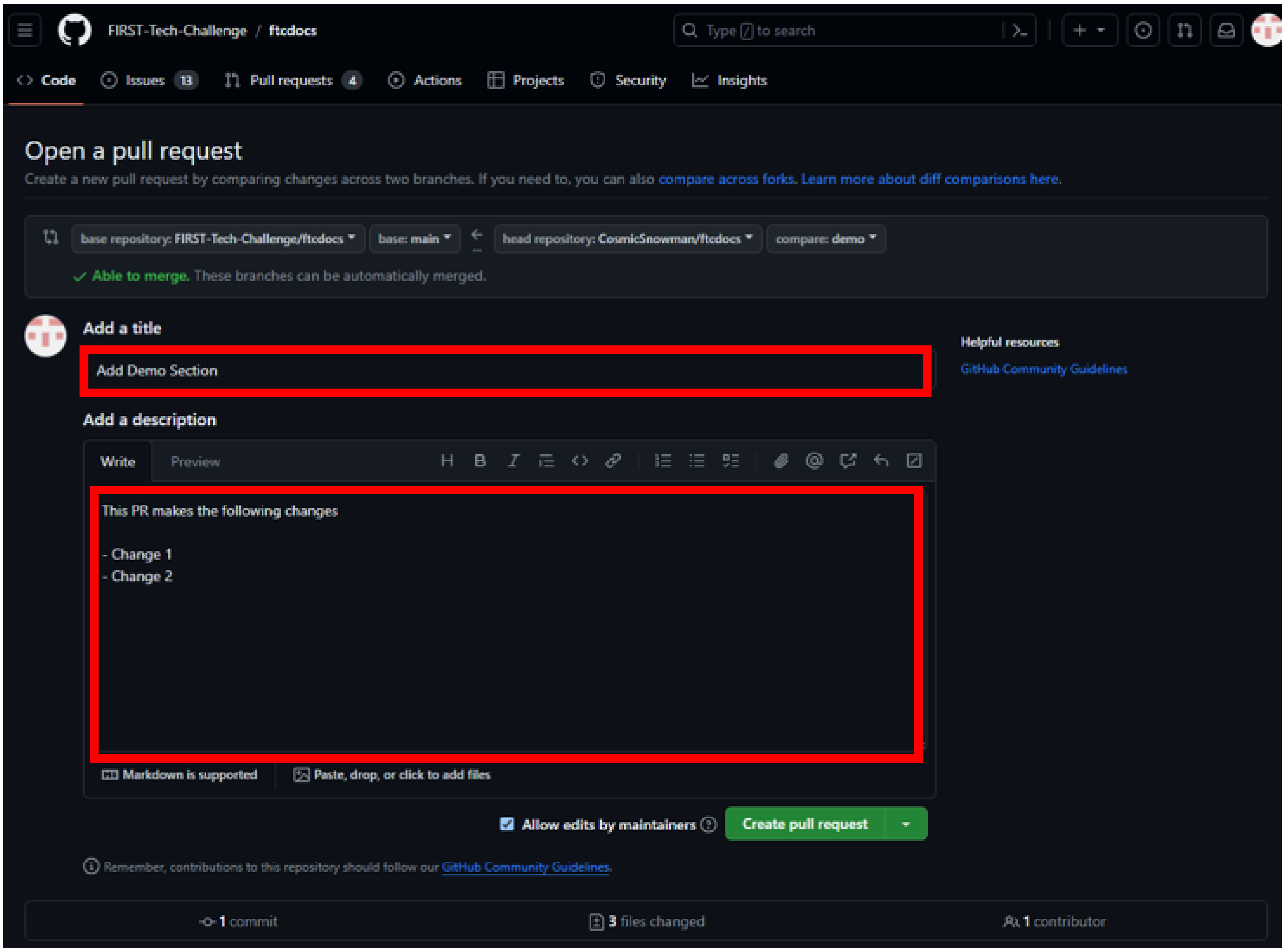
```
git push
```

7. Navigate to the official FTC Docs repository on GitHub.
8. Click `Compare & Pull Request`.

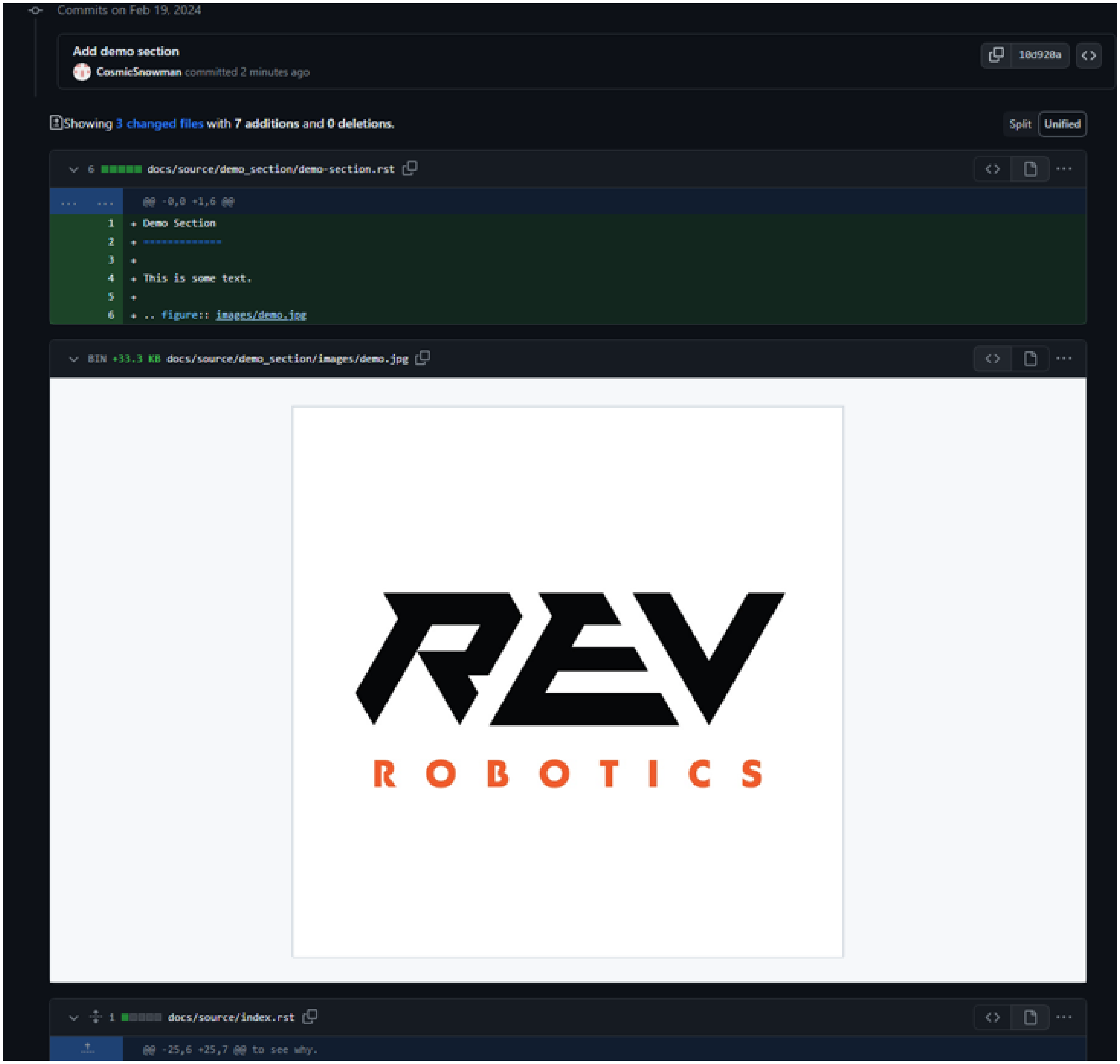
Note: This option is also available by clicking the “Pull Request” tab and then clicking the “New pull request” button. Then click `compare across forks`. Select `` `<USERNAME>/ftcdocs` `` for the head repository and the branch you want to merge. Select *FIRST-Tech-Challenge/ftcdocs* for the base repository and the branch you want to merge into. Then click `Create Pull Request`.



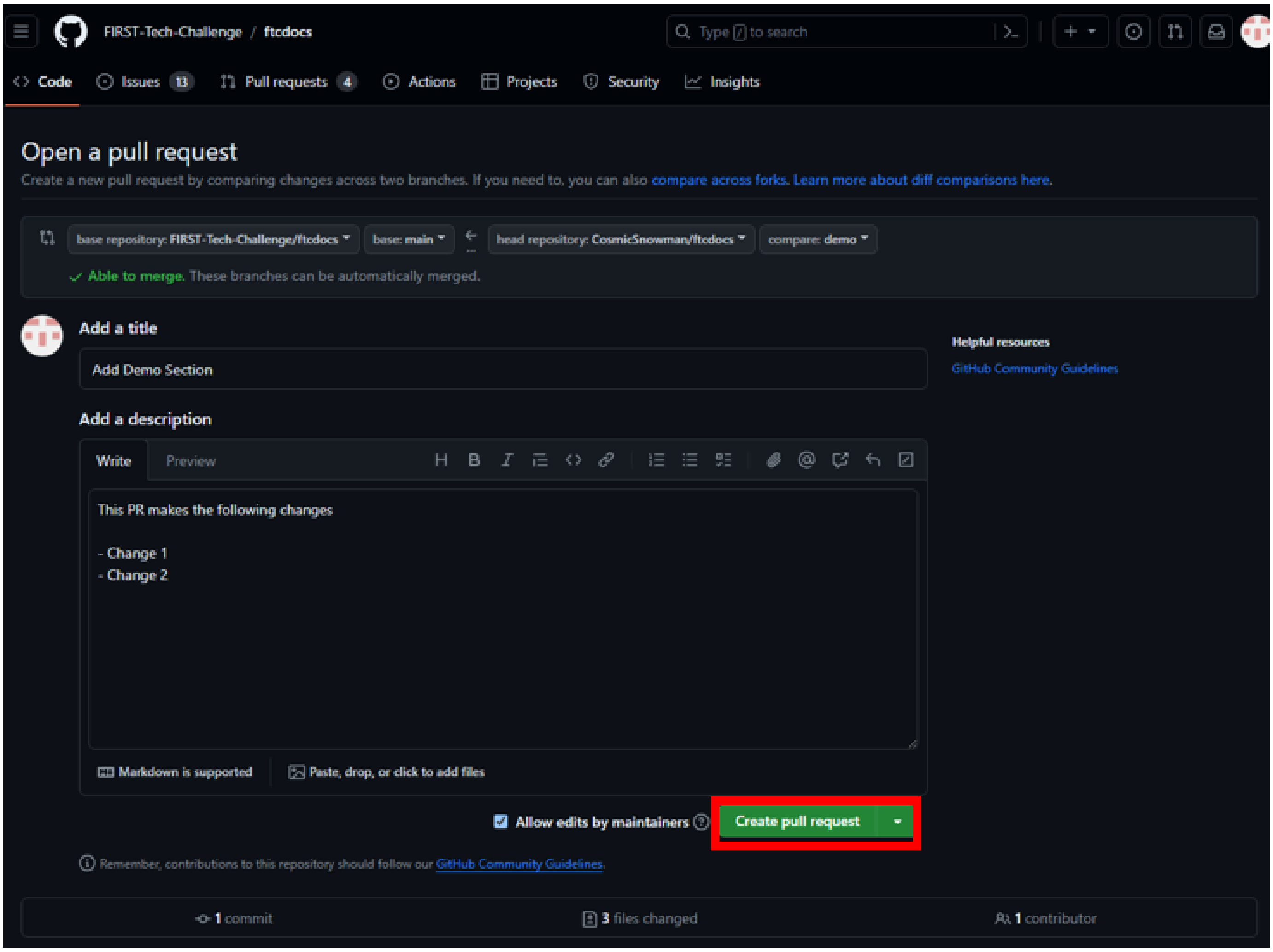
9. Fill out the pull request title and description. For more information on creating a pull request, see our *Contribution Guide*.



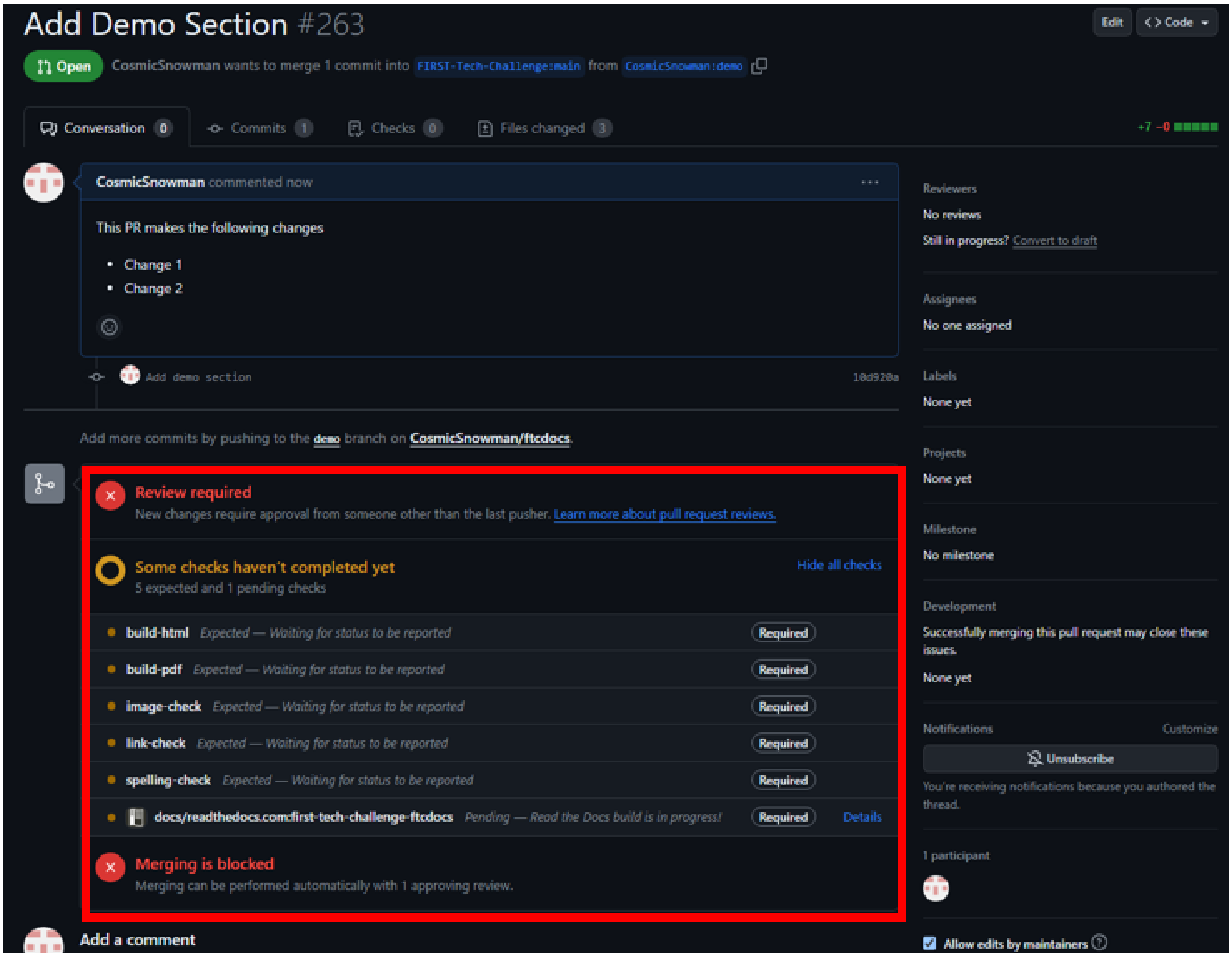
10. Scroll down to see a preview of the changes you are proposing. Make sure that everything looks correct and that no files or changes have been omitted. Also make sure no erroneous changes are included.



11. If everything looks good, click the `Create Pull Request` button.



12. After this you will be able to see your pull request and the status of the automated checks. First time contributors will have to wait for a maintainer approval before the checks are run.



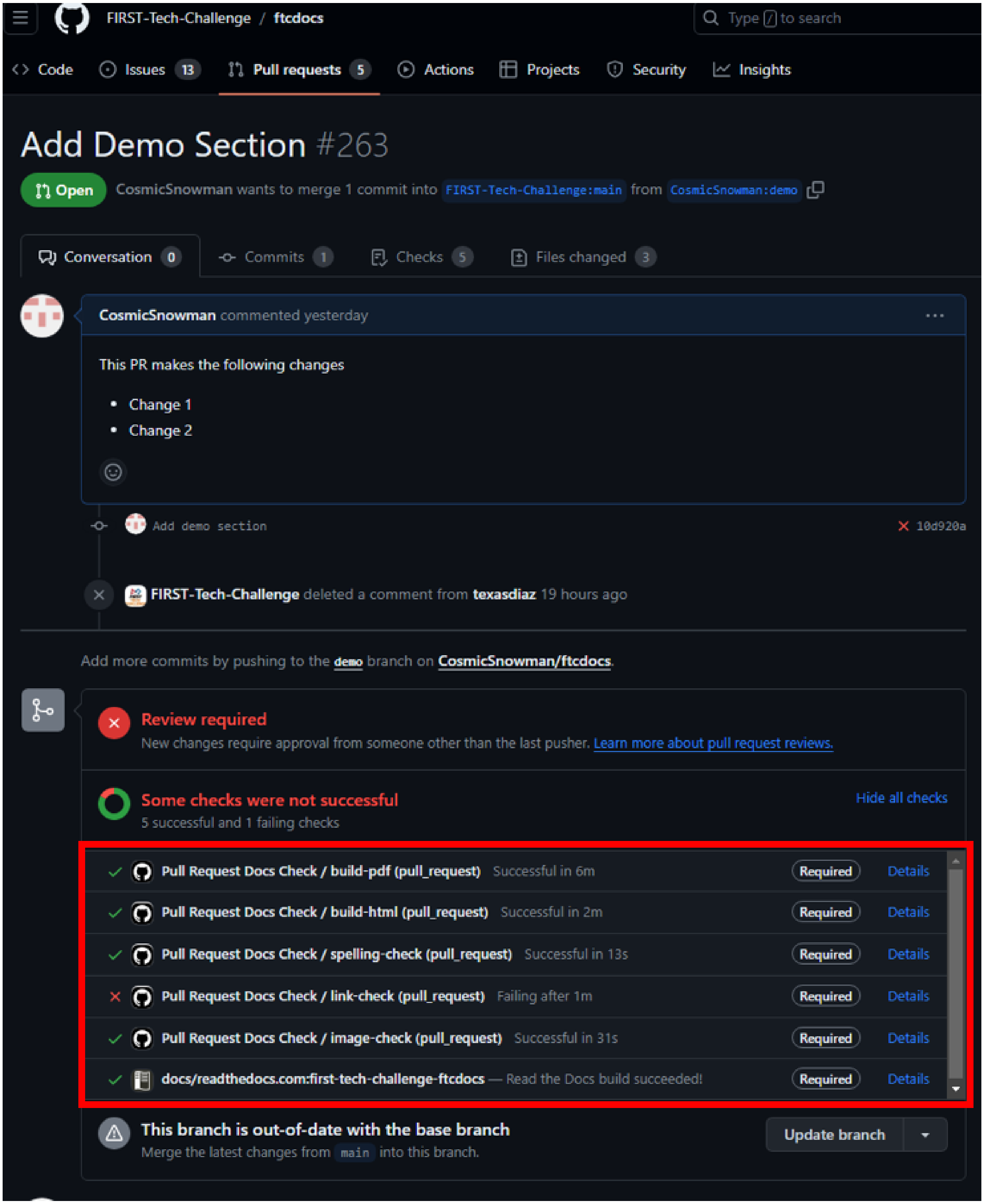
13. After approval the checks will run. In this case the Link Checker failed. When checks fail the FTC Docs maintainers will be notified and will help you fix the issue. Generally the FTC Docs team will help you with the process of passing the checks and updating your branch.

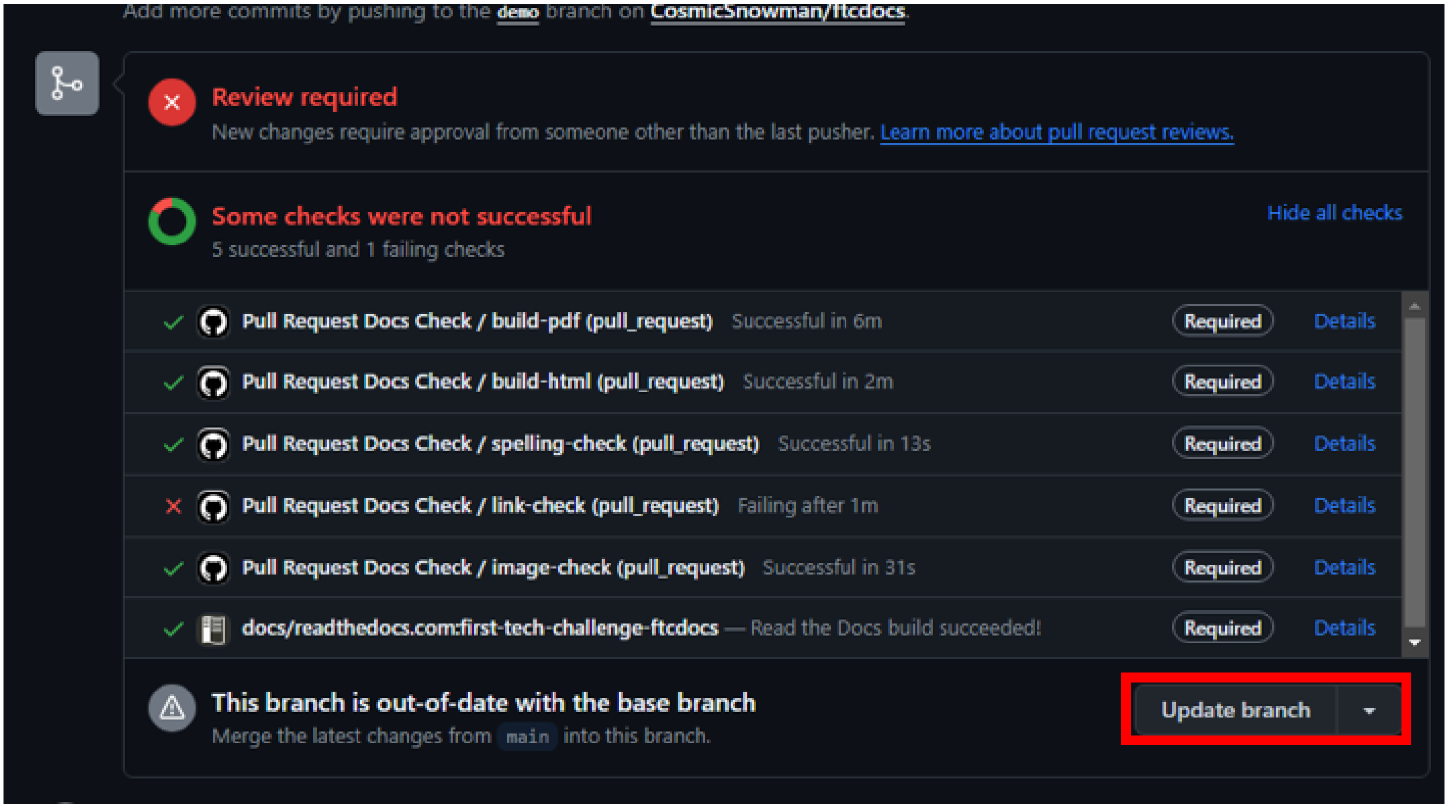
    Another problem was because our branch is no longer up to date with the main branch. This can easily be fixed by clicking the “Update branch” button.

    Note: It can take several minutes for the checks to fully run.

    ---

14. Once the checks have passed, the FTC Docs maintainers will review your pull request. They will provide feedback and help you make any necessary changes. Once the pull request is approved, it will be merged into the main branch.


FIRST-Tech-Challenge / ftcdocs
Type / to search
Code
Issues 13
Pull requests 5
Actions
Projects
Security
Insights
Add Demo Section #263
Open
CosmicSnowman wants to merge 1 commit into FIRST-Tech-Challenge:main from CosmicSnowman:demo
Conversation 0
Commits 1
Checks 5
Files changed 3
CosmicSnowman commented yesterday
This PR makes the following changes
Change 1
Change 2
Add demo section
10d920a
FIRST-Tech-Challenge deleted a comment from texasdiaz 19 hours ago
Add more commits by pushing to the demo branch on CosmicSnowman/ftcdocs.
Review required
New changes require approval from someone other than the last pusher. Learn more about pull request reviews.
Some checks were not successful
5 successful and 1 failing checks
Hide all checks
Pull Request Docs Check / build-pdf (pull_request) Successful in 6m
Required
Details
Pull Request Docs Check / build-html (pull_request) Successful in 2m
Required
Details
Pull Request Docs Check / spelling-check (pull_request) Successful in 13s
Required
Details
Pull Request Docs Check / link-check (pull_request) Failing after 1m
Required
Details
Pull Request Docs Check / image-check (pull_request) Successful in 31s
Required
Details
docs/readthedocs.com:first-tech-challenge-ftcdocs — Read the Docs build succeeded!
Required
Details
This branch is out-of-date with the base branch
Merge the latest changes from main into this branch.
Update branch



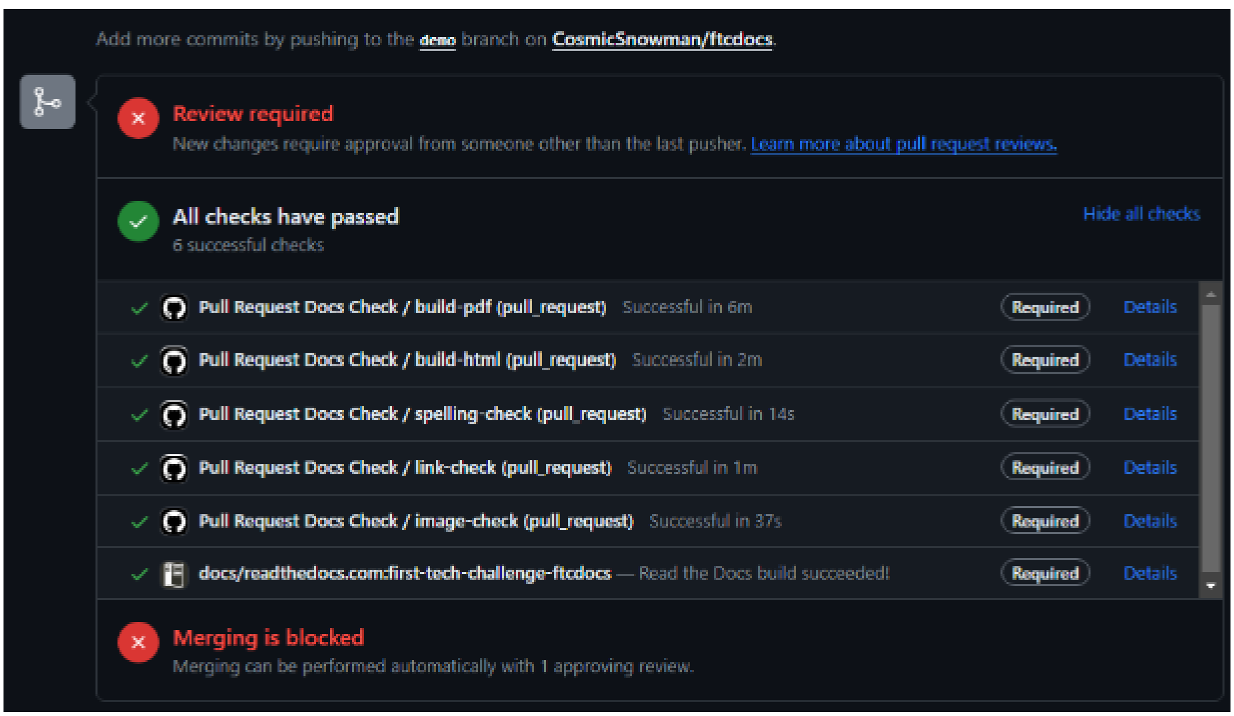


Fig. 24: This pull request still has a Review Required and Merging is Blocked until there is an approving review.

30.4.14 FTC Docs Contributor Glossary

Branch
A branch is a parallel version of a repository. It is contained within the repository, but does not affect the primary or master branch, allowing you to work freely without disrupting the "live" version. When you've made the changes you want to make, you can merge your branch back into the master branch to publish your changes.

Feature Branch
A feature branch is a branch that is created to work on a new feature or bug. When you create a feature branch, you are making a copy of the main branch and working on the feature in isolation. Once the feature is complete, you can merge the feature branch back into the main branch.

Fork
A fork is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project. Most commonly, forks are used to either propose changes to someone else's project or to use someone else's project as a starting point for your own idea.

Main
Main or main branch refers to the default branch of a repository. This is the branch that is checked out when you clone a repository. By convention, changes are made on feature branches and then merged into the main branch when they are ready to be published.

Main Repository
The main repository is the GitHub repository found at github.com/FIRST-Tech-Challenge/ftcdocs. The files in this repository are the official documentation for the FIRST Tech Challenge and the ones that are built and deployed to the FTC Docs website. This should not be confused with the main branch of a repository, which is the default branch of a repository.

Pull Request
A pull request is a way to propose changes to a branch in a repository. When you submit a pull request, you're requesting that the repository owner pull in your changes. Pull requests show differences between the content of the source branch and the target branch. The changes, additions, and subtractions are shown in green, red, and grey, respectively.

Repository
A repository is a central location in which data is stored and managed. It can be a collection of files or directories, and it can be a place where developers store their code, track changes, and collaborate with others.

30.5 Mission Statement

FTC Docs aims to provide a comprehensive documentation base for *FIRST* Tech Challenge teams and mentors. It is a community-driven project, hosted and moderated by FIRST Tech Challenge staff, and we welcome contributions from all teams and mentors. It is our hope that this project will help to make the community more connected and informed while reducing the fragmentation of documentation present in *FIRST* Tech Challenge. The information provided is not meant to endorse any specific method or approach, but rather to provide an information base for students and mentors to make informed decisions.

- *Contribution Guidelines*
- *Style Guide*
- *FTC Docs Workflow*
- *Tutorials*

FTC Docs is brought to you by:

- Daniel Alfredo Diaz, Jr — Maintainer.
- Elizabeth Gilibert — Project Manager

- Uday Vidyadharan, 7350 Watt's NXT? — Maintainer
- Chris Johannesen, Westside Robotics — Contributor
- Mike Silversides, BC FTA — Contributor
- Miriam Sinton-Remes, 7182 Mechanical Paradox — Contributor

A special thanks to everyone who has contributed to this project.

I am a…

- *New Team* New Teams may not know where to start. This is the way!
- *Returning Team* Returning Teams looking for resources can look here.
- *Coach* Coaches looking for help or Team Administrative Resources can look here.
- *Mentor* Technical Mentors looking for Technical Resources should look here first!

The main menu contains links to the top level content. The following are quick links organized by topic.

Programming Links

Quick Links for Programming Language Resources

Blocks

OnBot Java

Android Studio

AprilTags

All Resources

Control System Links

Let's get to know the *FIRST* Tech Challenge Control System!

Driver Station

Robot Controller

Device Connections

Hardware Configuration

Software Development Kit (SDK)

The Software Development Kit (SDK) is the collection of tools for developing software and executing it on the robot.

About the SDK

SDK GitHub Repository

SDK Releases

Javadoc Documentation

Game Links

Be sure you're following all of the rules of the competition! The Competition Manual is an essential document.

Competition Manual

Playing Field Resources

Game Question and Answer System

Note: This project is under active development. Anything contained herein is for informational purposes only; while this documentation is intended to support teams and in some way provide context to game rules, the game rules supercede all documentation found here. If you have feedback about this project, please use our *feedback form*.

Chapter 31

Version Information

Author: FIRST Tech Challenge
Version: 0.3
Release Date: 09/06/2025
Generation Time: 21:05
License: BSD 3-Clause
Git Hash: 48e66b7e2940102fd5217e60e2785bf881f5eaa6

Index