



FIRST® IN SHOW™  
presented by Qualcomm

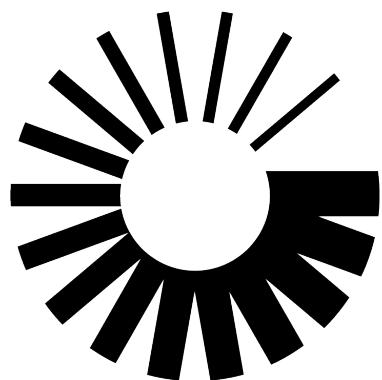
[firstinspires.org/robotics/ftc](http://firstinspires.org/robotics/ftc)

2023-2024 *FIRST* Tech Challenge

# FTC Programming Resources

***FIRST* Tech Challenge Documentation**

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



**RTX**

---

## Tutorials

1	Programming Tutorials	2
2	Supporting Documentation	297
3	AprilTag Programming	346
4	TensorFlow Programming	416
5	Vision Programming	471
6	Advanced Topics	507
7	Additional <i>FIRST</i> Website Resources	607
8	Version Information	608



## Chapter 1

---

### Programming Tutorials

FIRST Tech Challenge Programming Tutorials

- [Choosing a Programming Tool](#)
- [Blocks Tutorial](#)
- [Onbot Java Tutorial](#)
- [Android Studio Tutorial](#)

## 1.1 Choosing a Programming Tool

You will need to select a programming tool to be able to write op modes for your competition robot. FIRST strongly recommends that *all* users begin by learning how to use the [Blocks programming tool](#).

There are currently three programming tools that are available for the teams to use:

1. **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 screenshot shows the OnBot Java Programming Tool interface. At the top, there are tabs for 'Blocks' (selected), 'OnBotJava', and 'Manage'. Below the tabs are buttons for 'Save Op Mode', 'Export to Java', 'Download Op Mode', and 'Download Image of Blocks'. The main area has sections for 'Op Mode Name: My Tank Drive', 'TeleOp', 'Group:', 'Enabled' (checked), and 'Show Java' (checked). A sidebar on the left lists categories: LinearOpMode, Gamepad, Actuators, Sensors, Other Devices, Android, Utilities, Logic, Loops, Math, Text, Lists, Variables, Functions, and Miscellaneous. The central workspace displays a LinearOpMode block with nested blocks for 'runOpMode', 'set right\_drive', 'call My Tank Drive . waitForStart', 'if call My Tank Drive . opModelsActive', 'do Put run blocks here.', 'repeat while call My Tank Drive . opModelsActive', 'do Put loop blocks here.', 'set Power', 'left\_drive . to gamepad1 . LeftStickY', 'right\_drive . to gamepad1 . RightStickY', 'call Telemetry . addData', 'key Left Pow', 'number left\_drive . Power', 'call Telemetry . addData', 'key Right Pow', 'number right\_drive . Power', and 'call Telemetry . update'. To the right, the 'Java Code:' section shows the generated Java code:

```

package org.firstinspires.ftc.teamcode;

import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.eventloop.opmode.TeleOp;
import com.qualcomm.robotcore.hardware.DcMotor;
import com.qualcomm.robotcore.hardware.DcMotorSimple;

@TeleOp(name = "MyTankDrive (Blocks to Java)", group = "LinearOpMode")
public class MyTankDrive extends LinearOpMode {

    private DcMotor right_drive;
    private DcMotor left_drive;

    /**
     * This function is executed when this Op Mode is selected
     */
    @Override
    public void runOpMode() {
        right_drive = hardwareMap.dcMotor.get("right_drive");
        left_drive = hardwareMap.dcMotor.get("left_drive");

        // Reverse one of the drive motors.
        // You will have to determine which motor to reverse
        // In this example, the right motor was reversed.
    }
}

```

2. **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.

The screenshot shows the FTC Java code editor interface. At the top, there are tabs for "Blocks" and "OnBotJava". Below the tabs, the "Manage" tab is selected. On the left, a sidebar titled "Project Files" lists the package "org.firstinspires.ftc.teamcode" containing files: MyTankDrive.java, SimpleOnBotIterative.java, SimpleOnBotLinear.java, and testmotor.java. The main area is a code editor with the following Java code:

```

1 package org.firstinspires.ftc.teamcode;
2
3 import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
4 import com.qualcomm.robotcore.eventloop.opmode.TeleOp;
5 import com.qualcomm.robotcore.hardware.DcMotor;
6 import com.qualcomm.robotcore.hardware.DcMotorSimple;
7
8 @TeleOp(name = "MyTankDrive (Blocks to Java)", group = "")
9 public class MyTankDrive extends LinearOpMode {
10
11     private DcMotor right_drive;
12     private DcMotor left_drive;
13
14     /**
15      * This function is executed when this Op Mode is selected from
16      */
17     @Override
18     public void runOpMode() {

```

A red circular icon with a wrench symbol is positioned to the right of the code editor. Below the code editor is a terminal window showing the build log:

```

Build started at Mon Jun 17 2019 15:45:25 GMT-0400 (Eastern Daylight Time)

Build finished in 2.9 seconds
Build succeeded!

```

3. **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.

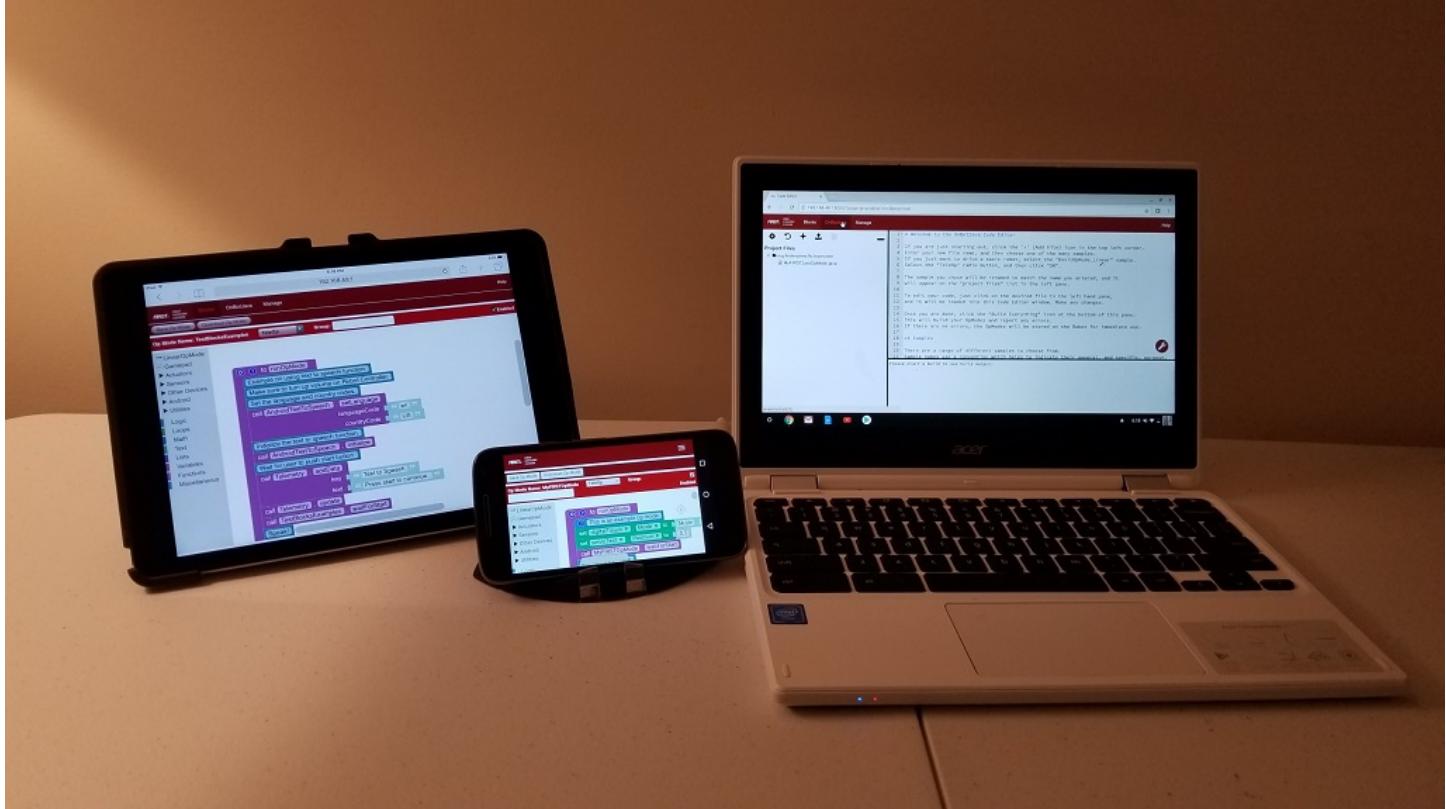
## FTC Docs

The screenshot shows the Android Studio interface with the following details:

- File Menu:** Android Studio, File, Edit, View, Navigate, Code, Analyze, Refactor, Build, Run, Tools, VCS, Window, Help.
- Toolbar:** Includes icons for back, forward, search, and file operations.
- Project Structure:** Shows the project tree under "FtcRobotController". It includes "TeamCode", "manifests", "java" (containing "org.firstinspires.ftc.teamcode" with "readme.md"), "jniLibs", "res", and "Gradle Scripts".
- Code Editor:** Displays the file "PushbotTeleopPOV\_Linear.java". The code is a Java class named "PushbotTeleopPOV\_Linear" that extends "LinearOpMode". It contains comments explaining its functionality, including the use of a POV Game style Teleop for a PushBot. It initializes hardware variables like "robot", "clawOffset", and "CLAW\_SPEED". The "runOpMode" method handles telemetry and waits for the game to start.
- Bottom Bar:** Includes tabs for Messages, Terminal, Android Monitor, TODO, Event Log, and Gradle Console. A status bar at the bottom shows the time as 60:14, encoding as UTF-8, and context as <no context>.

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 is only recommended for advanced users.

## 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!

```

package org.firstinspires.ftc.teamcode;

import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.eventloop.opmode.TeleOp;
import com.qualcomm.robotcore.hardware.DcMotor;
import com.qualcomm.robotcore.hardware.DcMotorSimple;

@TeleOp(name = "MyTankDrive (Blocks to Java)", group)
public class MyTankDrive extends LinearOpMode {

    private DcMotor right_drive;
    private DcMotor left_drive;

    /**
     * This function is executed when this Op Mode is selected
     */
    @Override
    public void runOpMode() {
        right_drive = hardwareMap.dcmotor.get("right_drive");
        left_drive = hardwareMap.dcmotor.get("left_drive");

        // Reverse one of the drive motors.
        // You will have to determine which motor to reverse
        // In this example, the right motor was reversed.
    }

    // Put loop blocks here
}

```

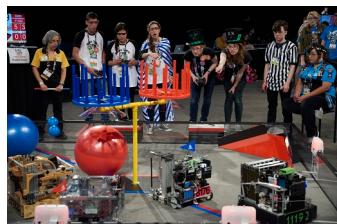
**Note:** Blocks indicates that the content is specific to Blocks Programming

## 1.2.1 Introduction Blocks

### Control System Introduction

#### 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](http://www.dandonovan.com))

## AUTO vs. TELEOP

~~FTC Docs~~ 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.

~~FTC Programming Resources, 8~~

### Point-to-Point Control System

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



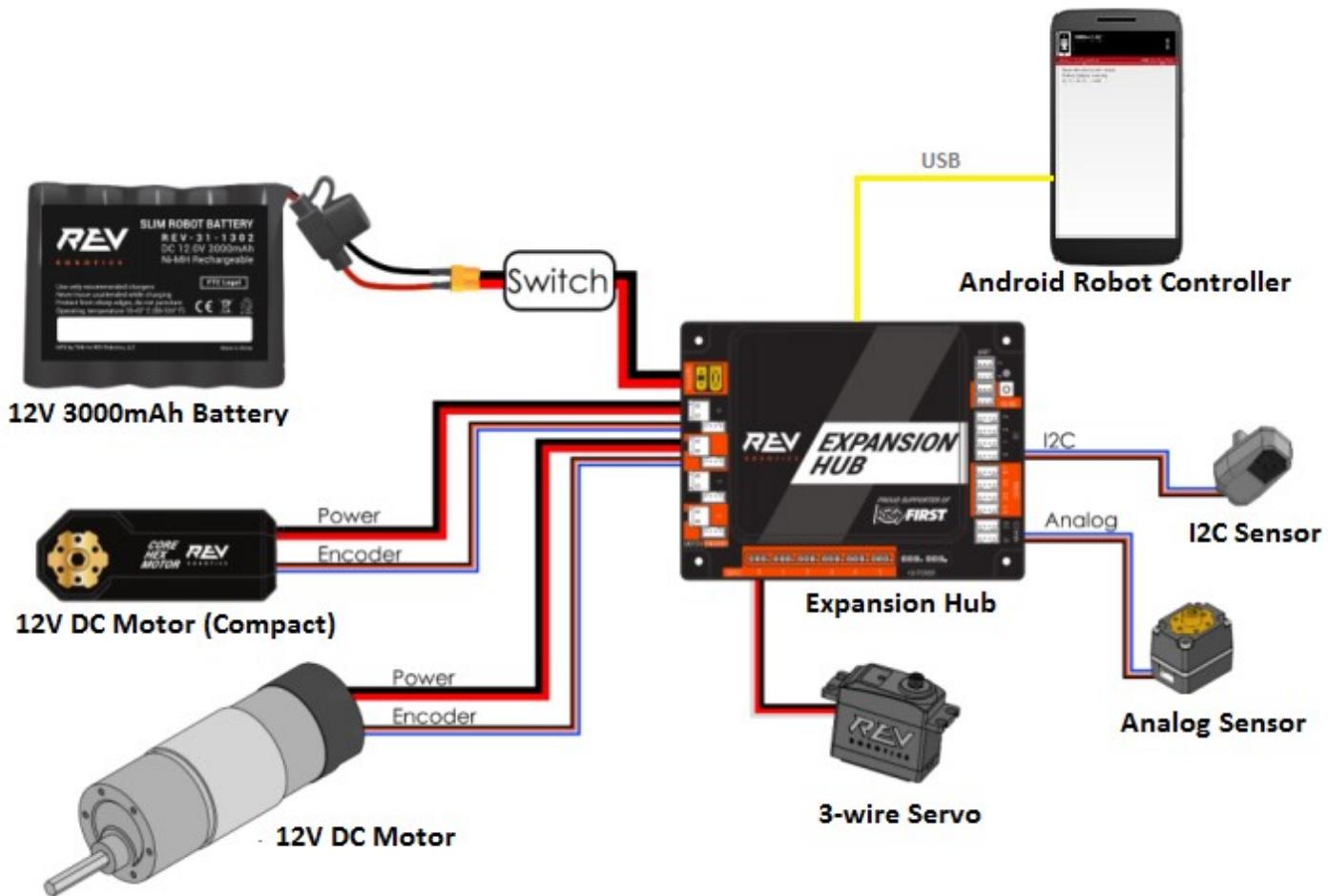
One Android device is mounted onto the robot and is called the *Robot Controller*. The Robot Controller acts as the “brains” of the robot. It does all of the thinking for the robot and tells the robot what to do. It consists of an Android device running an Robot Controller app. There are two hardware options currently being used: REV Robotics Expansion Hub or the REV Robotics Control Hub.

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 sort of 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.

### REV Robotics Expansion Hub

The REV Robotics 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 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. For the situation where a REV Robotics Control Hub is used, an internal serial connection exists between the built-in Android device and the Expansion Hub.

The Expansion Hub is also connected to a 12V battery which is used to power 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. If a REV Robotics Control Hub is used as the Robot Controller, then the Control Hub will use the main 12V battery to power its internal Android device.

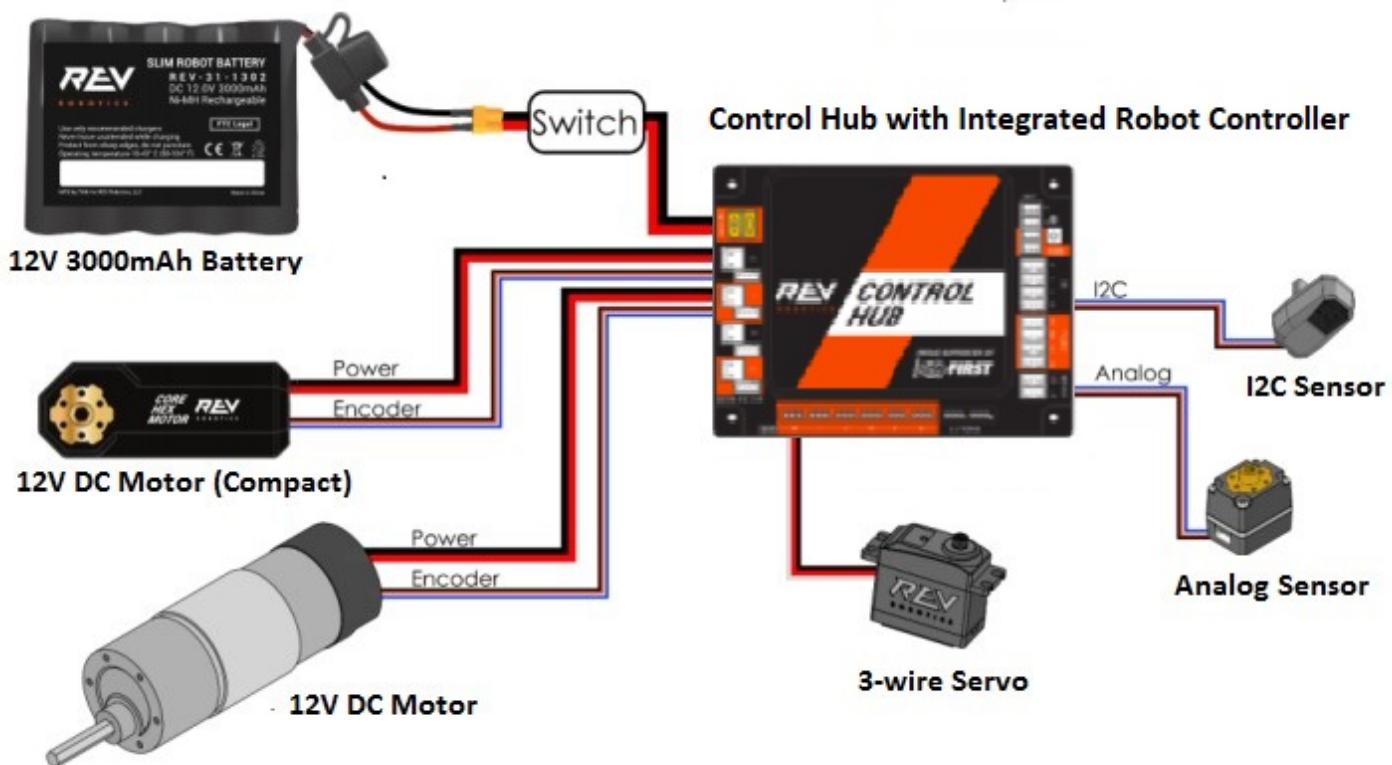


### REV Robotics Control Hub

The REV Robotics Control Hub is an integrated version of the Robot Controller. It combines an Android device built into the same case as a REV Robotics Expansion Hub.



The Control Hub, which has its built-in Android device connected directly to the Expansion Hub using an internal serial bus, eliminates the need for an external USB connection between the Android Robot Controller and the I/O module.



## What's an Op Mode?

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 "op modes" (which stand for "operational modes") to specify the behavior for their robot.

*Op modes* are computer programs that are used to customize the behavior of a competition robot. The Robot Controller can execute a selected op mode 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 op modes. 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 op modes.

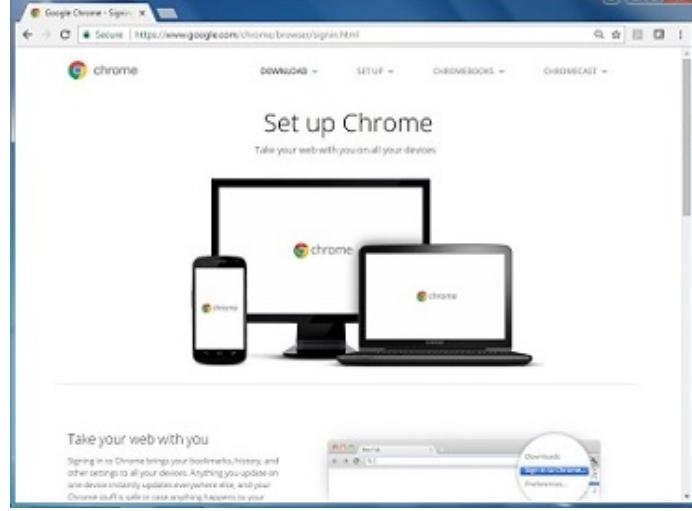
## Required Materials

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
Two (2) FIRST-approved* Android devices OR One (1) Control Hub and one (1) FIRST-approved* Android device for the DRIVER STATION	    <p>Or</p>
Wireless Internet access.	

continues on next page

Table 1 – continued from previous page

Required Item(s)	Image
<p>Laptop with Microsoft Windows 7, 8 or 10 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 howto connect to a Wi-Fi network.</p>	
<p>Javascript-enabled web browser (Google Chrome is the recommended browser).</p>	
<p>If you are using a smartphone as part of your Robot Controller, you will also need a REV Robotics Expansion Hub (REV-31-1153) to connect to the motors, servos, and sensors. Control Hub users will use the integrated ports built into the Control Hub to connect motors, servos, and sensors.</p>	

continues on next page

Table 1 – continued from previous page

Required Item(s)	Image
REV Robotics Switch, Cable, & Bracket (REV-31-1387).	
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.	
<i>FIRST</i> -approved* 12V Battery (such as Tetrix W39057 or REV Robotics REV-31-1302).*For a list of <i>FIRST</i> -approved 12V batteries, refer to the current Competition Manual.	 Or
<i>FIRST</i> -approved* 12V DC Motor (such as Tetrix W39530, with power cable W41352).*For a list of <i>FIRST</i> -approved 12V motors, refer to the current Competition Manual.	
REV Robotics Anderson to JST VH Cable (REV-31-1381).	

continues on next page

Table 1 – continued from previous page

Required Item(s)	Image
180-Degree Standard Scale Servo (such as Hitec HS-485HB).	
REV Robotics Color Sensor with 4-Pin Cable (REV-31-1154).	
REV Robotics Touch Sensor with 4-Pin Cable( REV-31-1425).	
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.	
Logitech F310 USB Gamepad.	

## Using Your Android Device

**FTC Docs** Before you get started with your control system, it is helpful if you familiarize yourself with the basic operation of your Android device.

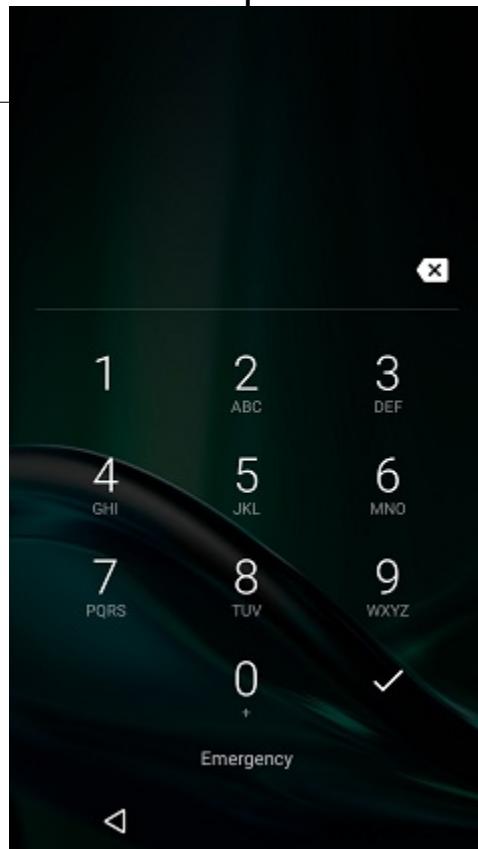
**FTC Programming Resources** 15

### 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.

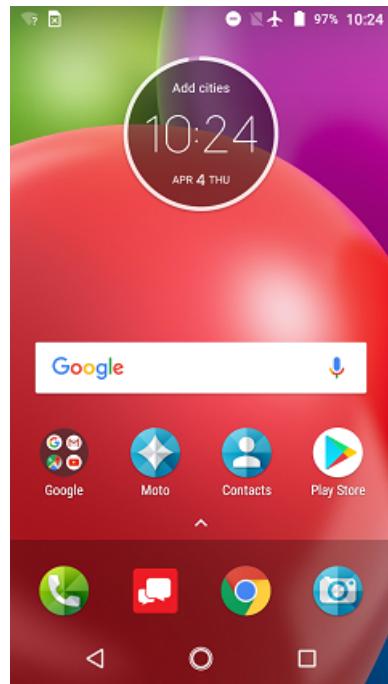


Depending on your security settings, you might be challenged for a pass code or PIN number. Use the touch screen to enter in the pass code or PIN value and tap on the check mark to log into the device.



## 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.



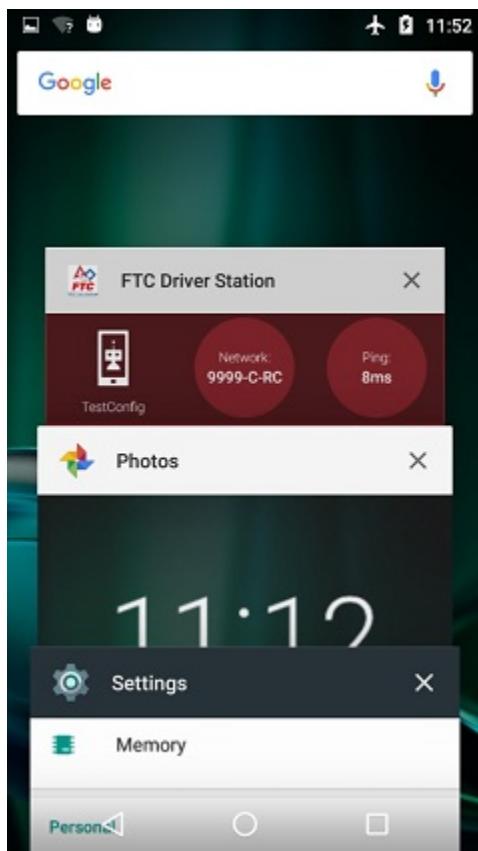
At the bottom of the screen there should be some buttons that you can use to navigate the screens on your Android device.



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

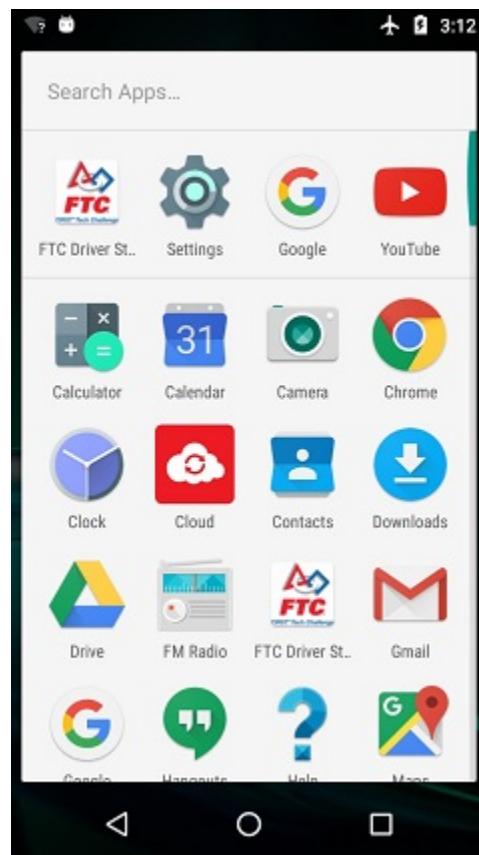
### Android Marshmallow Users

If you are using a device with Android Marshmallow (6.x) or earlier, you can display the available apps using the *Android App Drawer* button that is available on the home screen.

There should be another row of buttons visible above the “Back”, “Home” and “Recent Apps” buttons. In the center of this row of buttons is a button that has an array of dots or squares.

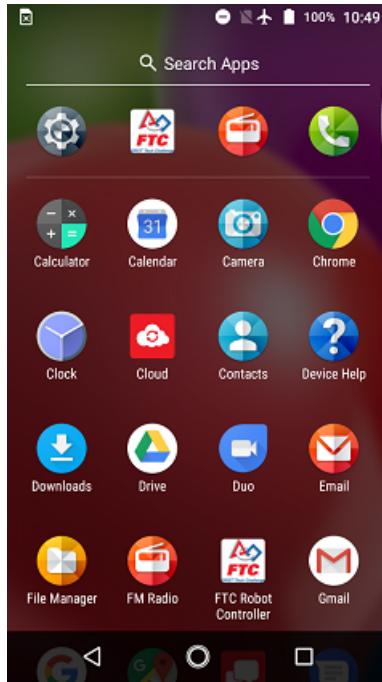


Tapping on this button will launch the *Android App Drawer*. The App Drawer displays a list of all of the apps that are available on your Android device. You can scroll through the App Drawer screens to find and launch an app.



## Android Nougat Users

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.



### 1.2.2 Configuring Your Hardware Blocks

#### Configuring your Android Devices

##### What Needs to Be Configured for My Control System?

##### Control Hub Configuration

---

**Note:** References to the DRIVER STATION smartphone may instead apply to the REV 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 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 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 <RS01>) 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.

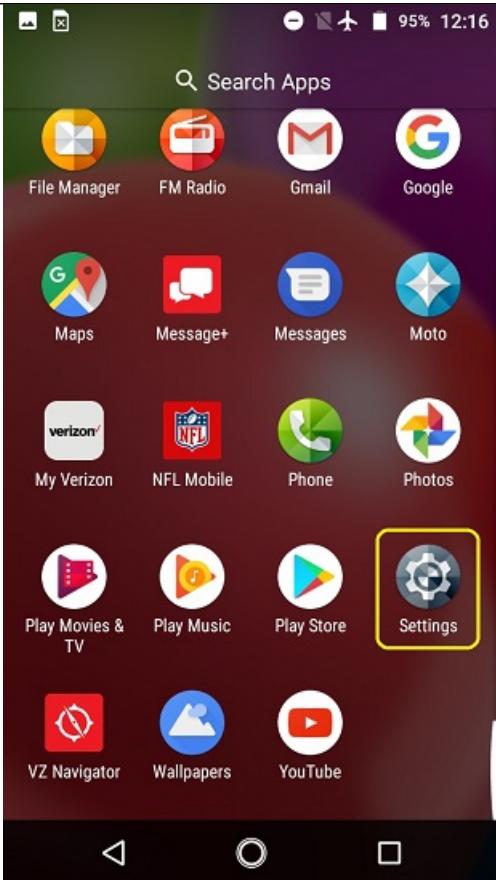
If, for example, a team has a team number of 9999 and the team has multiple sets of phones, the team might decide to name one phone “9999-C-RC” for the Robot Controller and the other phone “9999-C-DS” for the DRIVER STATION. The “-C” indicates that these devices belong to the third set of phones for 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](#).

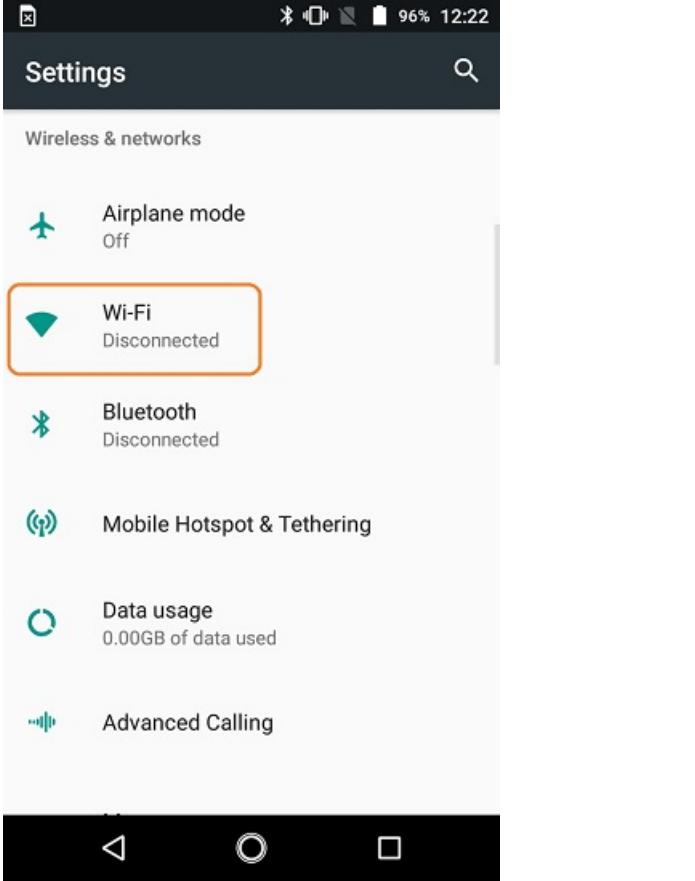
As an alternate, the device names can be changed at the Android system level, as described below.

**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 <b>Settings</b> icon. Click on <b>Settings</b> icon to display the Settings screen.	

continues on next page

Table 2 – continued from previous page

	
2. Click on <b>Wi-Fi</b> to launch the Wi-Fi screen.	

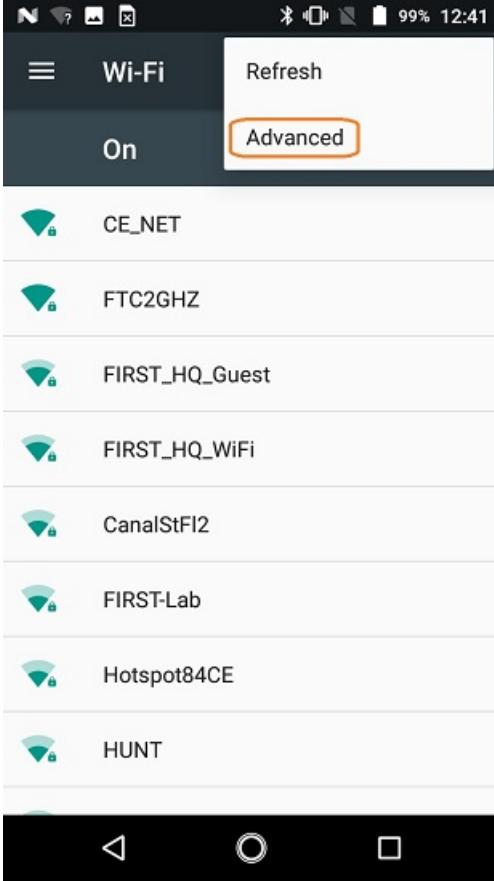
continues on next page

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 <b>Advanced</b> from the pop-up menu.	

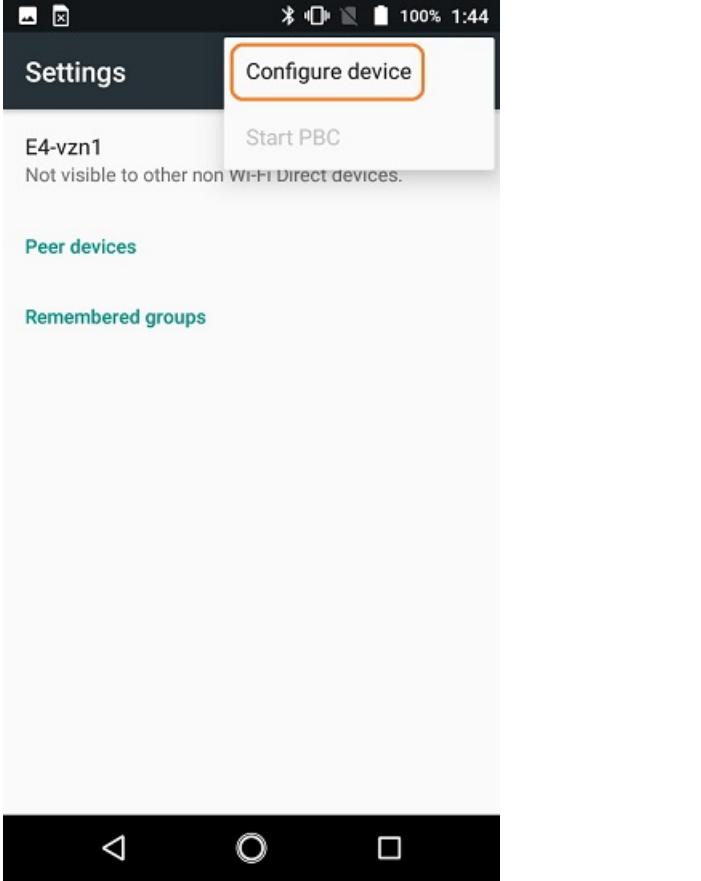
continues on next page

Table 2 – continued from previous page

5. Select <b>Wi-Fi Direct</b> from the <b>Advanced Wi-Fi</b> screen.	

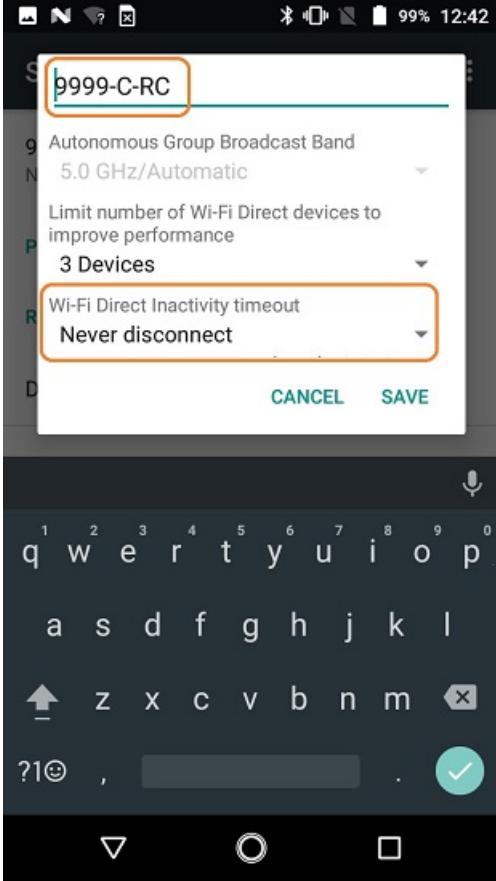
continues on next page

Table 2 – continued from previous page

	
7. Select <b>Configure Device</b> from the pop-up menu.	

continues on next page

Table 2 – continued from previous page

	 <p>The screenshot shows the Wi-Fi settings on an Android device. The team number '9999-C-RC' is entered in the device name field. The 'Wi-Fi Direct Inactivity timeout' dropdown is set to 'Never disconnect'. The 'SAVE' button is highlighted.</p>
8.	<p>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 <i>Never disconnect</i> and then hit the <b>SAVE</b> 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.</p>
9.	<p>After renaming phone, power cycle the device.</p>

## Installing the *FIRST* Tech Challenge Apps

**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 Control Hub, REV Expansion Hub, REV 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 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.

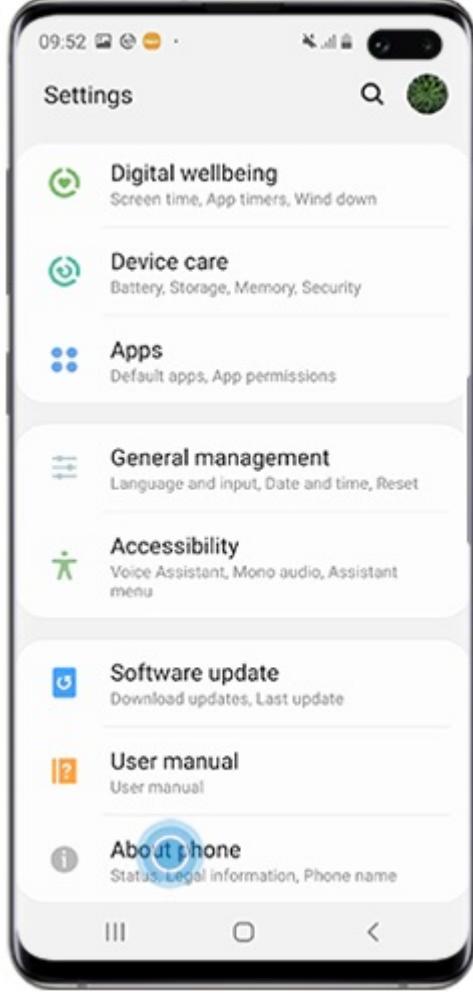
The app releases are also available on the [FTCRobotController Github repository](#). It is possible to “side-load” the apps onto the Robot Controller (RC) and Driver Station (DS) phones. However, this section of the document does **not** include such instructions; other document pages describe side-loading the [RC app](#) and the [DS app](#).

### Updating Apps and Firmware on REV Devices (REV Expansion Hub, REV Control Hub, REV 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 Control Hub](#), [update the Driver Station \(DS\) app on a REV 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:

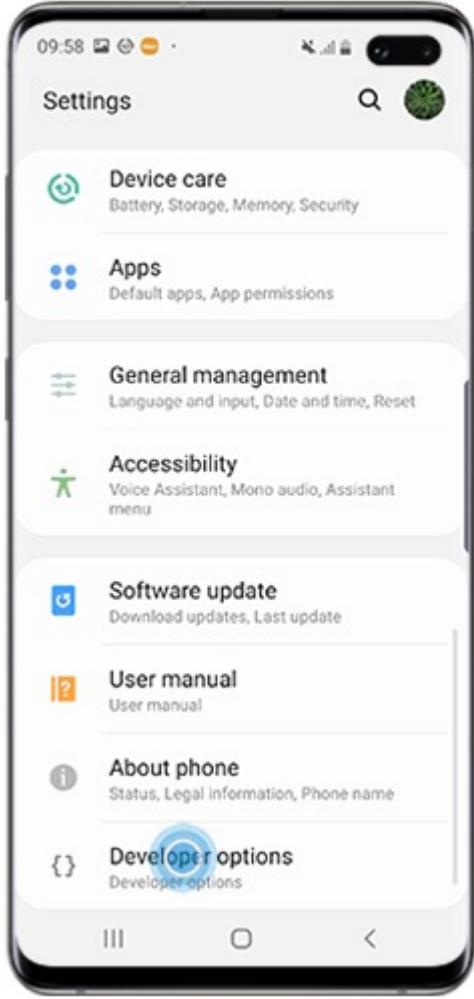
Step	Image
1. Go to "Settings", then tap "About device" or "About phone".	 A screenshot of an Android smartphone's Settings menu. The screen shows several sections: Digital wellbeing (Screen time, App timers, Wind down), Device care (Battery, Storage, Memory, Security), Apps (Default apps, App permissions), General management (Language and input, Date and time, Reset), Accessibility (Voice Assistant, Mono audio, Assistant menu), Software update (Download updates, Last update), User manual (User manual), and About phone (Status, Legal information, Phone name). The 'About phone' option is highlighted with a blue circle.

continues on next page

Table 3 – continued from previous page

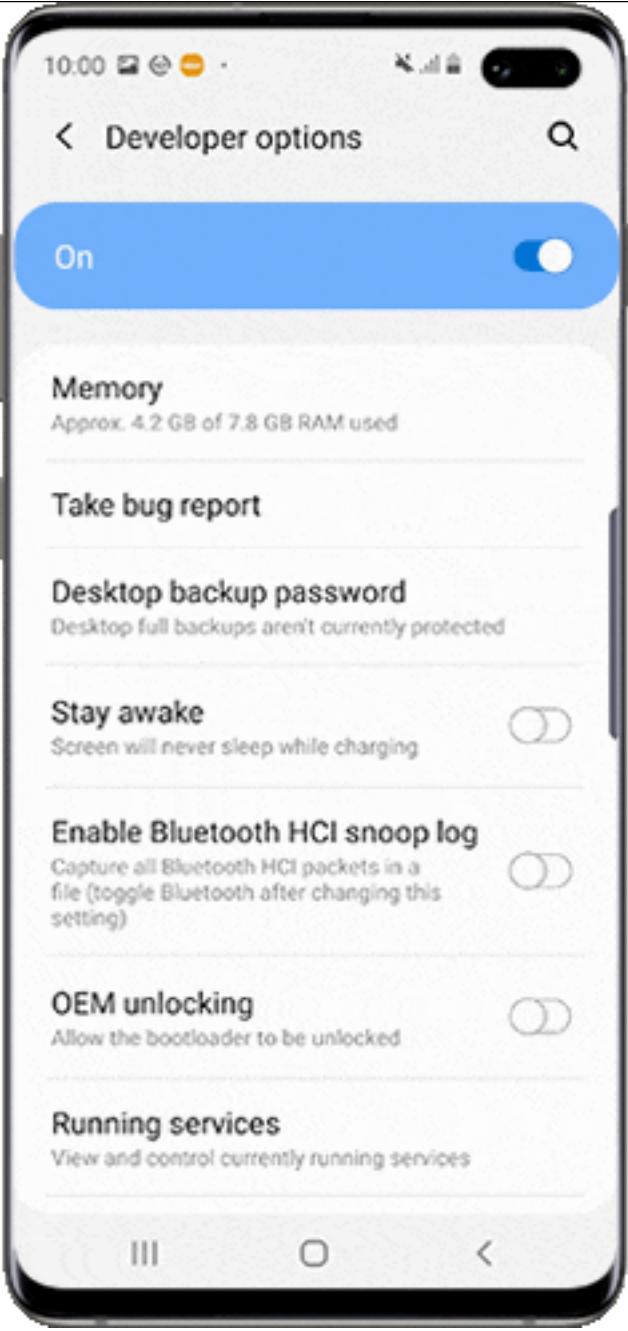
Step	Image										
	<p>09:53</p> <p>&lt; About phone</p> <p>Edit</p> <table> <tbody> <tr><td>Phone number</td><td>Unknown</td></tr> <tr><td>Model number</td><td>SM-G973F</td></tr> <tr><td>Serial number</td><td>RF8M11VHRNJ</td></tr> <tr><td>IMEI (slot 1)</td><td>351910100206866</td></tr> <tr><td>IMEI (slot 2)</td><td>351911100206864</td></tr> </tbody> </table> <p>Status View the SIM card status, IMEI, and other information.</p> <p>Legal information</p> <p>Software information View the currently installed Android version, baseband version, kernel version, build number, and more.</p> <p>Battery information View your phone's battery status, remaining power, and other information.</p>	Phone number	Unknown	Model number	SM-G973F	Serial number	RF8M11VHRNJ	IMEI (slot 1)	351910100206866	IMEI (slot 2)	351911100206864
Phone number	Unknown										
Model number	SM-G973F										
Serial number	RF8M11VHRNJ										
IMEI (slot 1)	351910100206866										
IMEI (slot 2)	351911100206864										
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.	<p>09:53</p> <p>&lt; Software information</p> <table> <tbody> <tr><td>One UI version</td><td>1.1</td></tr> <tr><td>Android version</td><td>9</td></tr> <tr><td>Baseband version</td><td>G973FXXU1ASD4</td></tr> <tr><td>Kernel version</td><td>4.14.85-15820661 #1 Tue Apr 16 17:32:20 KST 2019</td></tr> <tr><td>Build number</td><td>PPR1.180610.011.G973FXXU1ASD5</td></tr> </tbody> </table> <p>SE for Android status Enforcing SEPF_SM-G973F_9_0001 Tue Apr 16 17:09:58 2019</p> <p>Knox version Knox 3.3 Knox API level 28 Trust 4.1.0 DualDAR 1.0</p>	One UI version	1.1	Android version	9	Baseband version	G973FXXU1ASD4	Kernel version	4.14.85-15820661 #1 Tue Apr 16 17:32:20 KST 2019	Build number	PPR1.180610.011.G973FXXU1ASD5
One UI version	1.1										
Android version	9										
Baseband version	G973FXXU1ASD4										
Kernel version	4.14.85-15820661 #1 Tue Apr 16 17:32:20 KST 2019										
Build number	PPR1.180610.011.G973FXXU1ASD5										

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.	 A screenshot of an Android smartphone displaying the 'Settings' menu. The screen shows various options: 'Device care' (Battery, Storage, Memory, Security), 'Apps' (Default apps, App permissions), 'General management' (Language and input, Date and time, Reset), 'Accessibility' (Voice Assistant, Mono audio, Assistant menu), 'Software update' (Download updates, Last update), 'User manual' (User manual), 'About phone' (Status, Legal information, Phone name), and 'Developer options' (Developer options). The 'Developer options' item is highlighted with a blue oval.

continues on next page

Table 3 – continued from previous page

Step	Image
5. To disable the Developer options at anytime, tap the switch.	

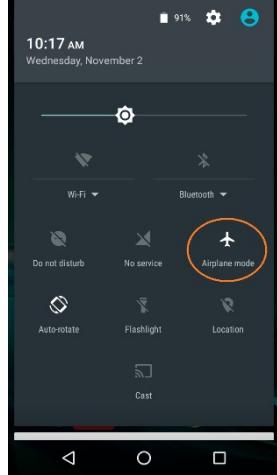
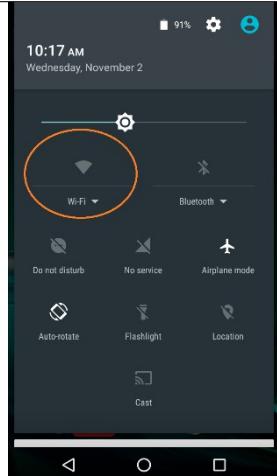
## Placing Phones into Airplane Mode with Wi-Fi On

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
<p>1. On the main Android screen of each smartphone, use your finger to slide from the top of the screen down towards the bottom of the 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.</p>	
<p>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.</p>	

## 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.	

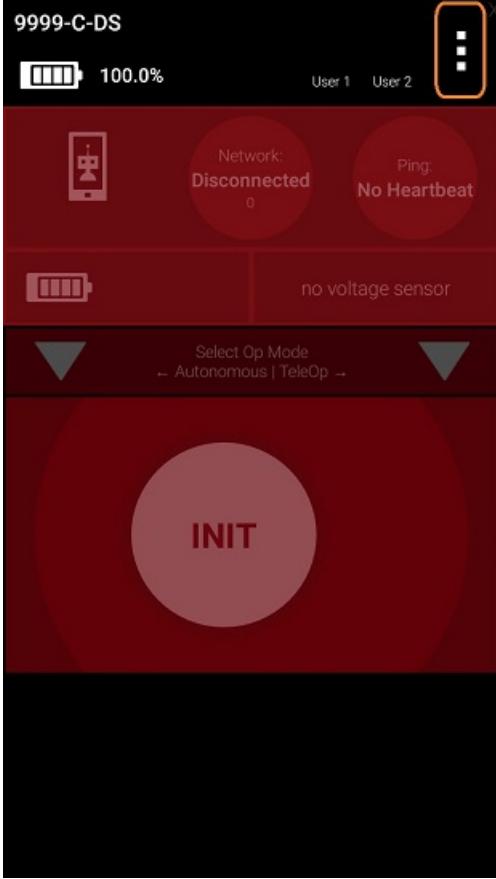
continues on next page

Table 4 – continued from previous page

Step	Image
<p>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.</p>	 <p>~5 Seconds</p>
<p>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 <b>Allow</b> to grant the requested permission.</p>	

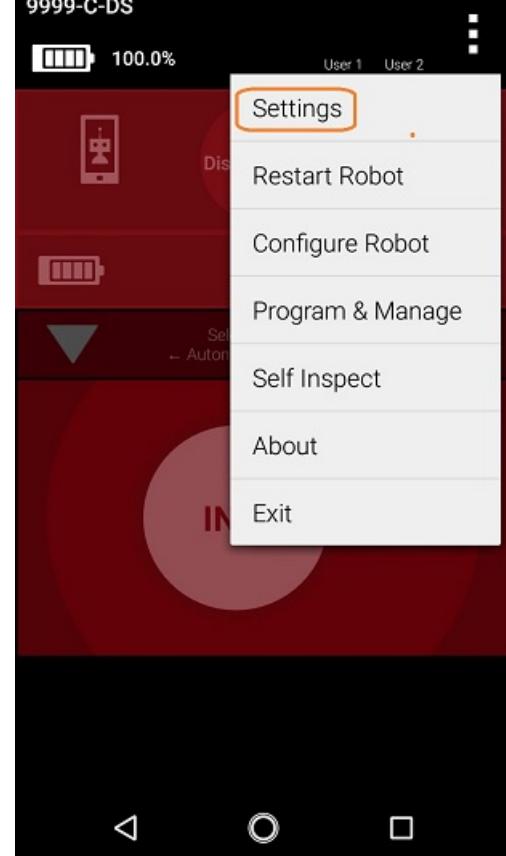
continues on next page

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.	

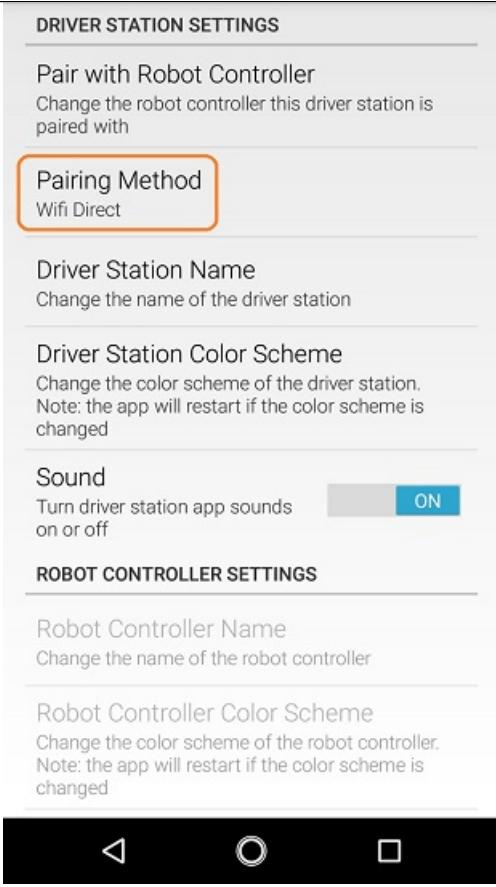
continues on next page

Table 4 – continued from previous page

Step	Image
<p>5. Select <b>Settings</b> from the pop-up menu.</p>	 <p>The image shows a smartphone screen displaying the FTC DriverStation app interface. At the top, it says "9999-C-DS" and shows a battery icon at 100.0%. Below this is a red navigation bar with icons for robot status, battery level, and auton selection. A white context menu is overlaid on the screen, listing several options: "Settings" (which is highlighted with an orange rectangle), "Restart Robot", "Configure Robot", "Program &amp; Manage", "Self Inspect", "About", and "Exit". The bottom of the screen has standard Android navigation buttons (back, home, recent apps).</p>

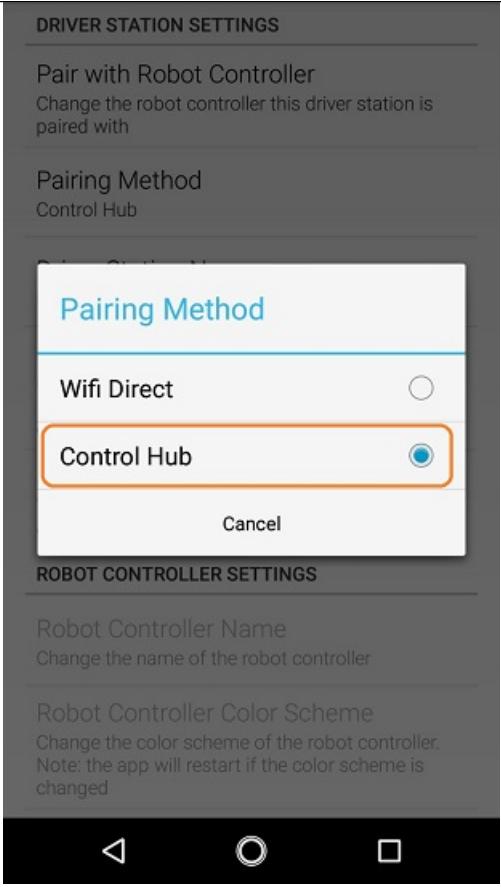
continues on next page

Table 4 – continued from previous page

Step	Image
6. From the <b>Settings</b> screen, look for and select <b>Pairing Method</b> to launch the <b>Pairing Method</b> screen.	 <p><b>DRIVER STATION SETTINGS</b></p> <p>Pair with Robot Controller Change the robot controller this driver station is paired with</p> <p><b>Pairing Method</b> Wifi Direct</p> <p>Driver Station Name Change the name of the driver station</p> <p>Driver Station Color Scheme Change the color scheme of the driver station. Note: the app will restart if the color scheme is changed</p> <p>Sound Turn driver station app sounds <input checked="" type="checkbox"/> ON on or off</p> <p><b>ROBOT CONTROLLER SETTINGS</b></p> <p>Robot Controller Name Change the name of the robot controller</p> <p>Robot Controller Color Scheme Change the color scheme of the robot controller. Note: the app will restart if the color scheme is changed</p> <p style="text-align: center;">◀ ○ □</p>

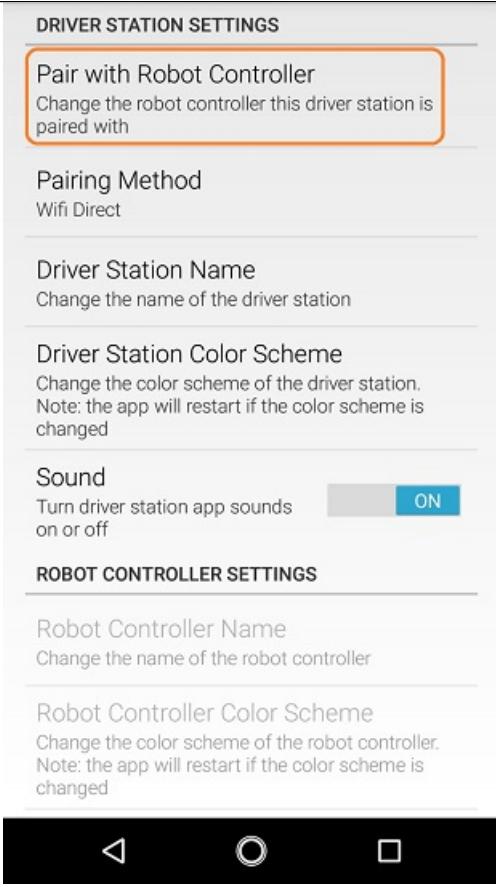
continues on next page

Table 4 – continued from previous page

Step	Image
7. Touch the words <b>Control Hub</b> to indicate that this DRIVER STATION will be pairing with a Control Hub.	 <p>The image shows the 'Pairing Method' screen of the FTC Driver Station app. At the top, there are two options: 'Wifi Direct' and 'Control Hub'. The 'Control Hub' option is selected, indicated by a blue dot next to it and a blue outline around the text. Below the selection area is a 'Cancel' button. Above the selection area, there is a note: 'Pair with Robot Controller' and 'Change the robot controller this driver station is paired with'. At the bottom of the screen, there is a navigation bar with three icons: a triangle pointing left, a circle, and a square.</p>

continues on next page

Table 4 – continued from previous page

Step	Image
8. From the <b>Settings</b> screen, look for and select <b>Pair with Robot Controller</b> to launch the <b>Pair with Robot Controller</b> screen.	 <p><b>DRIVER STATION SETTINGS</b></p> <p><b>Pair with Robot Controller</b> Change the robot controller this driver station is paired with</p> <p>Pairing Method Wifi Direct</p> <p>Driver Station Name Change the name of the driver station</p> <p>Driver Station Color Scheme Change the color scheme of the driver station. Note: the app will restart if the color scheme is changed</p> <p>Sound Turn driver station app sounds <input checked="" type="checkbox"/> ON on or off</p> <p><b>ROBOT CONTROLLER SETTINGS</b></p> <p>Robot Controller Name Change the name of the robot controller</p> <p>Robot Controller Color Scheme Change the color scheme of the robot controller. Note: the app will restart if the color scheme is changed</p> <p style="text-align: center;">◀ ○ □</p>

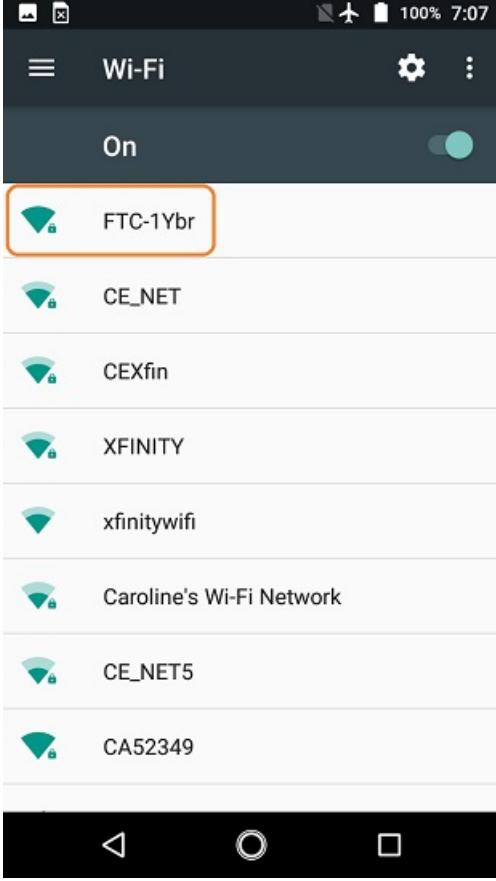
continues on next page

Table 4 – continued from previous page

Step	Image
<p>9. From <b>Pair with Robot Controller</b> screen, look for and press the <b>Wifi Settings</b> button to launch the device's Android WiFi Settings screen.</p>	<p>Wireless access point pairing is used to pair a driver station with a robot controller running on a Control Hub (use WiFi Direct to pair with other robot controllers).</p> <p>Each Control Hub robot controller hosts its own WiFi network named with the name of the robot controller (default password: "password"). Click the button below to use the system WiFi Settings of your driver station to select the network of the robot controller you want to pair with.</p> <p>Current Robot Controller: None </p> 

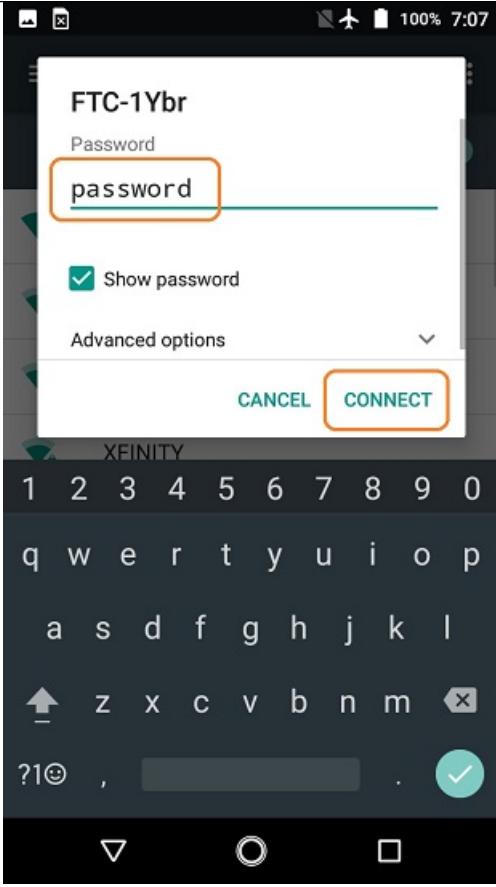
continues on next page

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 <b>Connect</b> to connect to the Hub. Note that the default password for the Control Hub network is <b>password</b> . 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.	<p>Wireless access point pairing is used to pair a driver station with a robot controller running on a Control Hub (use WifiDirect to pair with other robot controllers).</p> <p>Each Control Hub robot controller hosts its own WiFi network named with the name of the robot controller (default password: "password"). Click the button below to use the system WiFi Settings of your driver station to select the network of the robot controller you want to pair with.</p> <p>Current Robot Controller:</p> 

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 WiFi 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.





Table 5 – continued from previous page

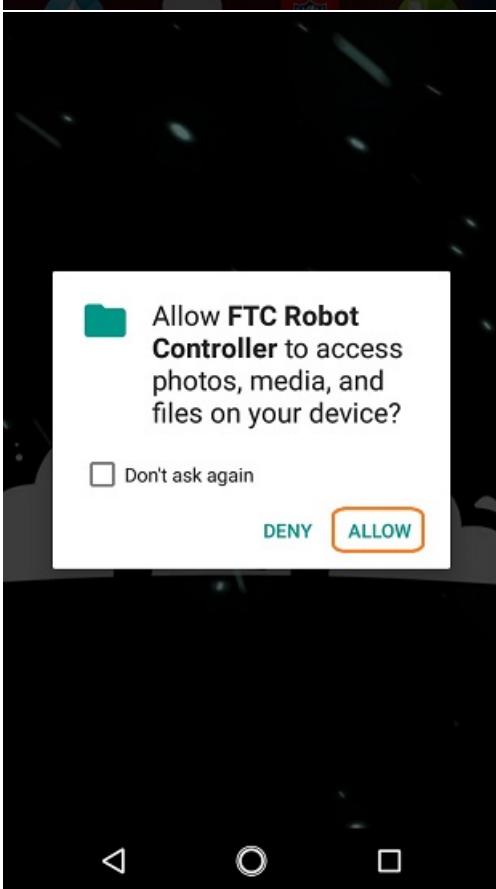
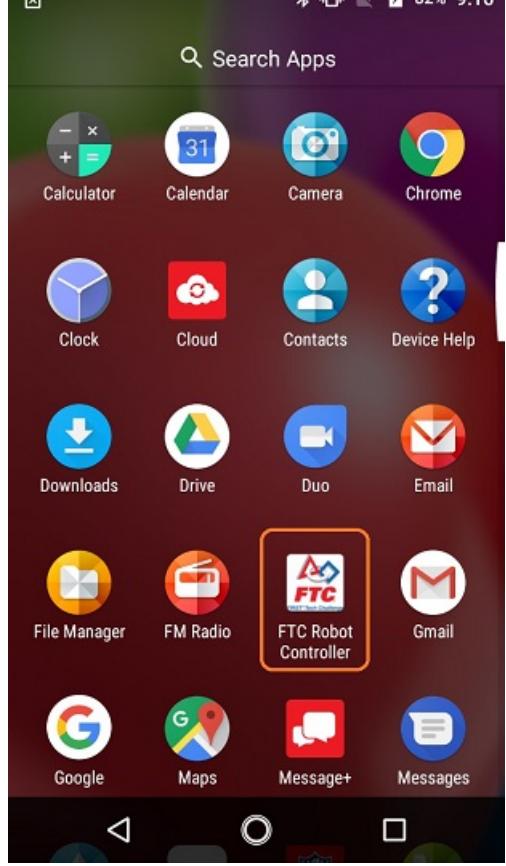
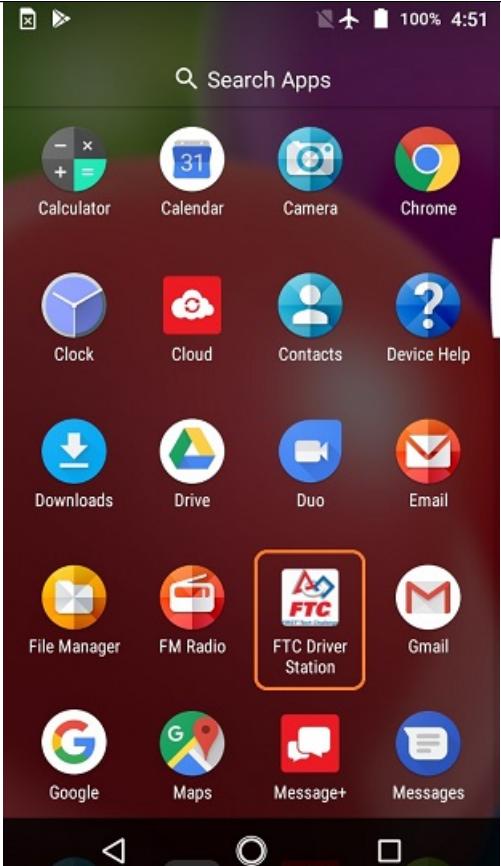
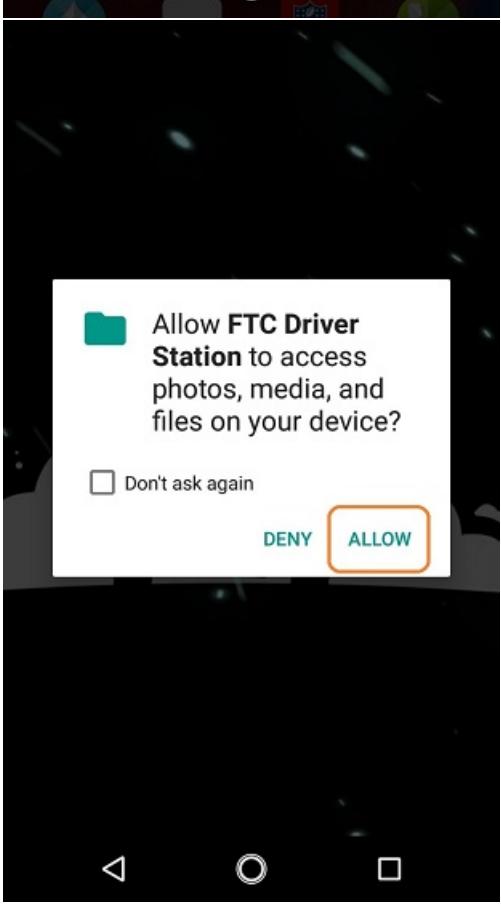
Step	Image
Step	Image
1. On the Robot Controller device, browse the available apps and locate the <b>FTC Robot Controller</b> icon. Tap on the icon to launch the Robot Controller app. Note that the first time you launch the app your Android device might prompt you for permissions that the app will need to run properly. Whenever prompted, press <b>Allow</b> to grant the requested permission.	 

Table 5 – continued from previous page

Step	Image
<p>2. Verify that the Robot Controller app is running. The <b>Robot Status</b> field should read running if it is working properly.</p>	 <p>The image shows the app drawer of an Android smartphone. At the top, there is a search bar labeled "Search Apps". Below the search bar, there are four rows of app icons. In the fourth row, the fifth icon from the left is the "FTC Robot Controller" app, which features a blue and white geometric logo. This icon is highlighted with a thick orange rectangular border. Other visible apps include Calculator, Calendar, Camera, Chrome, Clock, Cloud, Contacts, Device Help, Downloads, Drive, Duo, Email, File Manager, FM Radio, Gmail, Google, Maps, Message+, and Messages. The status bar at the very top shows signal strength, battery level (82%), and the time (9:16).</p>

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 <b>FTC Driver Station</b> 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 <b>Allow</b> to grant the requested permission.	 

continues on next page

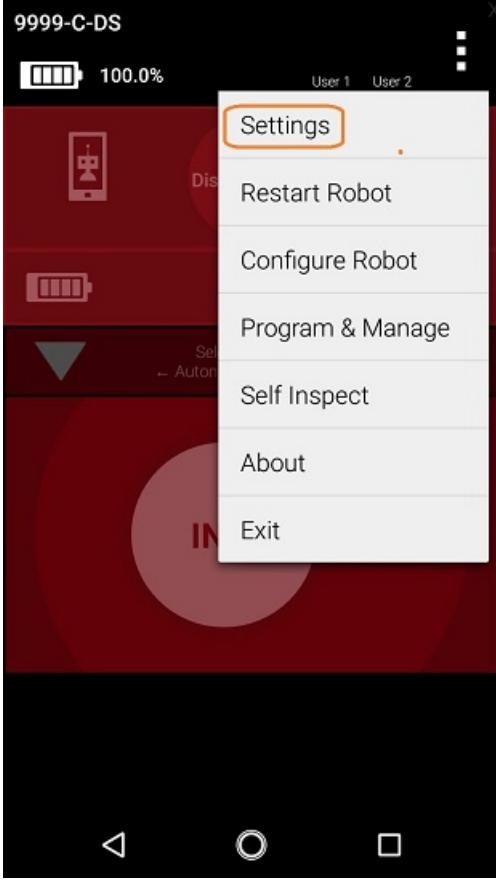
Release 0.2 21/08/2024

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.	

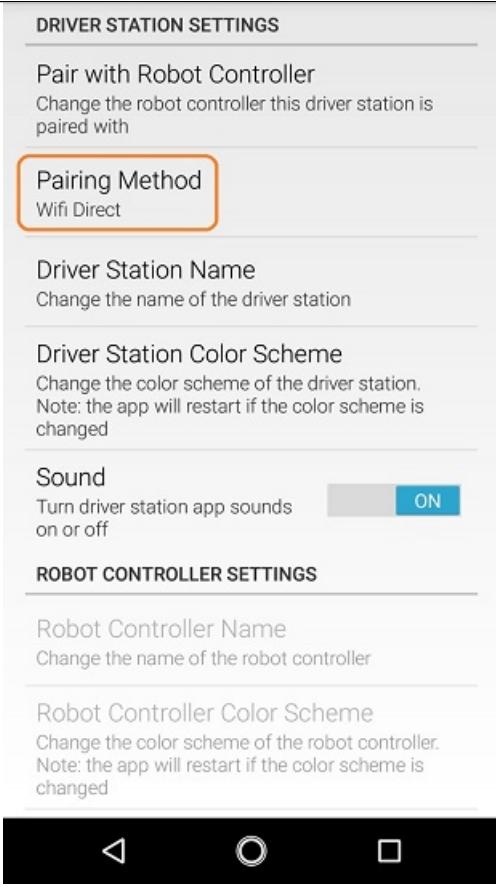
continues on next page

Table 5 – continued from previous page

Step	Image
5. Select <b>Settings</b> from the pop-up menu.	 <p>A screenshot of the FTC mobile application interface. At the top, it shows the team number "9999-C-DS" and a battery icon at "100.0%". Below this is a red navigation bar with icons for "Display", "Configure", "Program", "Inspect", and "About". A white pop-up menu is displayed over the screen, listing several options: "Settings" (which is highlighted with an orange border), "Restart Robot", "Configure Robot", "Program &amp; Manage", "Self Inspect", "About", and "Exit". At the bottom of the screen are three control buttons: a triangle pointing left, a circle, and a square.</p>

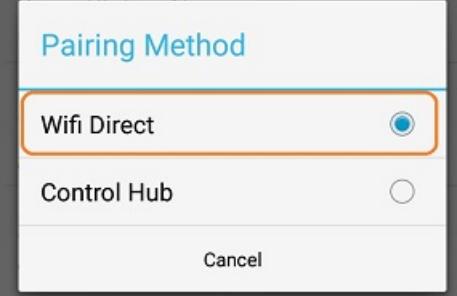
continues on next page

Table 5 – continued from previous page

Step	Image
6. From the <b>Settings</b> screen, look for and select <b>Pairing Method</b> to launch the <b>Pairing Method</b> screen.	 <p><b>DRIVER STATION SETTINGS</b></p> <p>Pair with Robot Controller Change the robot controller this driver station is paired with</p> <p><b>Pairing Method</b> Wifi Direct</p> <p>Driver Station Name Change the name of the driver station</p> <p>Driver Station Color Scheme Change the color scheme of the driver station. Note: the app will restart if the color scheme is changed</p> <p>Sound Turn driver station app sounds <input checked="" type="checkbox"/> ON on or off</p> <p><b>ROBOT CONTROLLER SETTINGS</b></p> <p>Robot Controller Name Change the name of the robot controller</p> <p>Robot Controller Color Scheme Change the color scheme of the robot controller. Note: the app will restart if the color scheme is changed</p> <p style="text-align: center;">◀ ○ □</p>

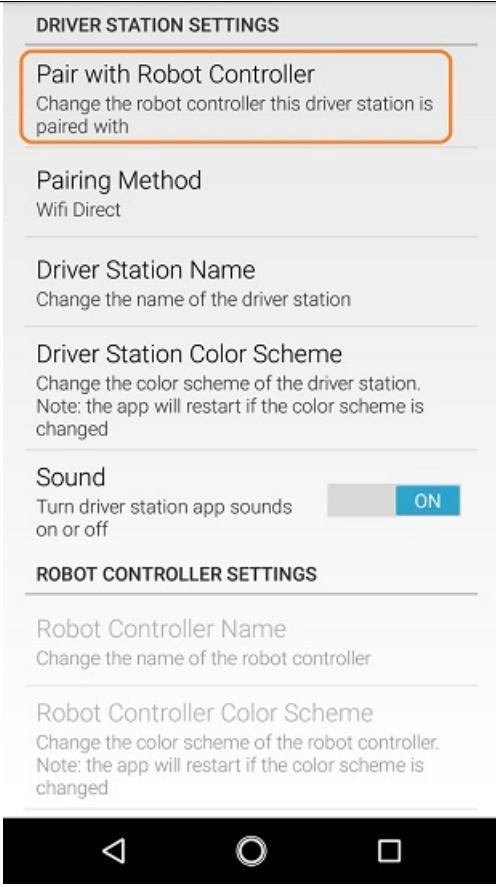
continues on next page

Table 5 – continued from previous page

Step	Image
7. Verify that the <b>Wifi Direct</b> mode is selected, which means that this DRIVER STATION will be pairing with another Android device.	<p>DRIVER STATION SETTINGS</p> <p>Pair with Robot Controller Change the robot controller this driver station is paired with</p> <p>Pairing Method Wifi Direct</p>  <p>ROBOT CONTROLLER SETTINGS</p> <p>Robot Controller Name Change the name of the robot controller</p> <p>Robot Controller Color Scheme Change the color scheme of the robot controller. Note: the app will restart if the color scheme is changed</p> <p>&lt;    ○    □</p>

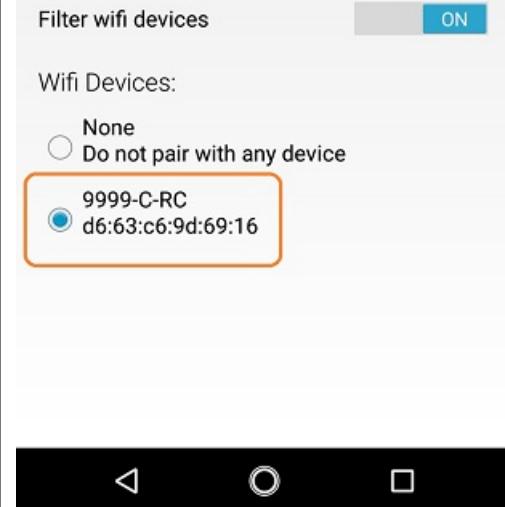
continues on next page

Table 5 – continued from previous page

Step	Image
8. From the <b>Settings</b> screen, look for and select <b>Pair with Robot Controller</b> to launch the <b>Pair with Robot Controller</b> screen.	 <p><b>DRIVER STATION SETTINGS</b></p> <p><b>Pair with Robot Controller</b> Change the robot controller this driver station is paired with</p> <p>Pairing Method Wifi Direct</p> <p>Driver Station Name Change the name of the driver station</p> <p>Driver Station Color Scheme Change the color scheme of the driver station. Note: the app will restart if the color scheme is changed</p> <p>Sound Turn driver station app sounds <input checked="" type="checkbox"/> ON on or off</p> <p><b>ROBOT CONTROLLER SETTINGS</b></p> <p>Robot Controller Name Change the name of the robot controller</p> <p>Robot Controller Color Scheme Change the color scheme of the robot controller. Note: the app will restart if the color scheme is changed</p> <p style="text-align: center;">◀ ○ □</p>

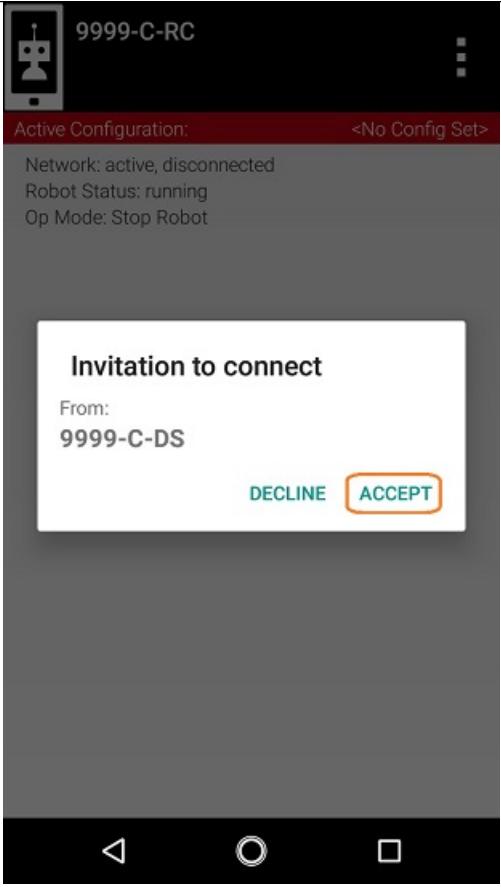
continues on next page

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.	<p>Make sure that the FTC Robot Controller app is open on your other device, and that both devices have wifi enabled.</p> <p>Choose from the following list of Robot Controller phones. Non-matching/incorrectly named phones will not appear. Robot Controller names need to begin with the team number matching your Driver Station.</p> <p>This device's name is: <b>9999-C-DS</b></p> 

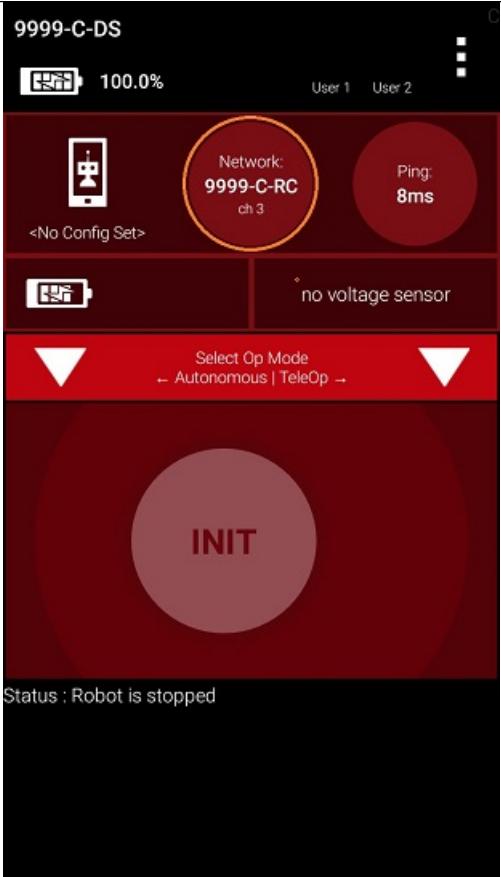
continues on next page

Table 5 – continued from previous page

Step	Image
10.	 <p>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 <b>ACCEPT</b> button to accept the connection request from the DRIVER STATION.</p>

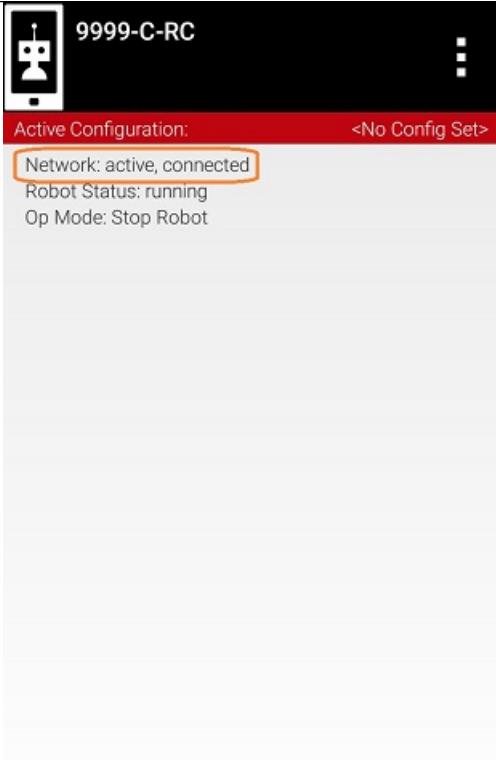
continues on next page

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 of the Robot Controller's remote network (9999-C-RC in this example) should be displayed in the Network field on the DRIVER STATION.	 <p>The screenshot shows the FTC Driver Station interface. At the top, it displays "9999-C-DS" and a battery icon at 100.0%. Below this, there are two user icons labeled "User 1" and "User 2". In the center, a circular "Network" field is highlighted with a yellow border, showing "9999-C-RC ch 3". To the right of this is a "Ping" field showing "8ms". On the left, a "Config Set" section says "&lt;No Config Set&gt;". Below the network field, there is a "no voltage sensor" message. At the bottom, a large red button labeled "INIT" is prominent, and the status message "Status : Robot is stopped" is displayed.</p>

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 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.

## Connecting 12V Power to the Hub

FTC Docs

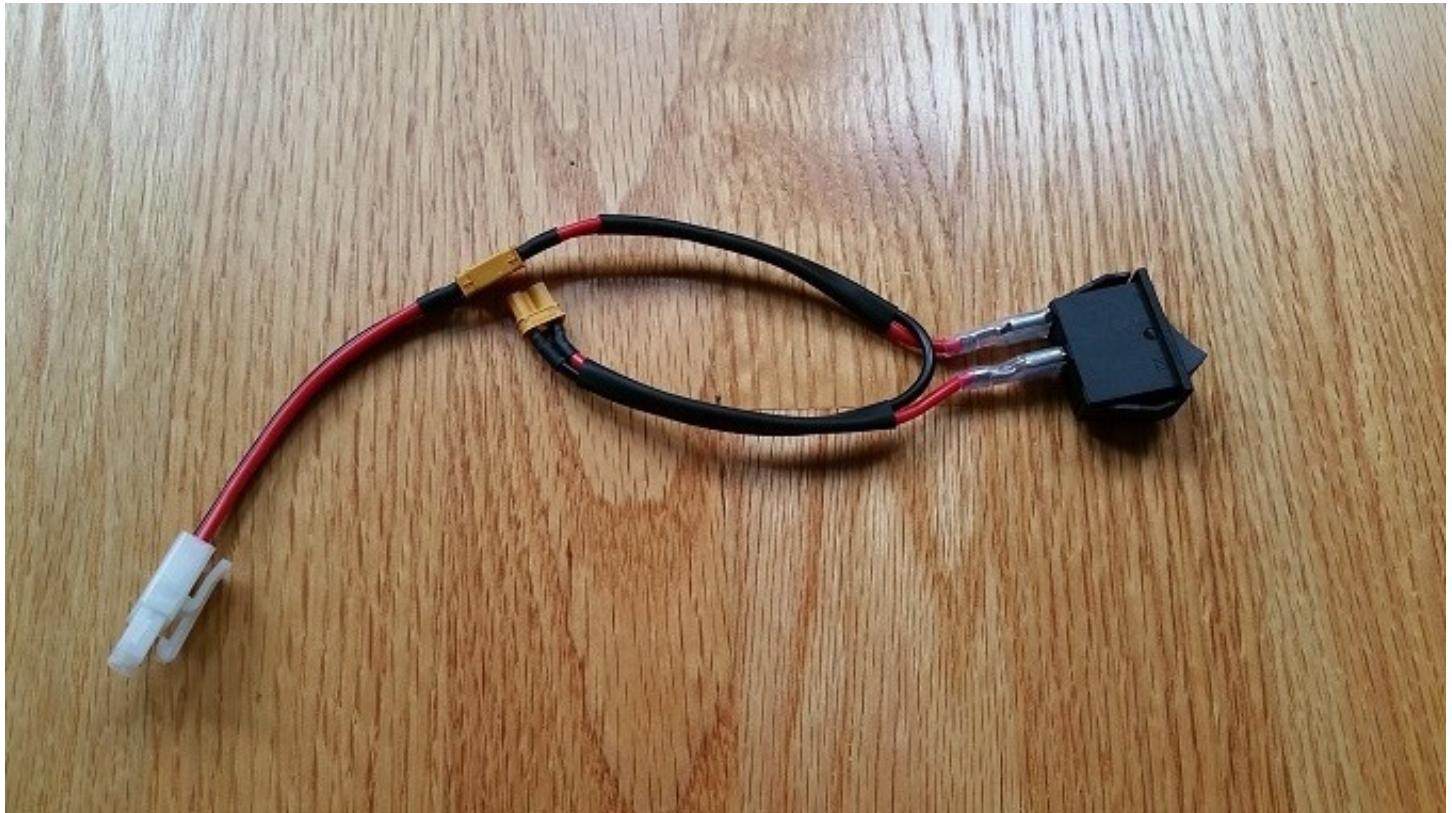
The Hub draws power from a 12V rechargeable battery. For safety reasons, the battery has a 20A fuse built in. A mechanical switch is used to turn on/turn off the power.

FTC Programming Resources, 62

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).

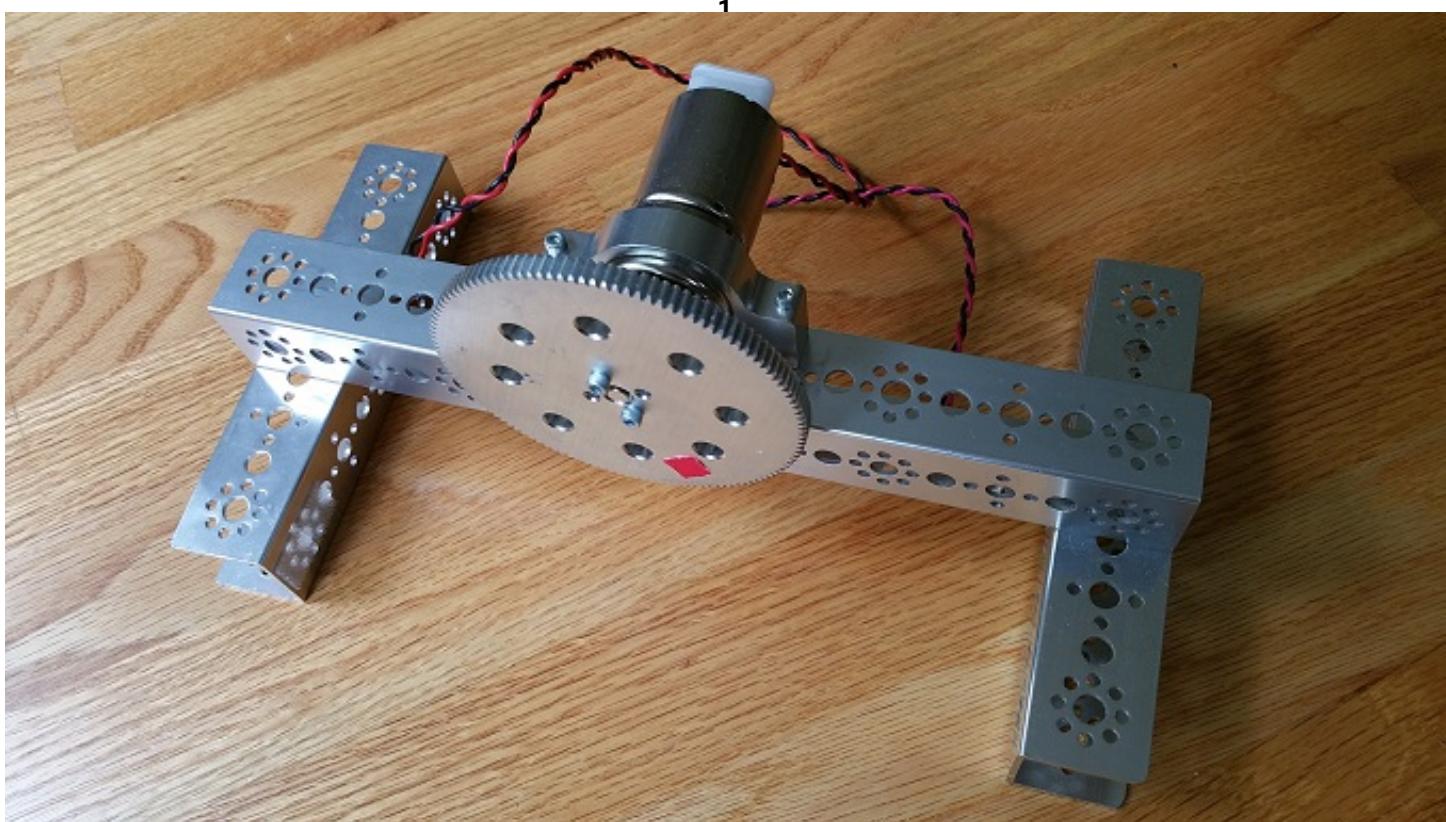


6. Turn off the switch and verify that the Hub is off. Note that the Hub's LED should not be illuminated.



### 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).

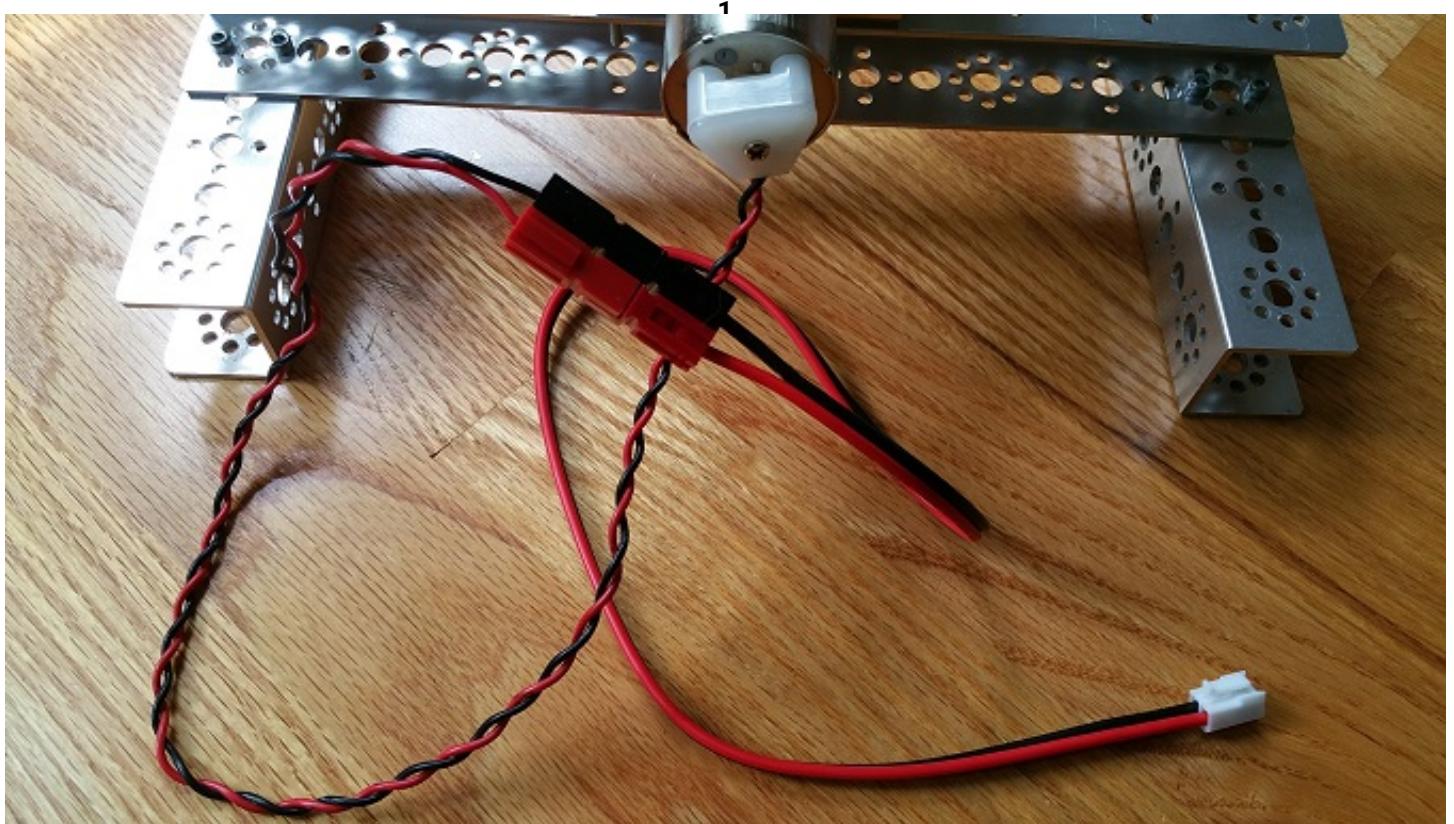


For the examples in this tutorial, *FIRST* recommends that the user build a simple rig to secure the motor in place and prevent it from moving about during the test runs. The image above shows a Tetrix motor installed in a rig built with a Tetrix motor mount and some Tetrix C-channels. A gear was mounted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

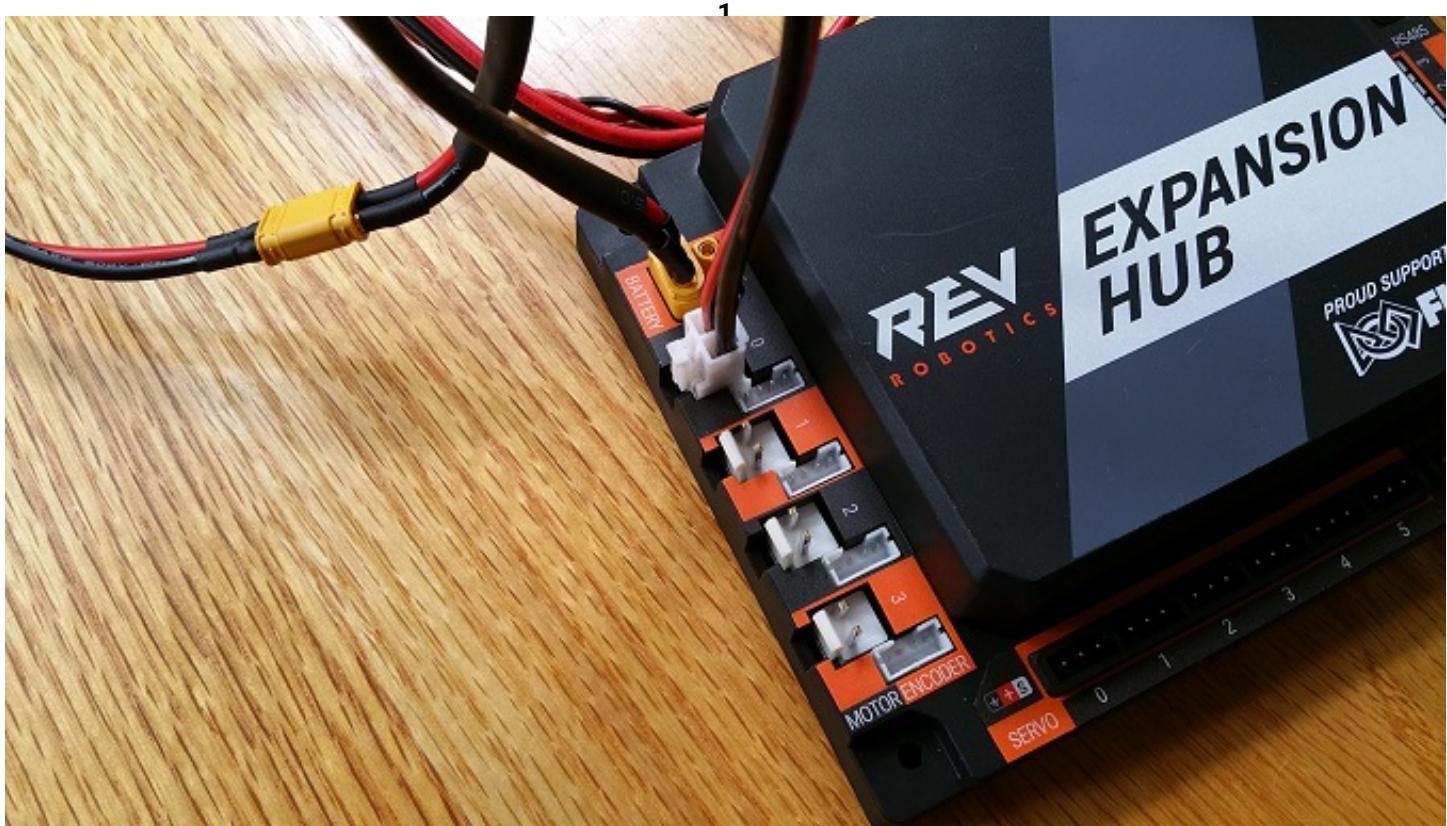
Note that it will take an estimated 2.5 minutes to complete this task.

#### **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.



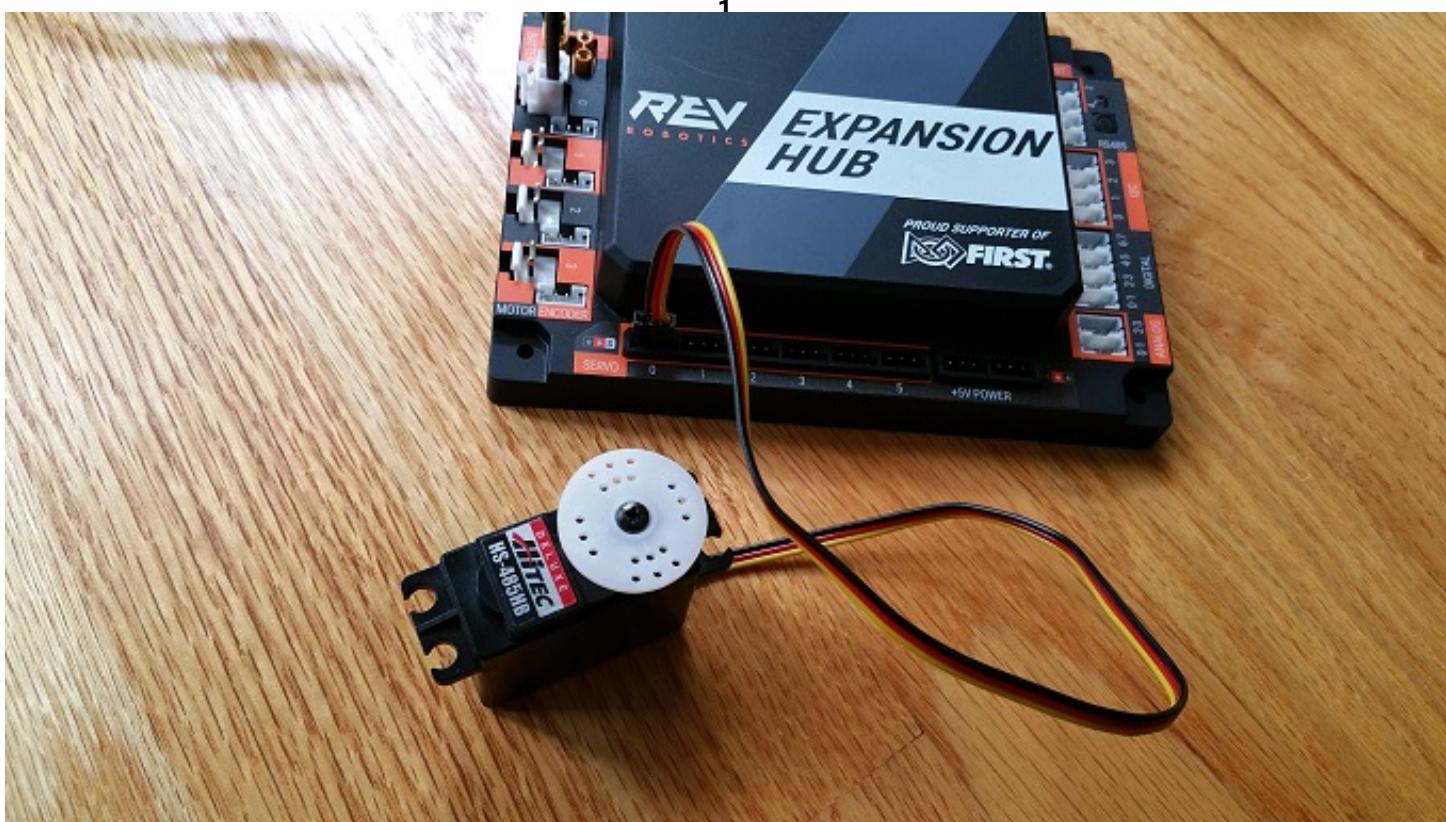
## Connecting a Servo to the Hub

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.

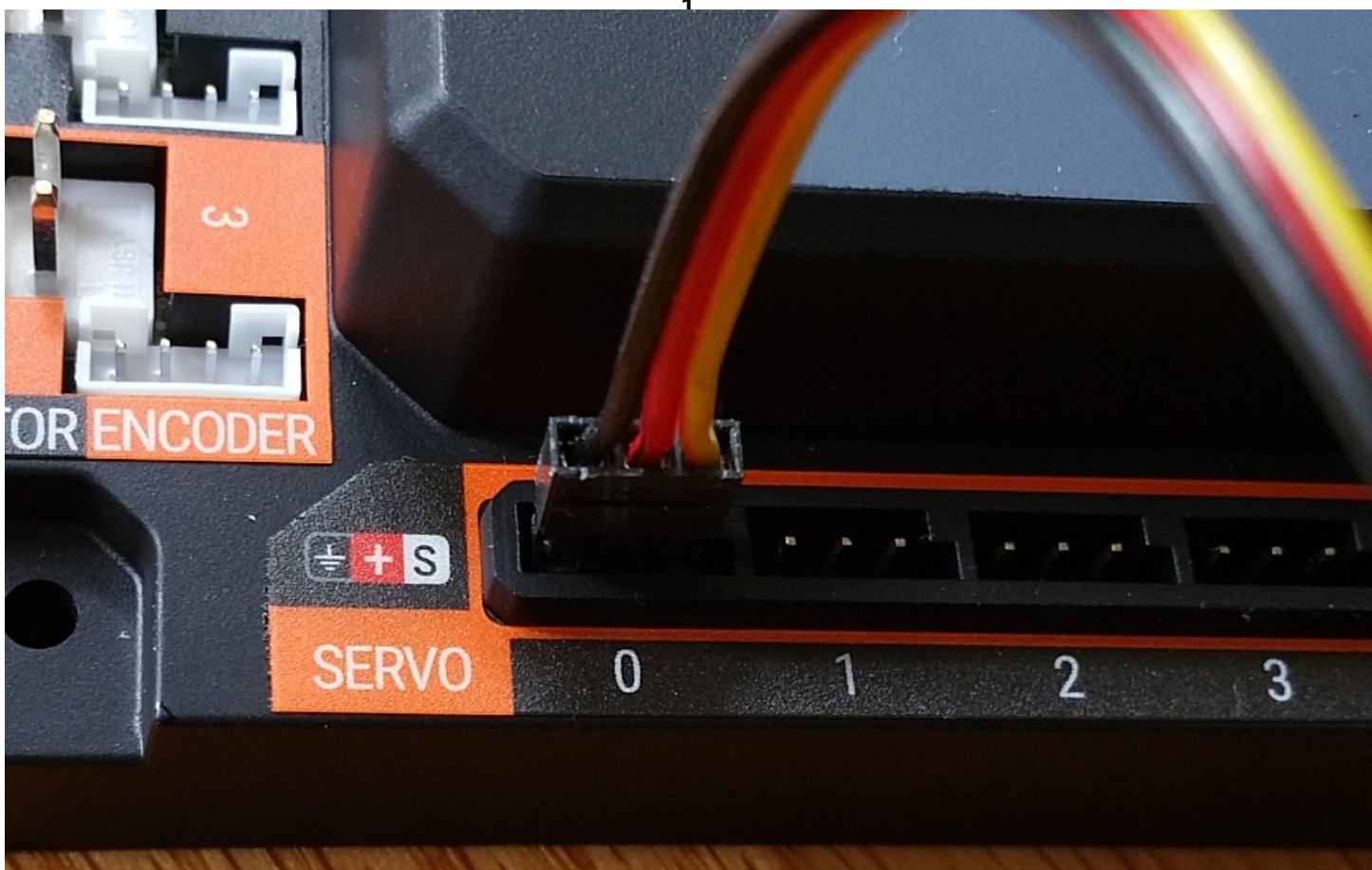
Note that it will take an estimated 2.5 minutes to complete this task.

### 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).



### 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

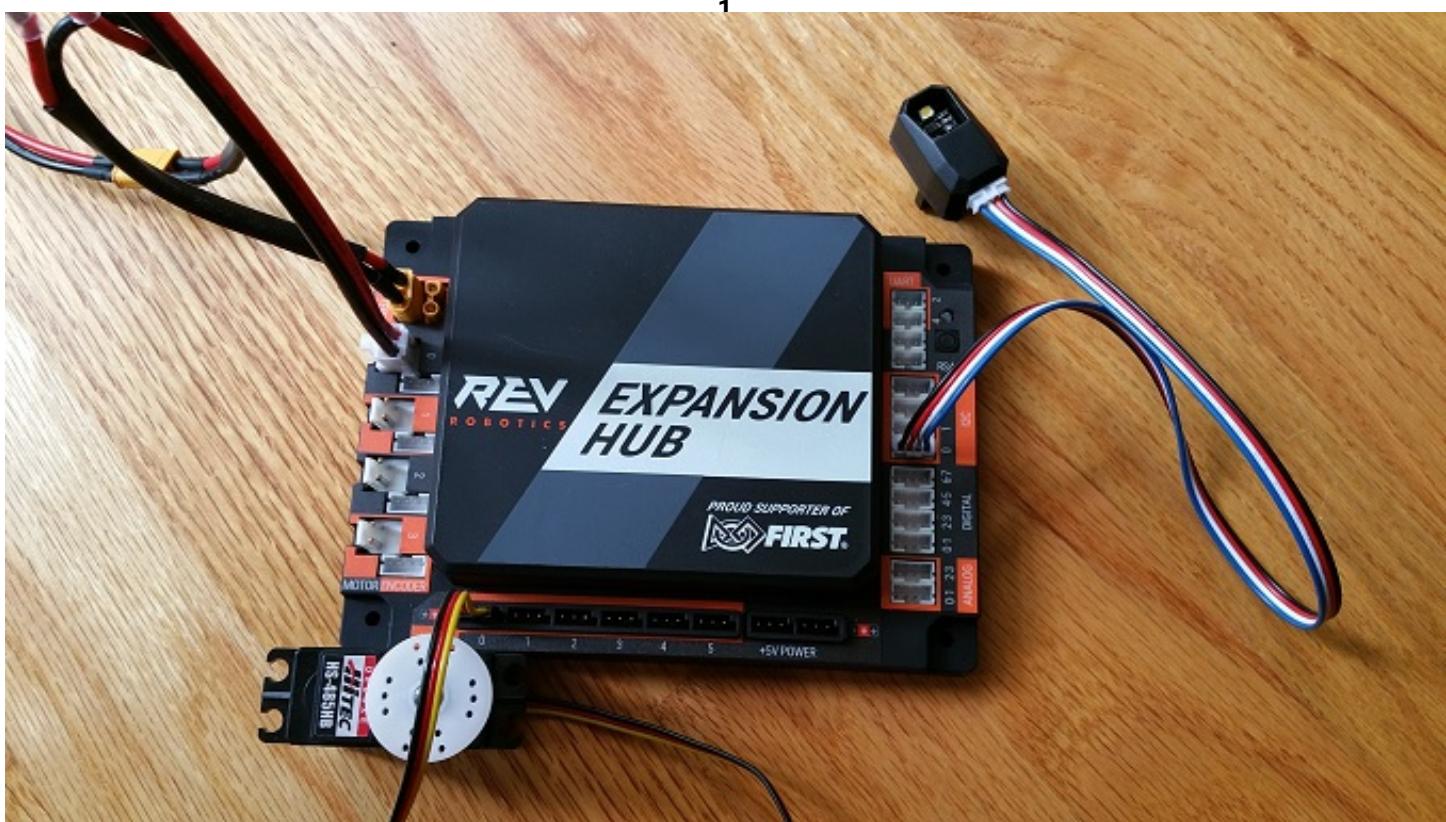
FTC Docs

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

FTC Programming Resources, 73



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



### 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.

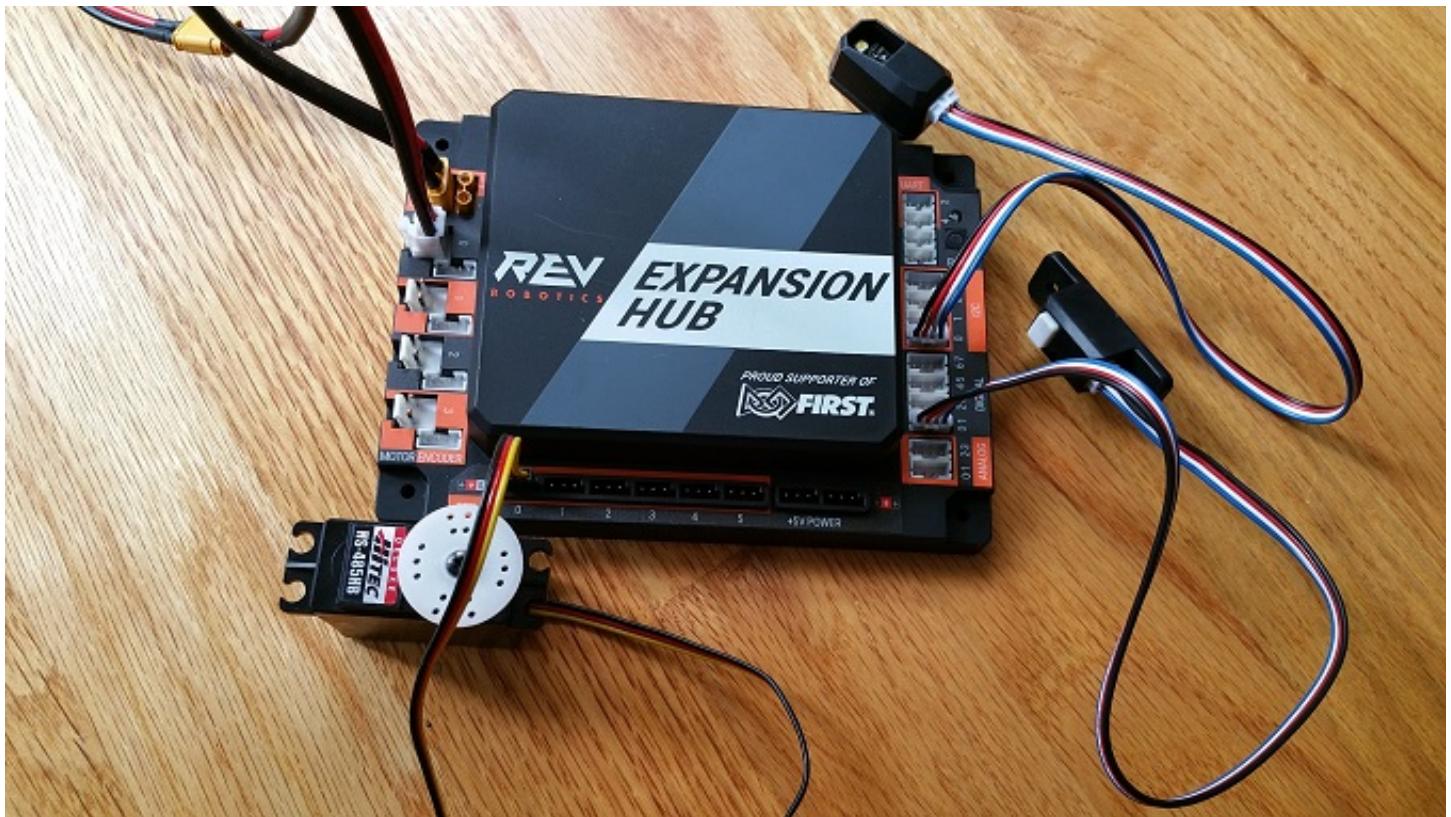
Note that it will take an estimated 2.5 minutes to complete this task.

## 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.



## Configuring Your Hardware

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

### Getting Started

#### Getting a Smartphone Robot Controller Ready

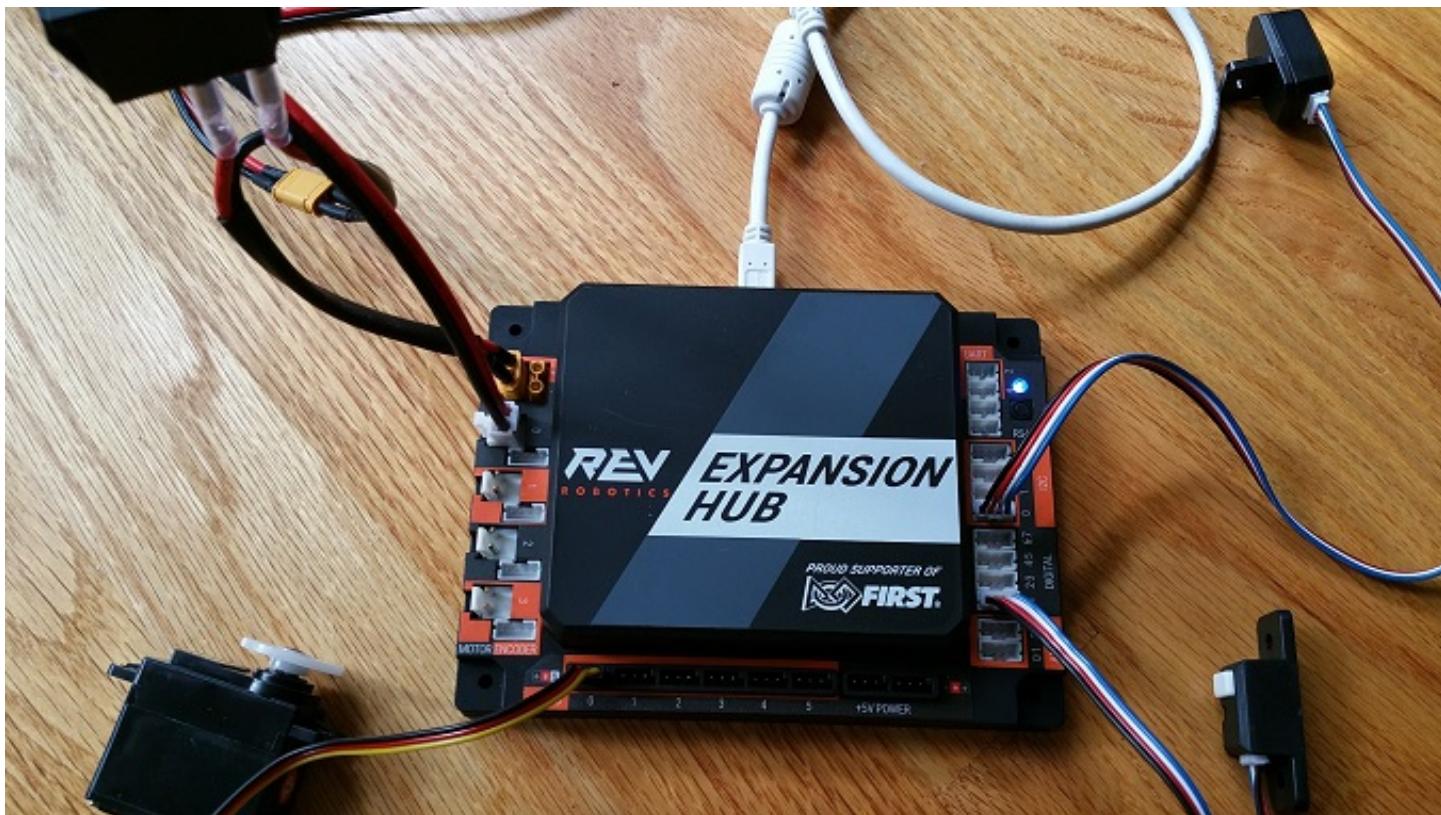
Before you can communicate with the motor, servo and sensors that are connected to the Control Hub or Expansion Hub, you first must create a configuration file on your Robot Controller, so that the Robot Controller will know what hardware is available on the Control Hub's or Expansion Hub's external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

#### Connecting an Android Smartphone to an Expansion Hub

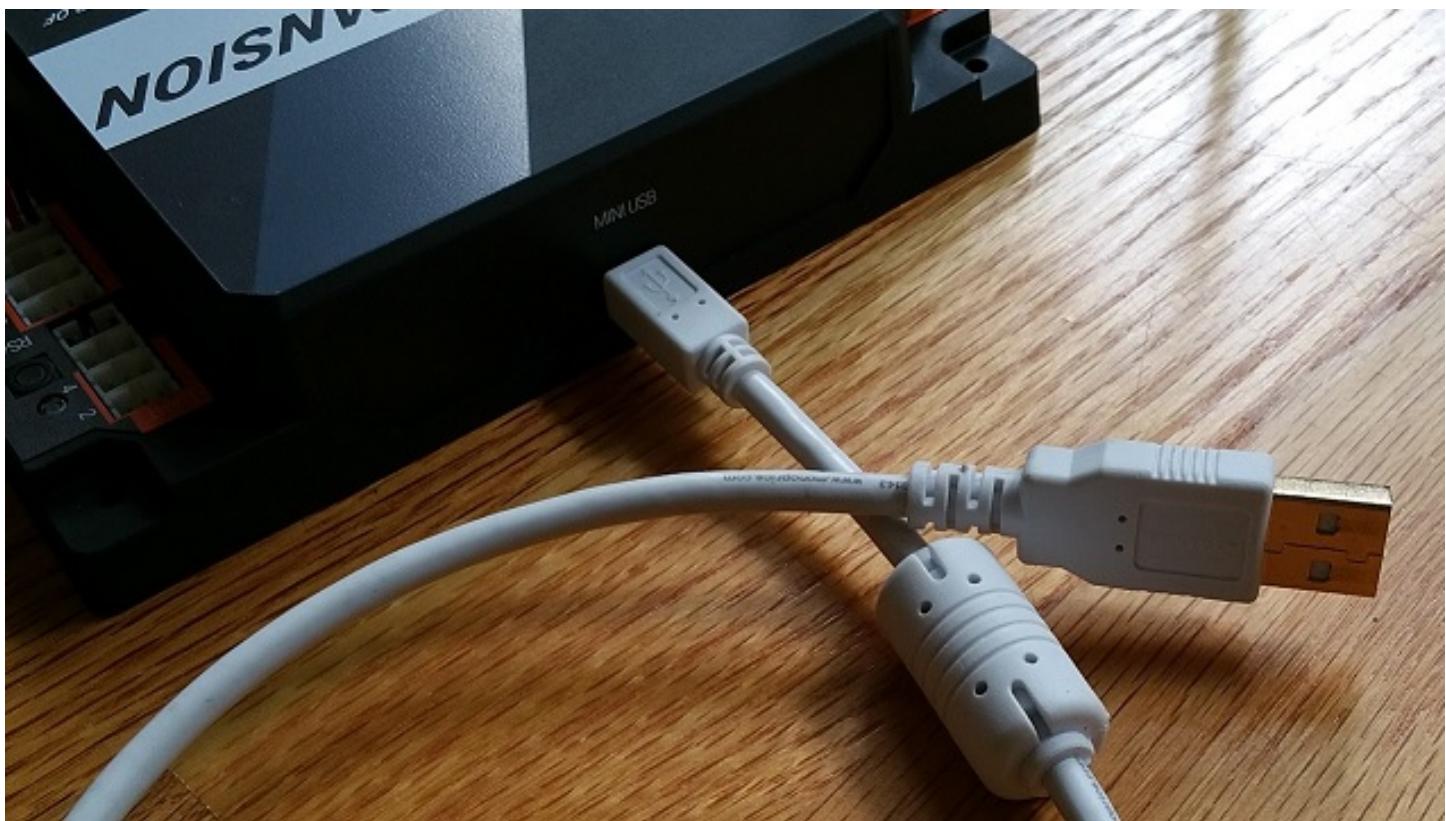
If you are using an Android smartphone as a Robot Controller, you must physically connect the Robot Controller smartphone to the Expansion Hub using a USB cable and an On-The-Go (OTG) adapter. Also, you should verify that the DRIVER STATION is currently paired to the Robot Controller.

#### Connecting an Android Smartphone to an Expansion Hub Instructions

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



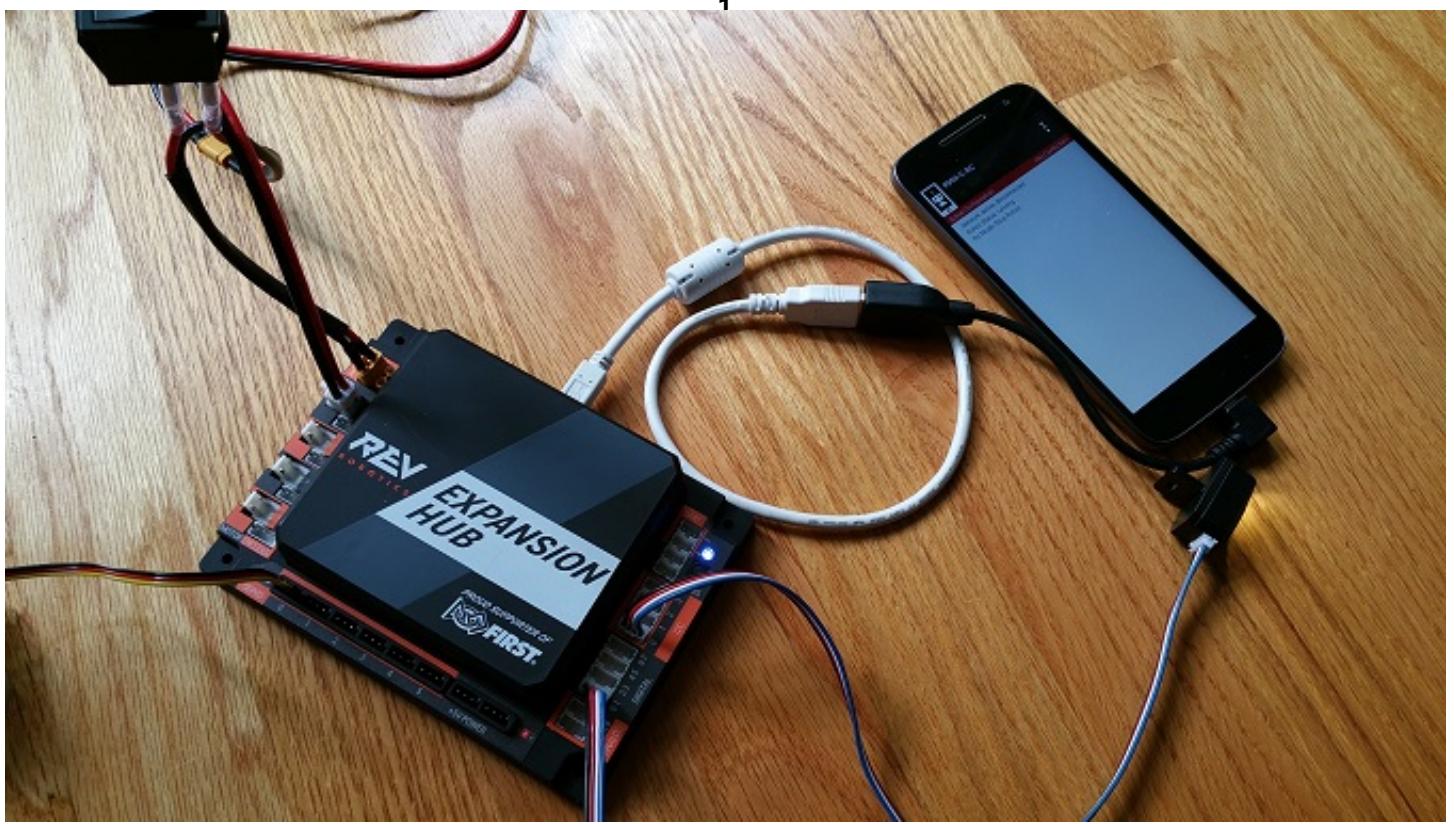
2. Plug the Type B Mini end of the USB cable into the USB mini port on the Expansion Hub.



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

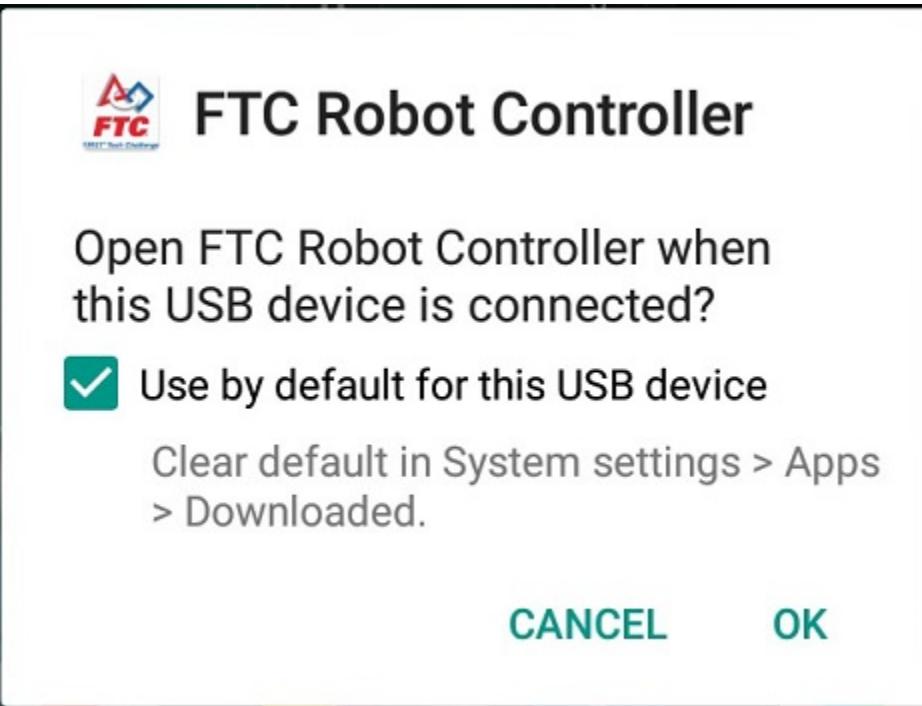


4. Verify that your Robot Controller smartphone is powered on and unlocked. Plug in the USB Micro OTG adapter into the OTG port of the Robot Controller phone.



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.

---

### 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.

## Creating a Configuration File Using the DRIVER STATION

FTC Docs

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.

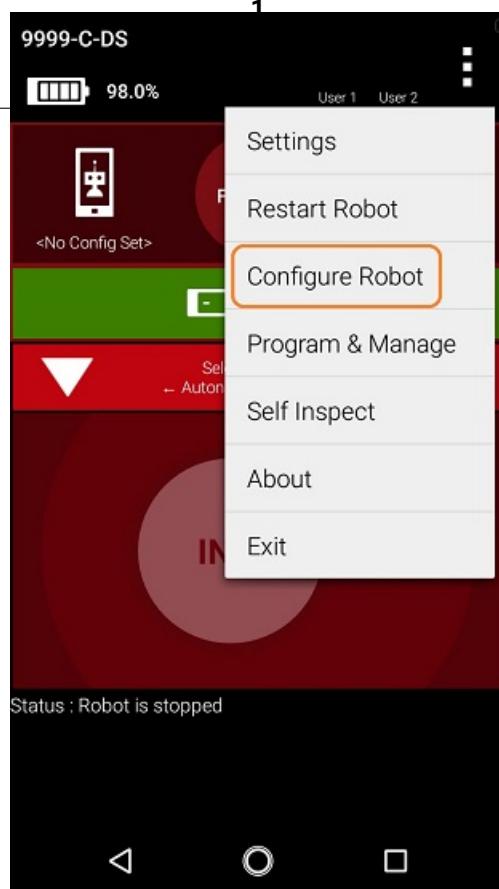
FTC Programming Resources, 81

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

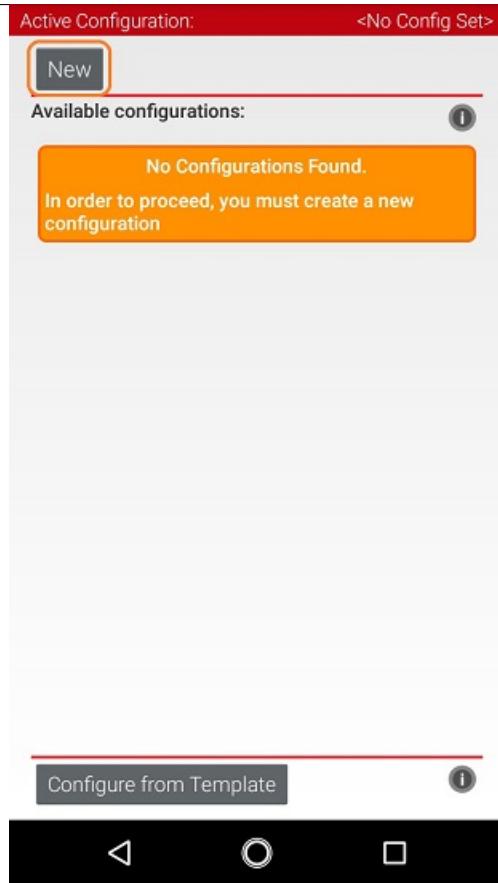
1. Touch the three vertical dots in the upper right hand corner of the Driver Station app. This will launch a pop-up menu.



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

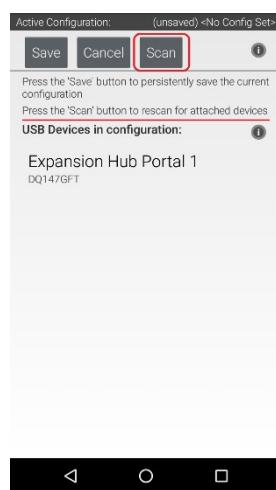


3. If your Robot Controller does not have any existing configuration files, the screen will display a message indicating that you need to create a file before proceeding.



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.

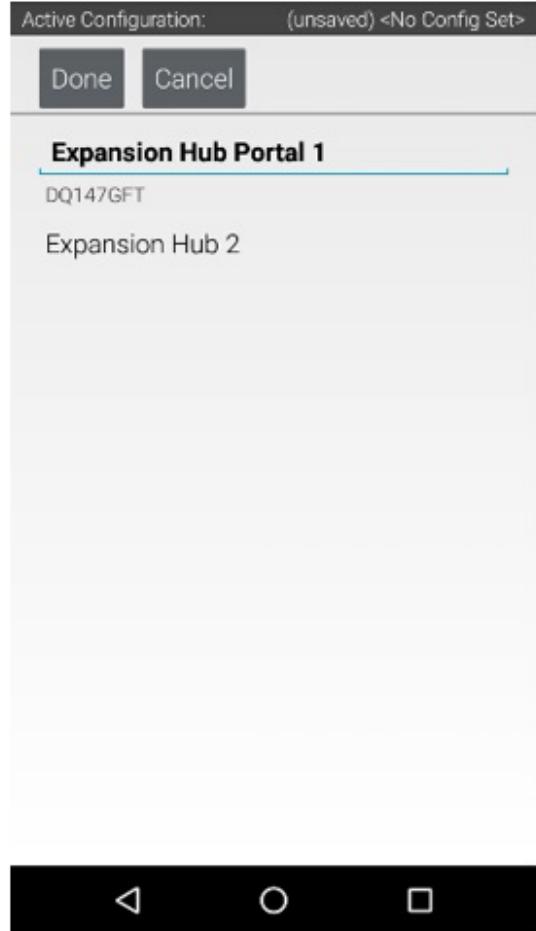
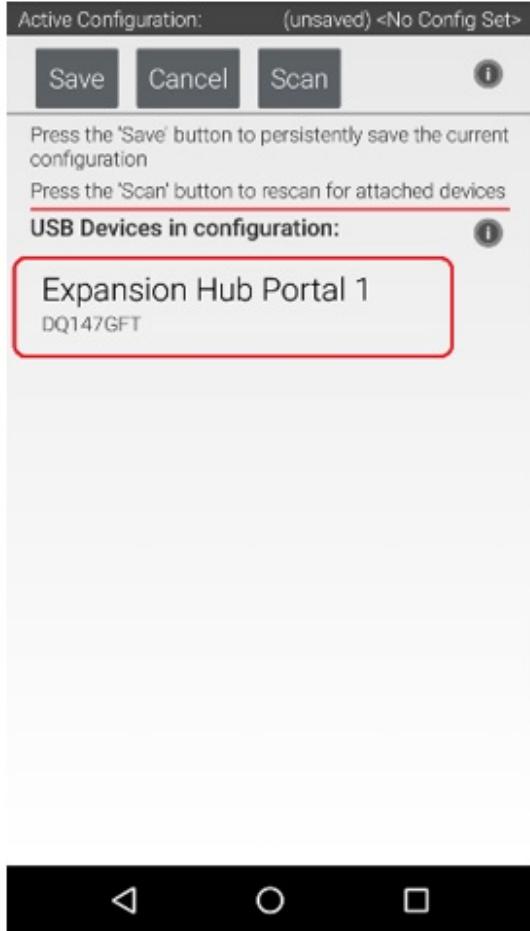
FTC Docs

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.

FTC Programming Resources, 84

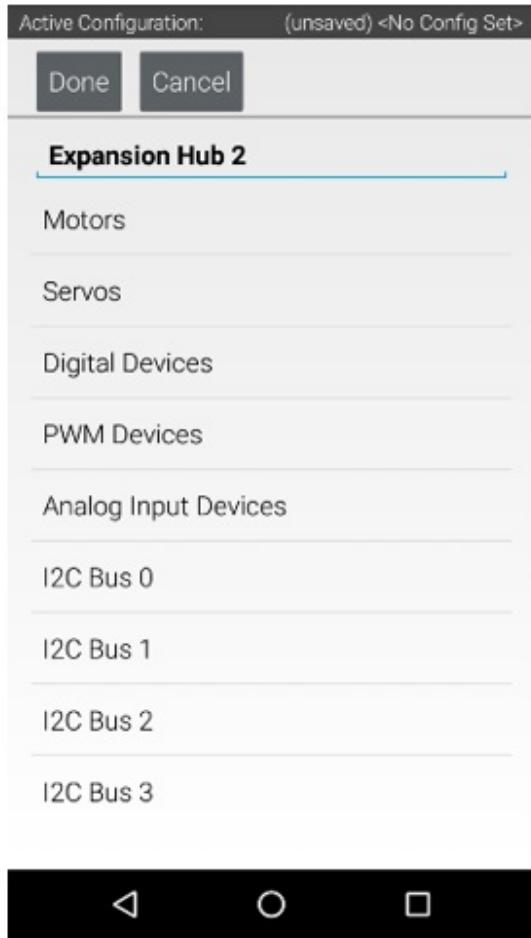
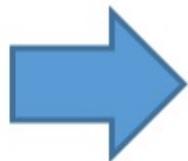
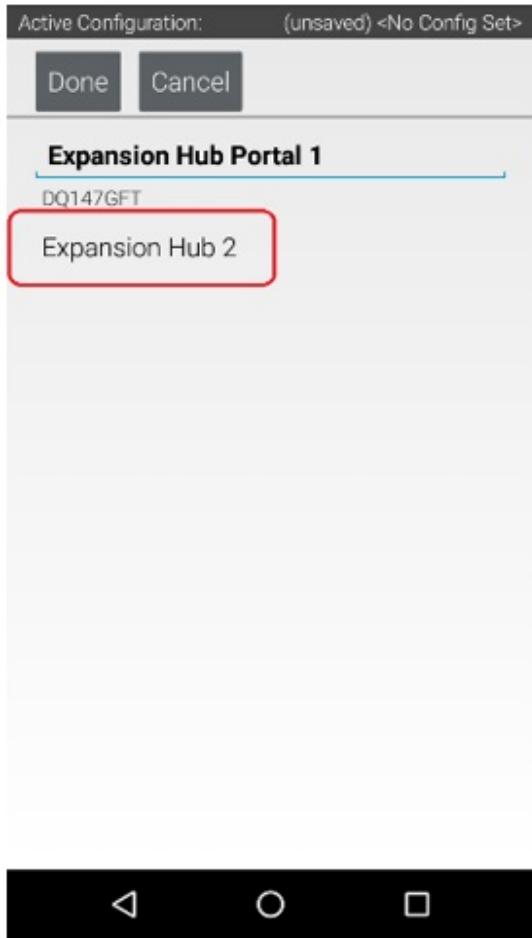
If you do not see your Expansion Hub Portal listed and you are using a smartphone as a Robot Controller, check the wired connections to make sure they are secure and then press the Scan button one or two times more to see if the smartphone detects the device on a re-scan of the USB bus.

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.

### Configuring a DC Motor

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

---

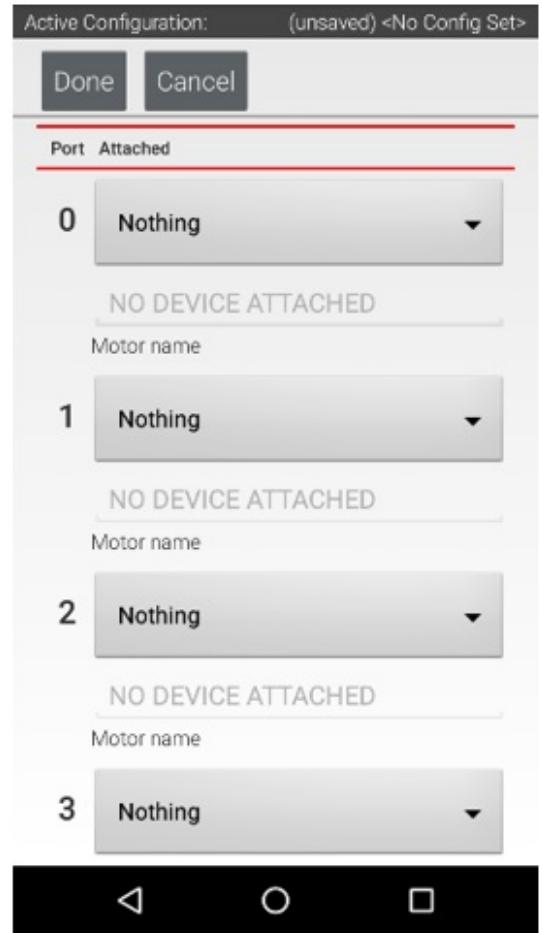
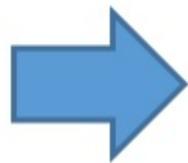
**Important:** At this point, although you have created your configuration file, you have not yet saved its contents to the Robot Controller. You will save the configuration file later in the [Saving the Configuration Information](#) step.

## Configuring a DC Motor Instructions

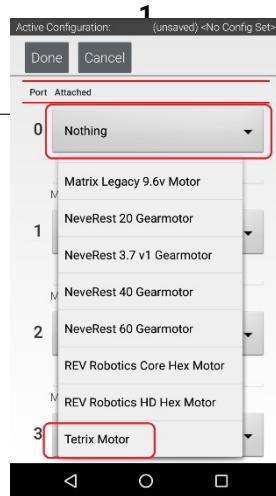
FTC Docs

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

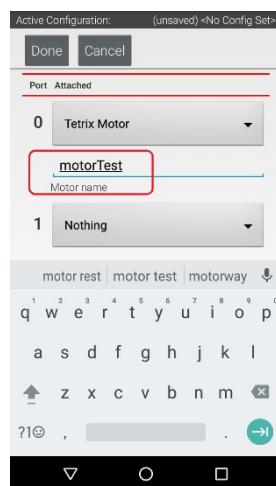
FTC Programming Resources, 86



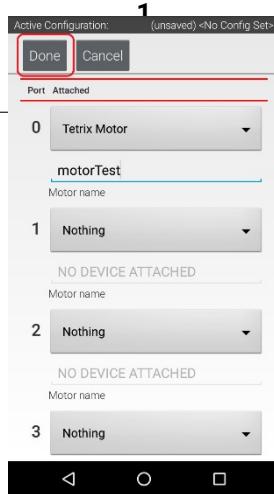
2. Since we installed our motor onto port #0 of the Expansion Hub, use the dropdown control for port 0 to select the motor type (Tetrix Motor for this example).



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.

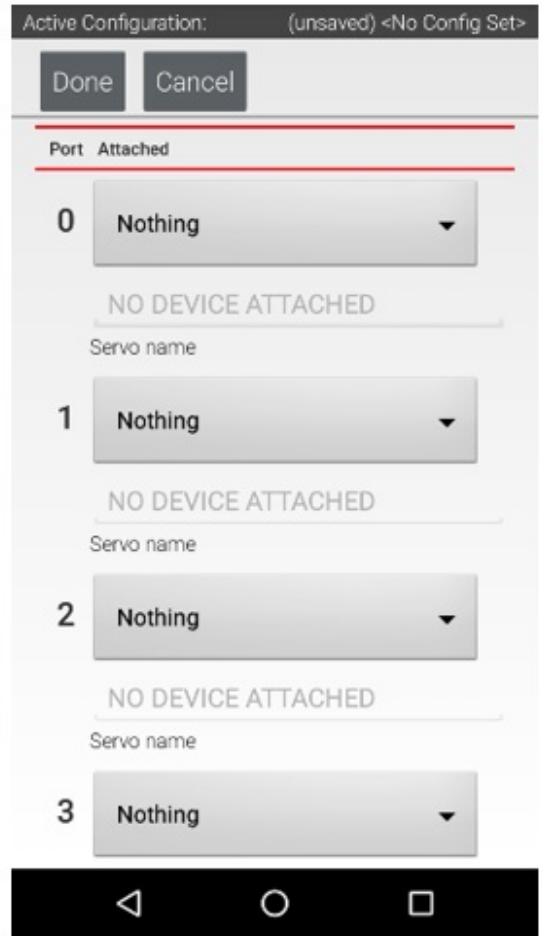
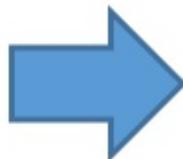


## Configuring a Servo

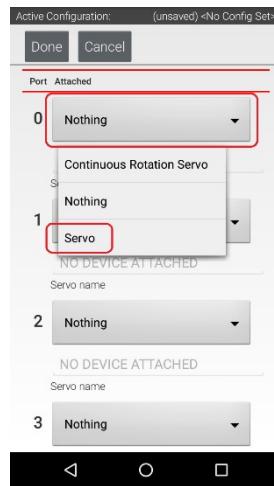
You will also want to add a servo to the configuration file. In this example, you are using a standard 180-degree servo.

### 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.



## Configuring a Color Distance Sensor

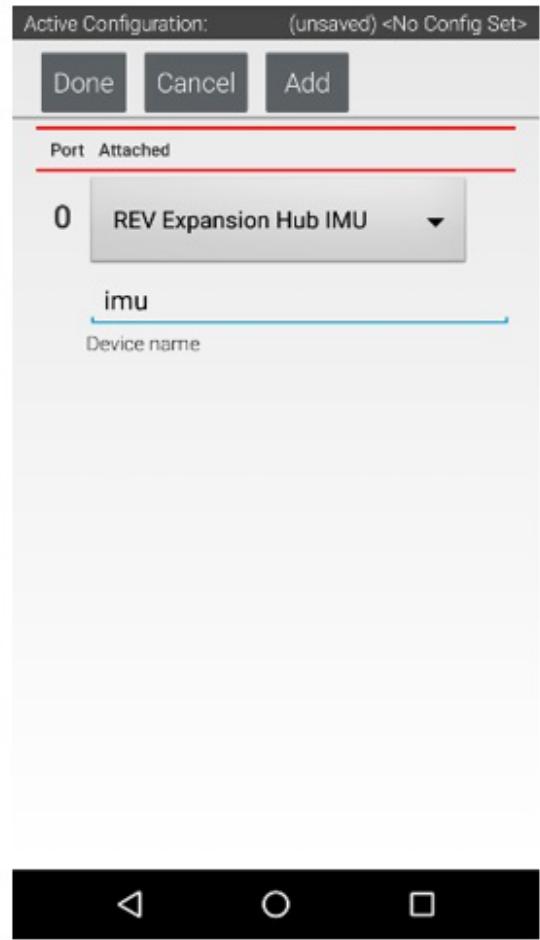
The REV Robotics Color Distance Sensor is an I<sup>2</sup>C sensor. It actually combines two sensor functions into a single device. It is a color sensor, that can determine the color of an object. It is also a distance or range sensor, that can be used to measure short range distances. Note that in this tutorial, the word "distance" is used interchangeably with the word "range".

## Configuring a Color Distance Sensor Instructions

FTC Docs

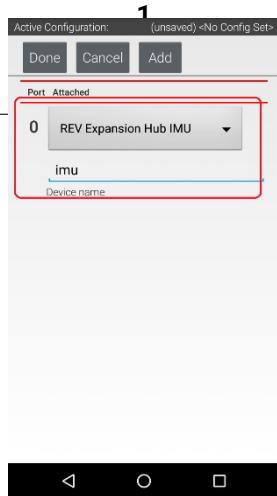
1. Touch the words **I2C Bus 0** on the screen to launch the I2C configuration screen for this I2C bus.

[FTC Programming Resources, 91](#)



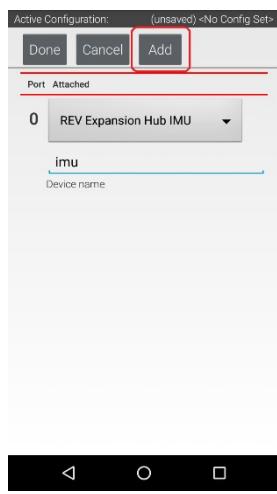
The Expansion Hub has four independent I2C buses, labeled "0" through "3". In this example, since you connected the Color Sensor to the port labeled "0", it resides on I2C Bus 0.

2. Look at the **I2C Bus 0** screen. There should already be a sensor configured for this bus. The 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.

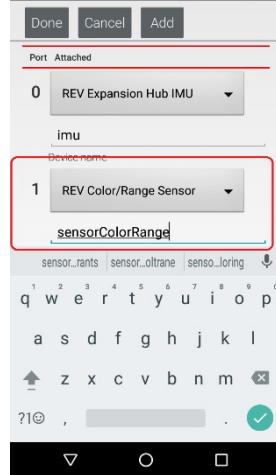


The built-in IMU is internally connected to I2C Bus 0 on each Expansion Hub. Whenever you configure an Expansion Hub using the Robot Controller, the app automatically configures the IMU for I2C Bus 0. You will need to add another I2C device for this bus to be able to configure the color sensor.

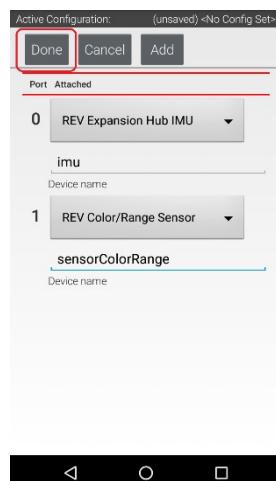
3. Press the **Add** button to add another I2C device to this bus.



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.



## Configuring a Digital Touch Sensor

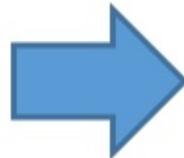
~~FTC Docs~~

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.

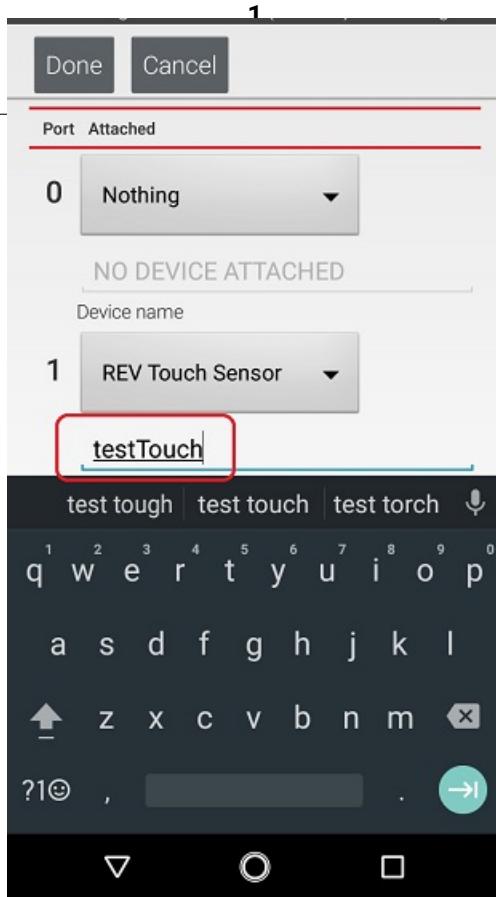
**FTC Programming Resources, 94**

### Configuring a Digital Touch Sensor Instructions

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

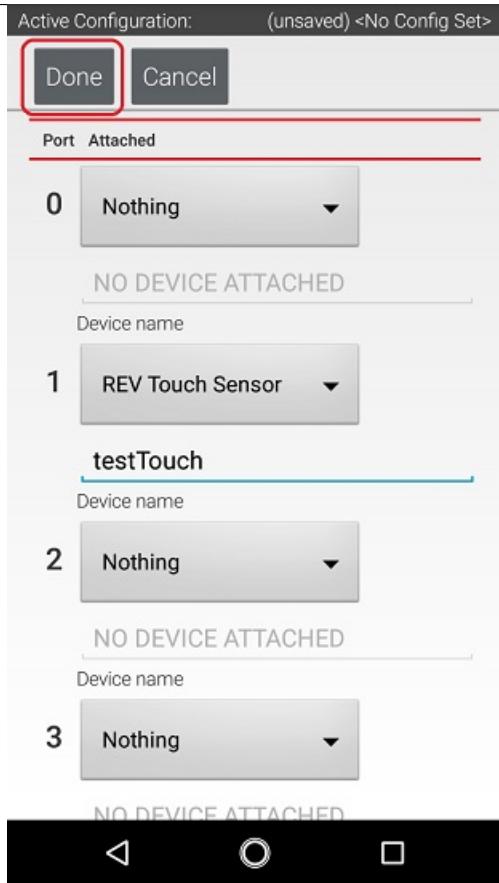


2. Use the touch screen to add a "REV Touch Sensor" for port #1 and name the device "testTouch".



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.



## Configuring an External UVC Camera and a Powered USB Hub

### Introduction

Game rules and in the Game Manual Part 1 have been modified to allow the use of USB Video Class (UVC) Compatible Cameras for computer vision-related tasks. Teams have the option of using an externally connected camera instead of the Android smartphone's 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).



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 rule c(iii) permits the use of a powered USB hub to make this connection. 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 USB 5V Battery Pack.
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](https://www.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.

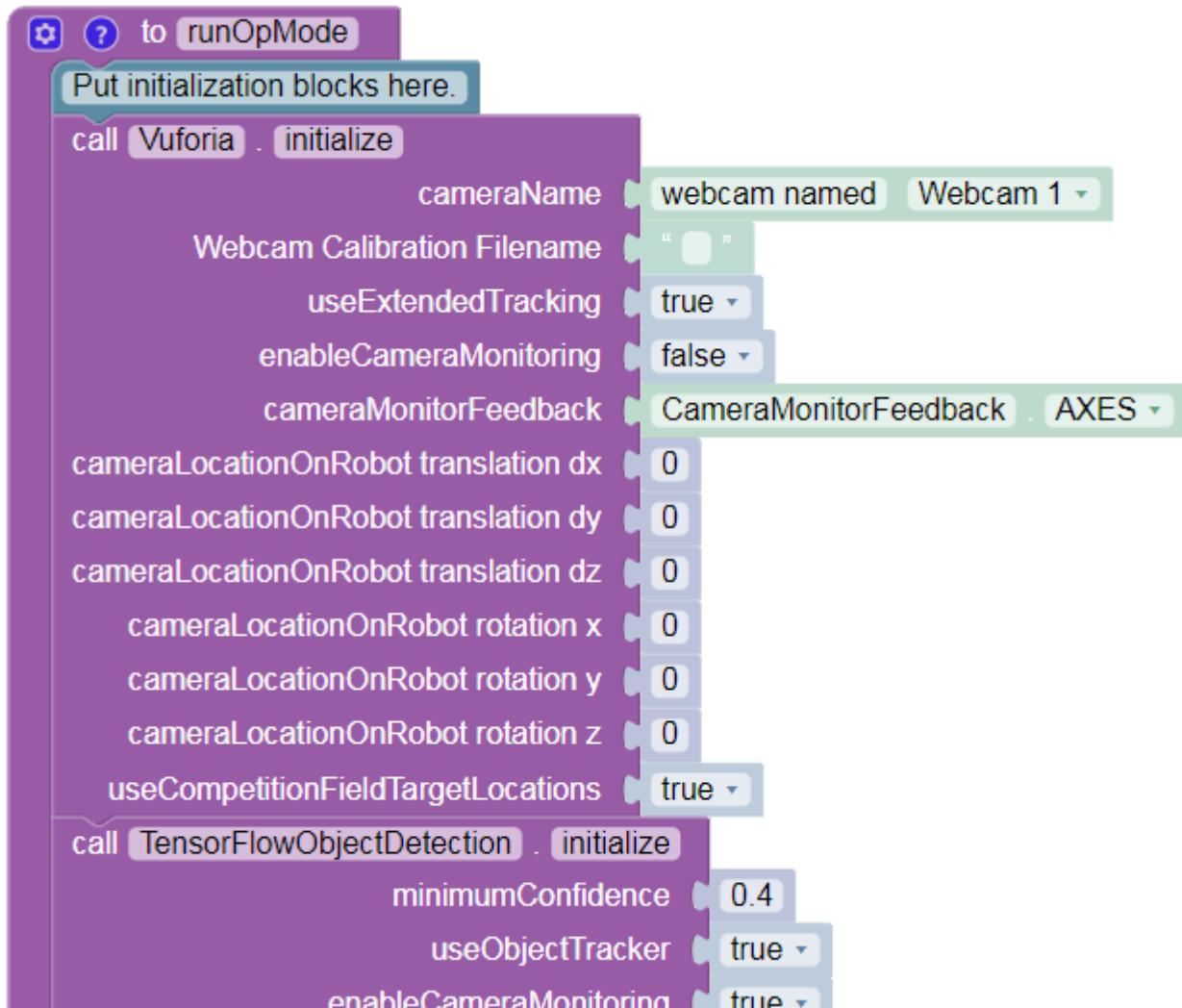


Note that teams can create this special cable using one end of a servo extension cable (to plug into the 5V aux port) and one end of a Micro USB cable (to plug into the Anker hub's power port). **Creating this cable is an advanced task and should only be attempted by teams who have guidance from an adult 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.



## Configuring an External Webcam with a Control Hub

### Introduction

The Game Manual Part 1 allows USB Video Class (UVC) cameras for robot vision tasks. 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 Johannessen 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

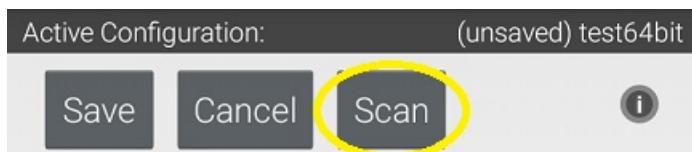
The UVC camera plugs directly into the USB 2.0 port on the REV Control Hub. Unlike the REV Expansion Hub, there is no need for an external powered USB hub.



## 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".



Press the 'Save' button to persistently save the current configuration

Press the 'Scan' button to rescan for attached devices

### USB Devices in configuration:



Webcam 1

0B5E63D0

Expansion Hub Portal 1

(embedded)

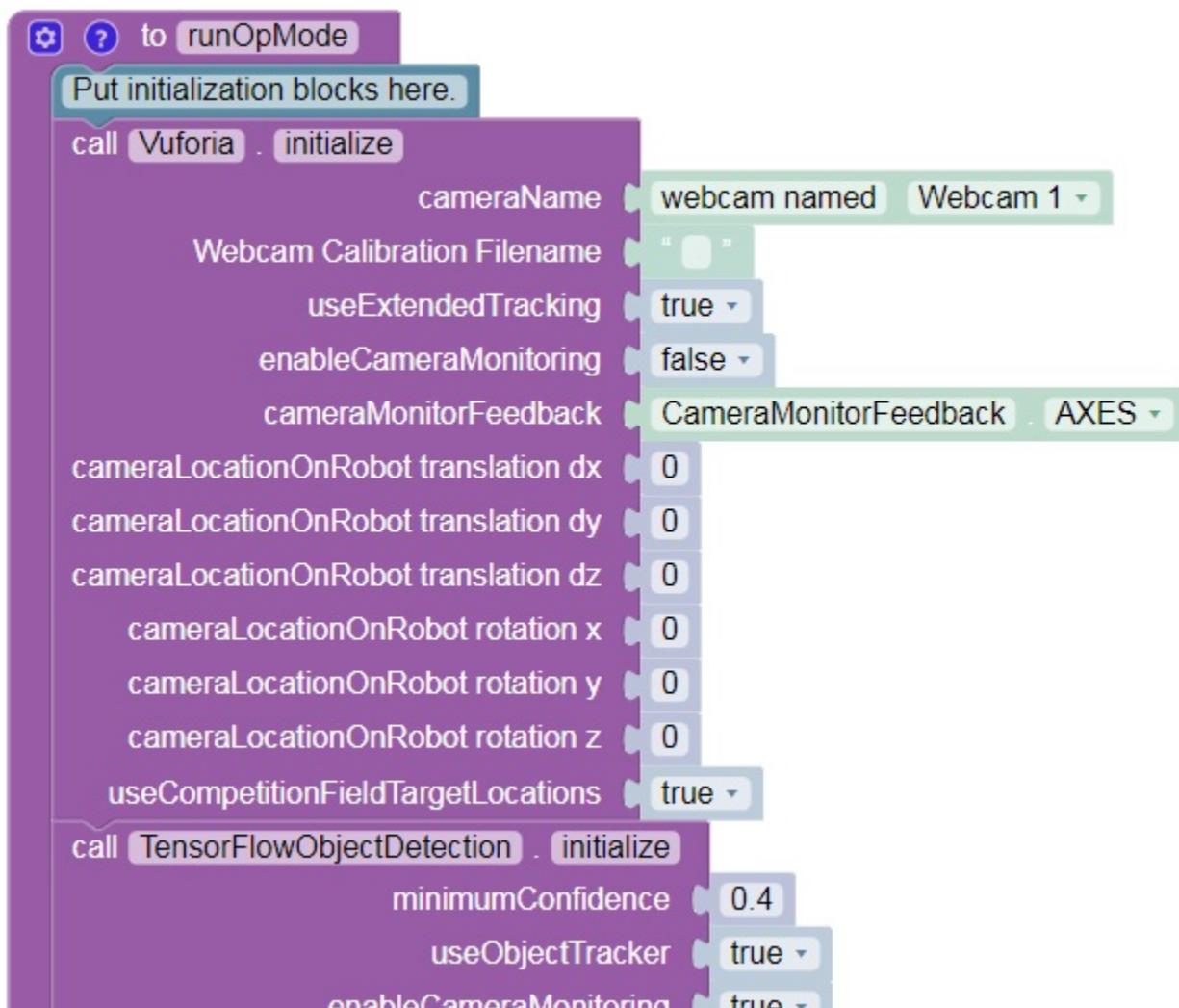


You can keep this default name (the sample Op Modes reference this name) or change it. If you change the webcam name, make sure your Op Modes refer to this new name.

## 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 Vuforia or TensorFlow operations.

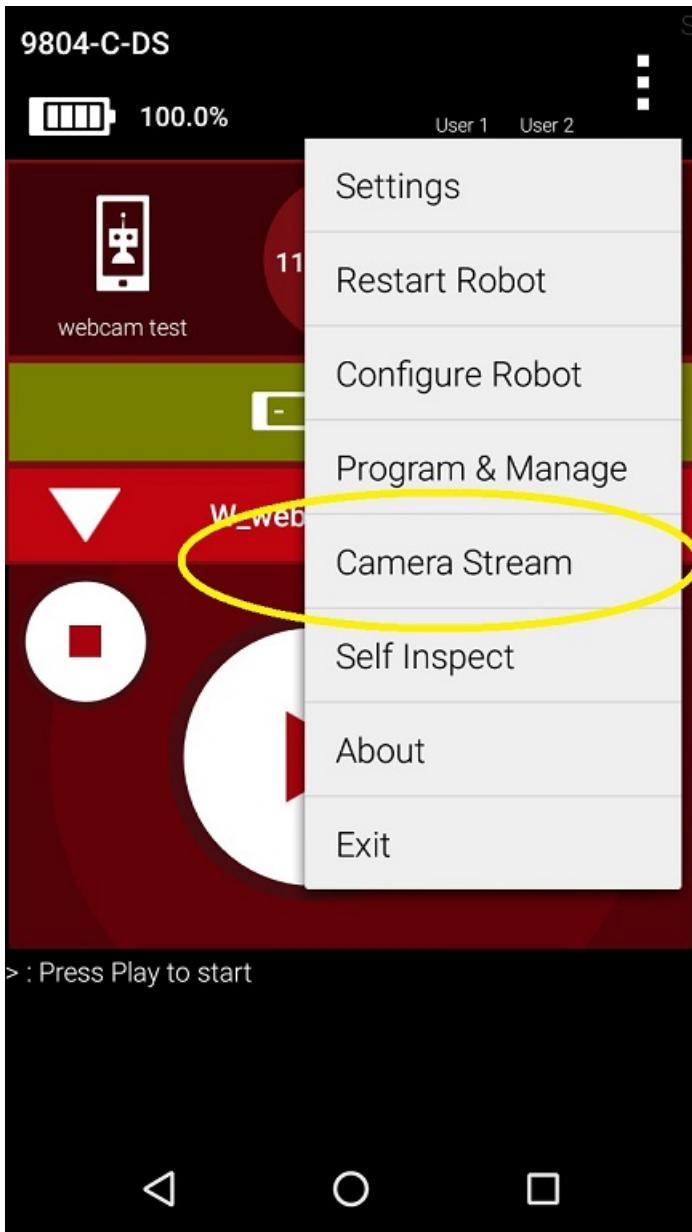


Before opening and editing an Op Mode, verify that the intended configuration (with camera) is active. Also verify that the name referenced in the Op Mode matches the name specified in the configuration file.

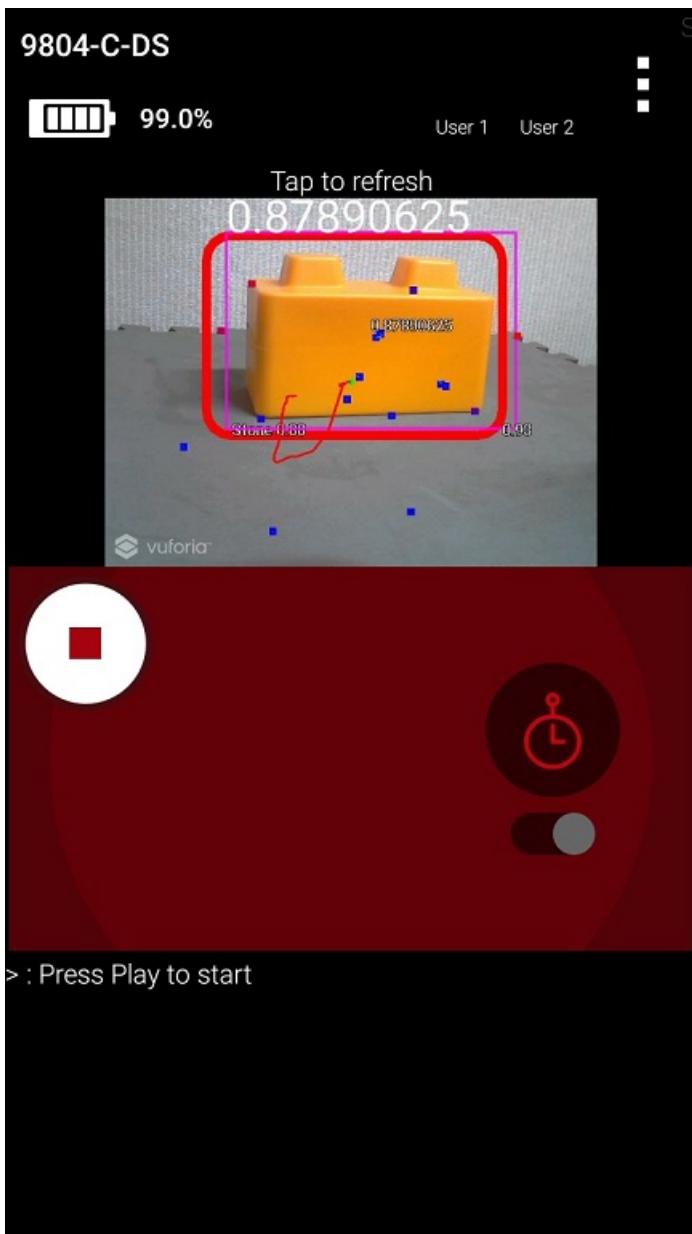
## Image Preview

The FIRST Tech Challenge apps provide camera preview for ‘stream-enabled’ Op Modes using Vuforia or TensorFlow Object Detection (TFOD).

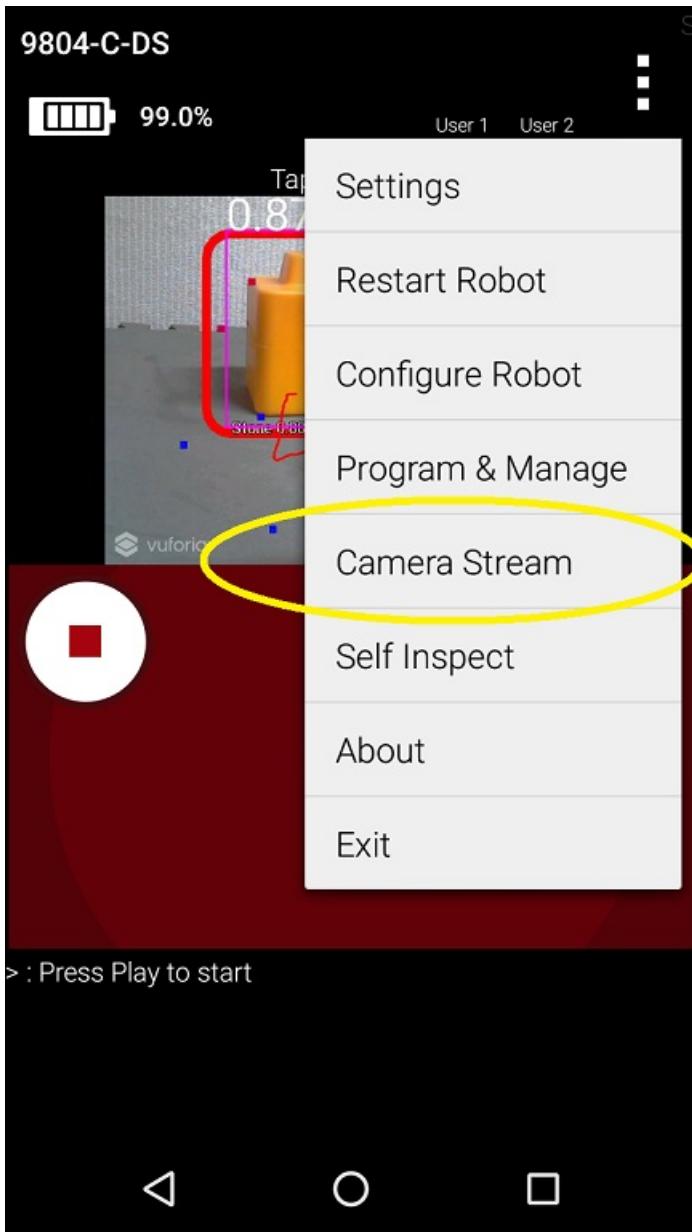
On a paired DRIVER STATION device, with the camera connected and configured, select a stream-enabled Op Mode. Press the INIT button, and wait briefly for streaming software to initialize; do not press the START button. Instead open the main menu (the 3 dots in upper right hand corner of the screen) and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



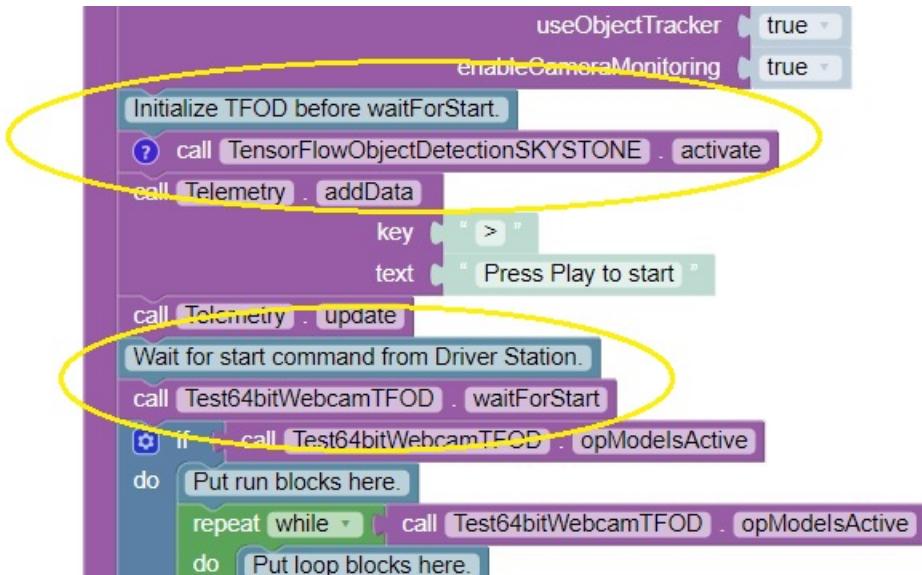
The camera image will appear on the DRIVER STATION screen. Manually touch the image to refresh it. To preserve bandwidth, only one frame is sent at a time.



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.



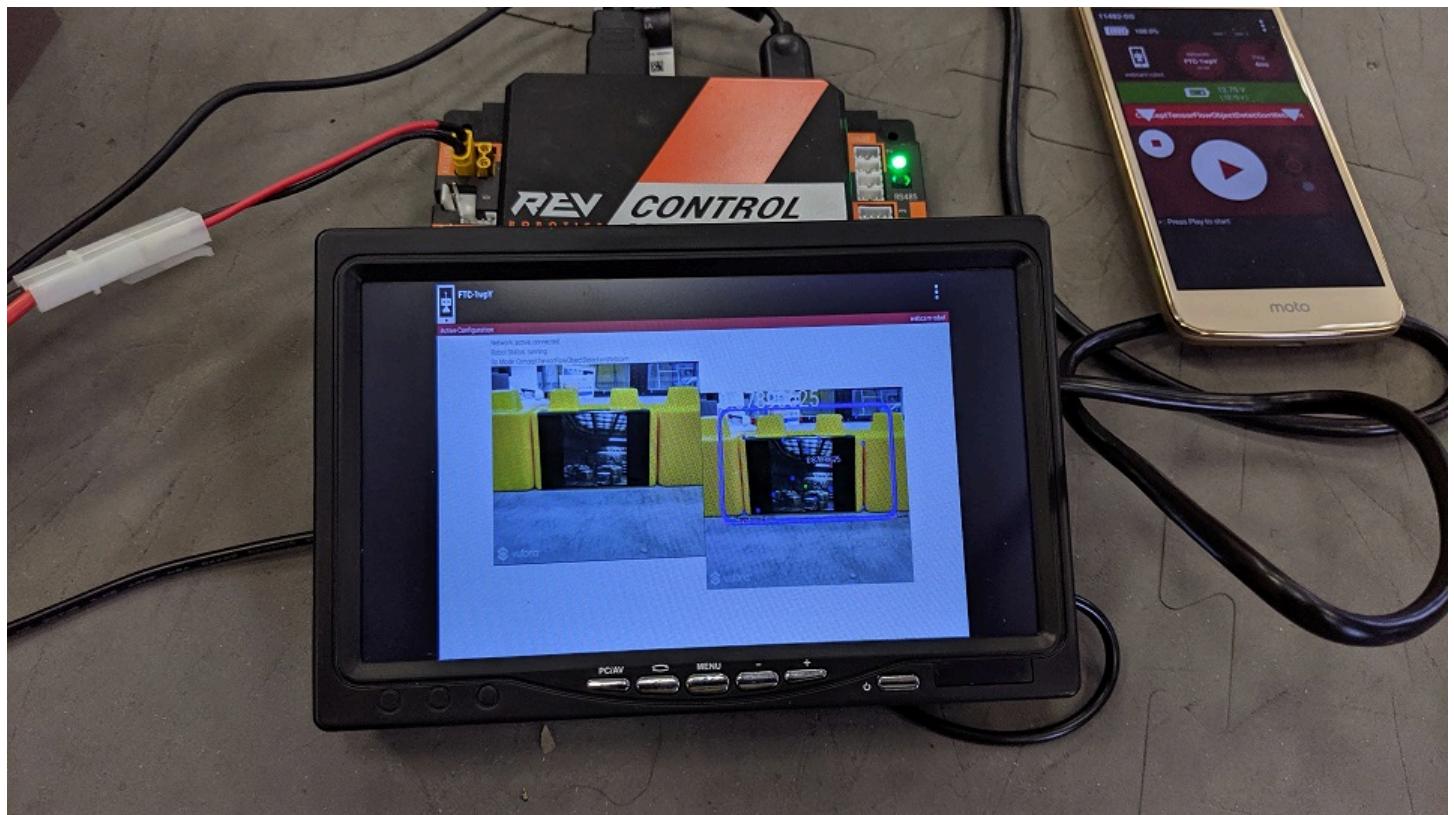
**Important Note:** Because the Camera Stream feature is only available during the INIT phase of an Op Mode, you must ensure that the Vuforia library is activated in your Op Mode **before** the waitForStart command:



If you do not see the Camera Stream option in your main menu on your DRIVER STATION, then verify that the Vuforia function 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.

### 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.

For custom streams, advanced users of Android Studio may consult the [API documentation](#) for `CameraStreamClient`, `CameraStreamServer` and `CameraStreamSource` classes.

## Using Two Expansion Hubs

### 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 a second Expansion Hub to your first Hub to add more I/O ports.

This document describes how to connect and configure two Expansion Hubs for use in the FIRST Tech Challenge. Note that the FIRST Tech Challenge Competition Manual limits the maximum number of Expansion Hubs on a single robot to two.

**Important Note:** This document describes the process for setting up a smartphone Robot Controller with two Expansion Hubs. Control Hubs have a reserved address, so you do not need to worry about an Expansion Hub's address when it is the only Expansion Hub connected to a Control Hub. However, the process for physically connecting and configuring them is the same.

### Equipment Needed

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

Required Item(s)	Image
Two (2) FIRST-approved Android smartphones. One should have the Robot Controller app installed and the other should have the Driver Station app installed. For a list of FIRST-approved Android smartphones, refer to the current Game Manual Part 1, rule <RE06>.	
USB Type A male to type mini-B male cable.	
Micro USB OTG adapter.	

continues on next page

Table 6 – continued from previous page

Required Item(s)	Image
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 Game Manual Part 1, rule <RE03>.	
Two(2) REV Robotics Expansion Hubs (REV-31-1153).	
REV Robotics (or equivalent) 3-Pin JST PH Cable (REV-35-1414, 3 pack shown but only one needed).	

continues on next page

Table 6 – continued from previous page

Required Item(s)	Image
REV Robotics XT30 Extension Cable (REV-31-1394).	

## 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

Show 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

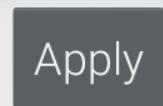
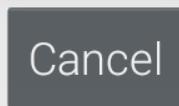
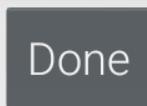
### Expansion Hub Firmware Update

Updates the firmware all currently attached Expansion Hubs

### Expansion Hub Address Change

Change the persistent hub address of one or more Expansion Hubs





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.

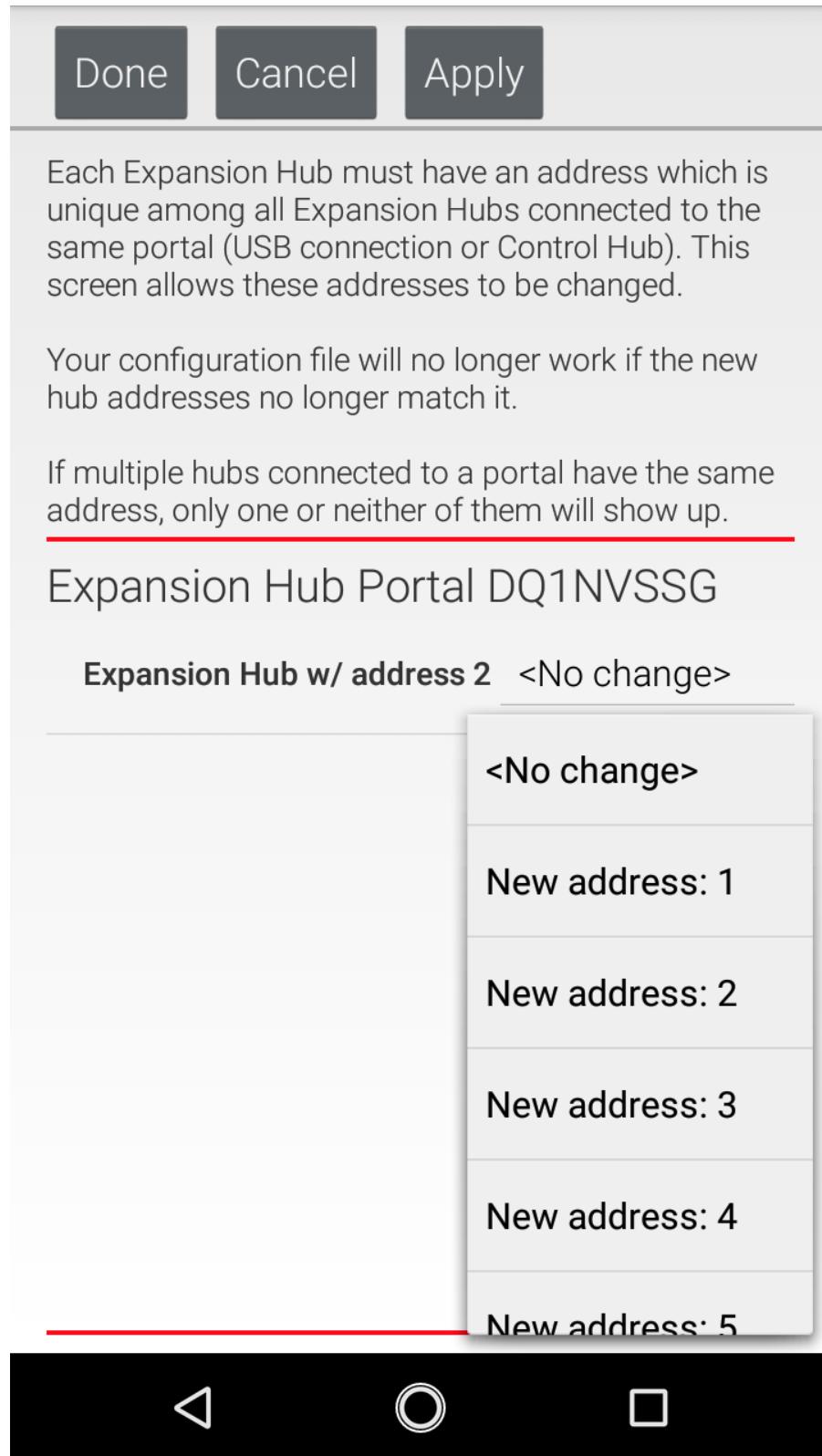
Your configuration file will no longer work if the new hub addresses no longer match it.

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>**





## 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**

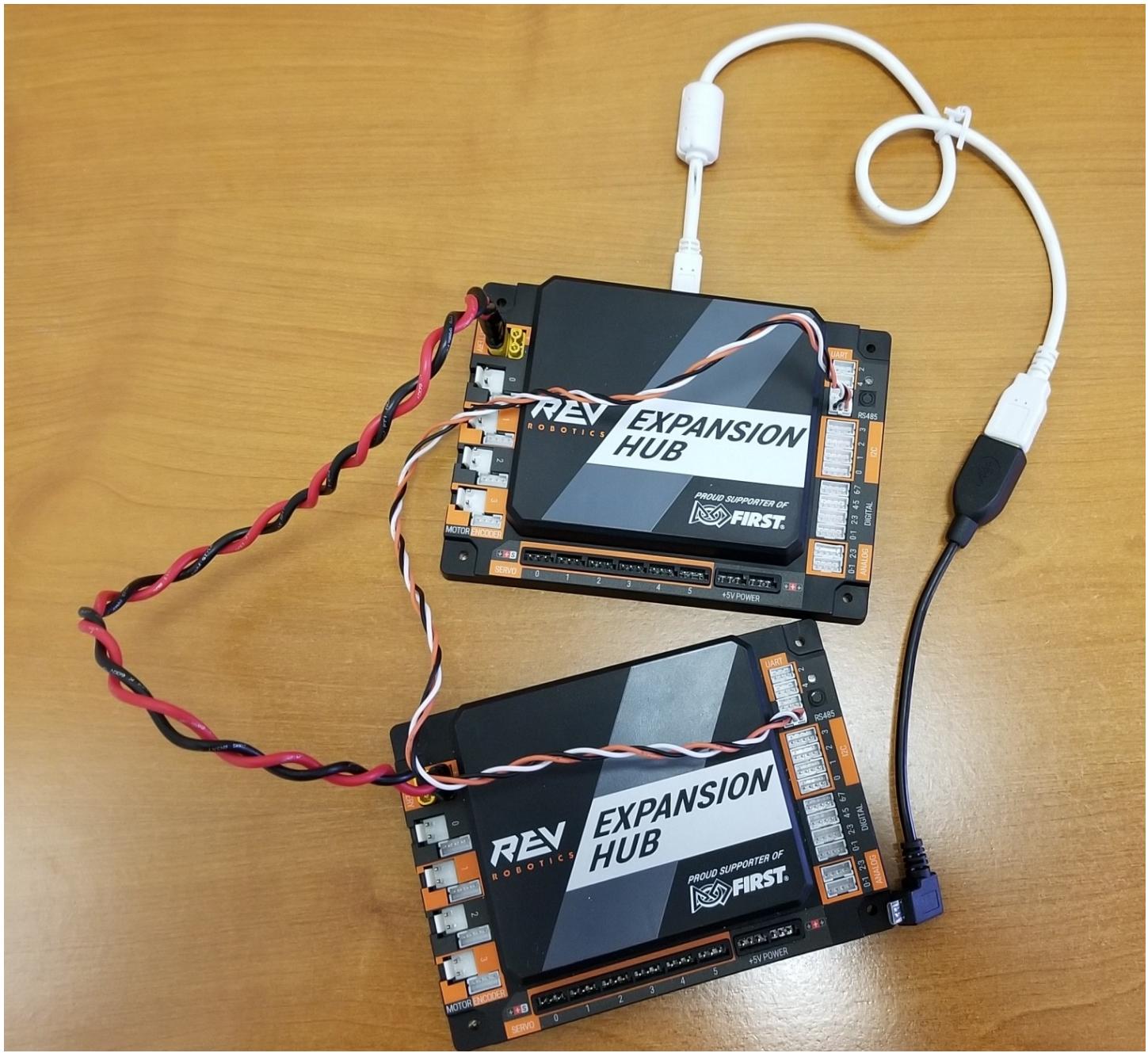
1. After you have changed the address of one of the Hubs, you can use the 3-pin JST PH cable and the XT30 cable to daisy chain the two Hubs together. Before you do this, disconnect the 12V battery and power switch from the first Expansion Hub.

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.



2. 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.





3. 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.

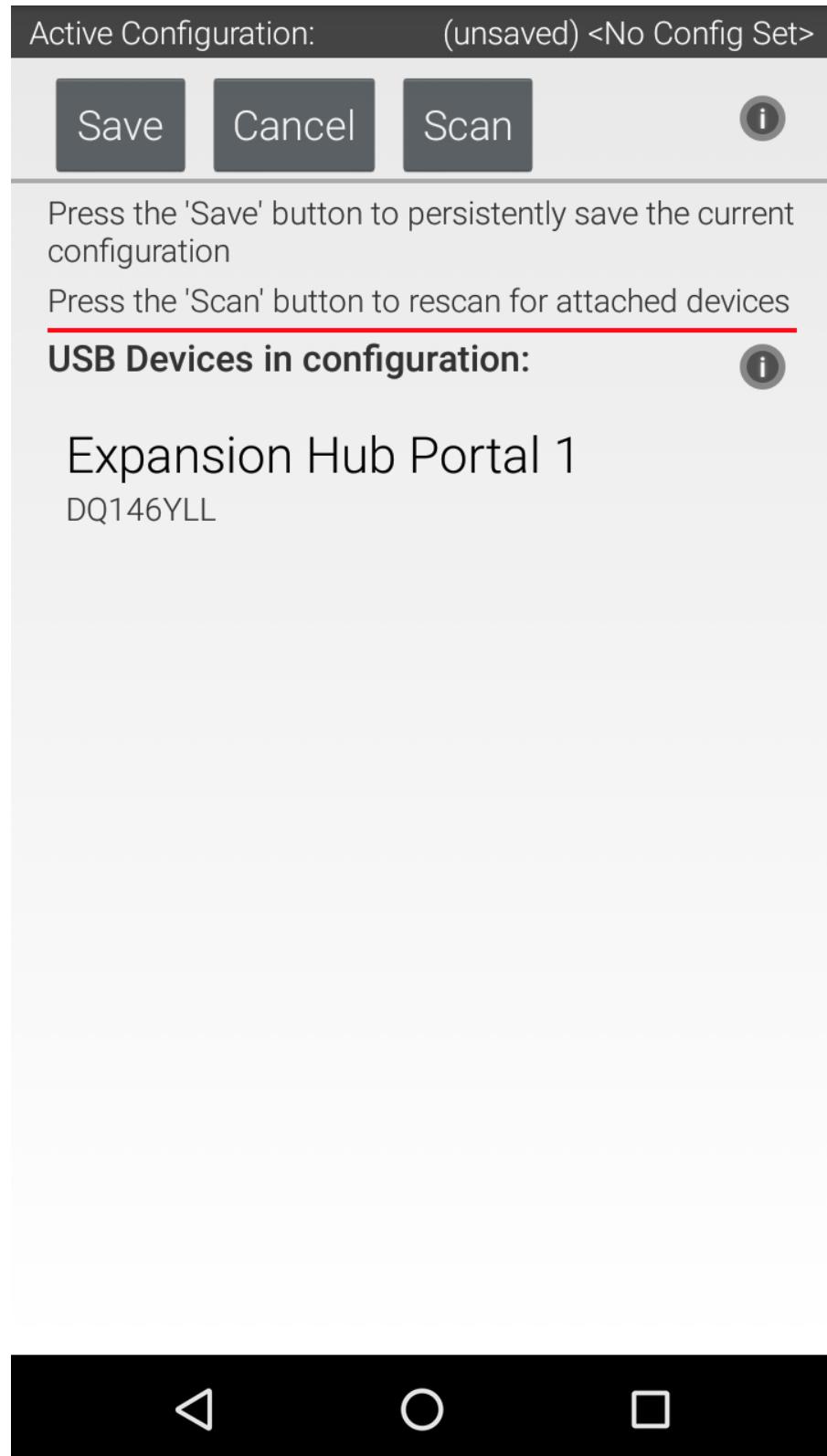
Connect the Robot Controller and select the Configure Robot option from the Settings menu. Press the New button to create a new configuration file. When you first scan for hardware, your Robot Controller should detect the Expansion Hub that is immediately connected to the Robot Controller via the OTG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

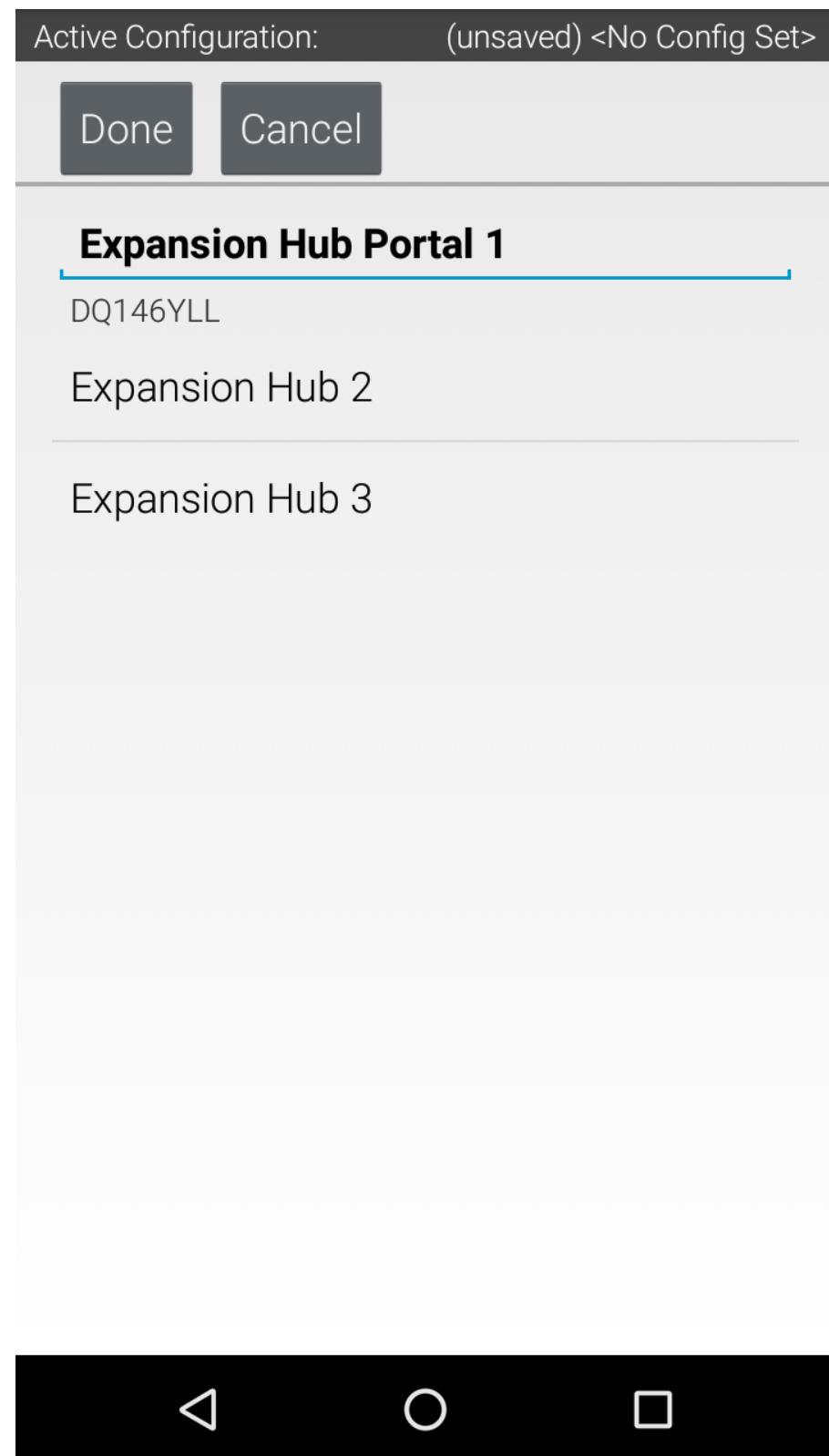
If you click on the Portal item in the configuration screen, you should see two Expansion Hubs listed, each with their respective addresses as part of their default device name.

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.







## Managing Electrostatic Discharge Effects

### 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 Sapiro, Patricia Strones, and David Thompson) for assisting in testing some of these mitigation techniques under the hot desert sun!

### 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.

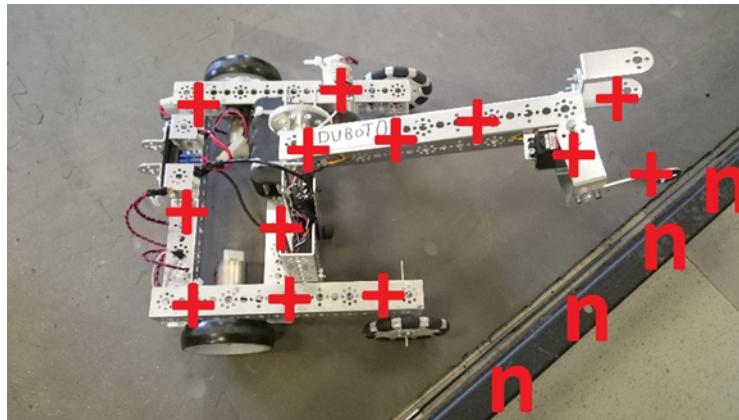


Fig. 1: Positively charged robot next to neutral field wall.

### 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.

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.

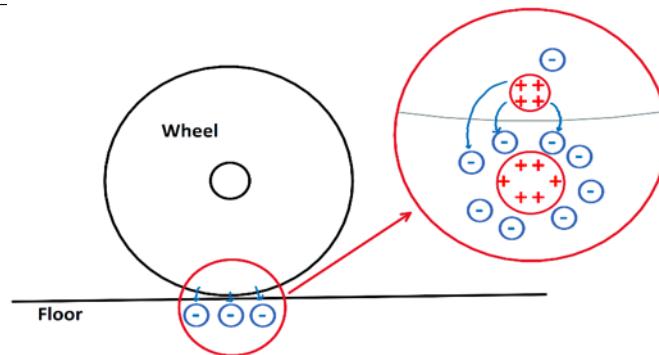


Fig. 2: Robots become charged due to the triboelectric effect.

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.

### 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.

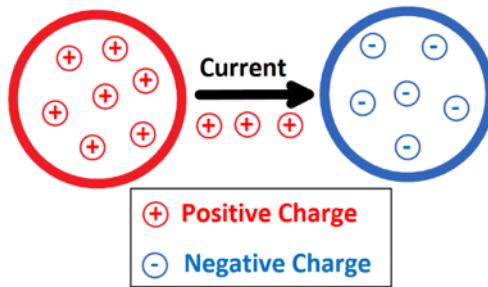


Fig. 3: Electric arc between two spheres of opposite charge.

### What Steps can be Taken to Mitigate the Risk of an ESD Disruption?

#### 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 they 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).



**Gracious Professionalism® - "Doing your best work while treating others with respect and kindness - It's what makes FIRST, first."**

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.

## 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.



Fig. 4: A snap-on ferrite choke.

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

## 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.



### 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.

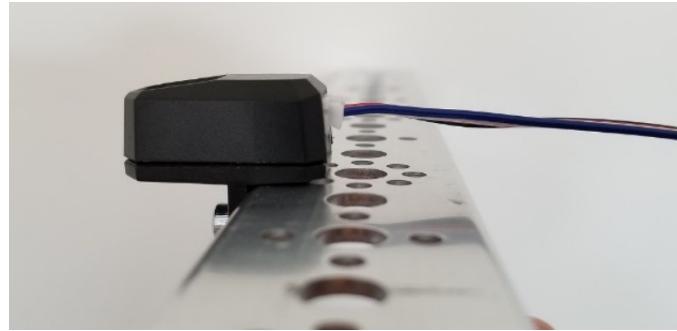


Fig. 6: Keep exposed portions of the electronics more than 3/8" (10mm) away from the frame.

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. 7: Electrical tape can be used to insulate exposed or poorly insulated metal.

### 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.

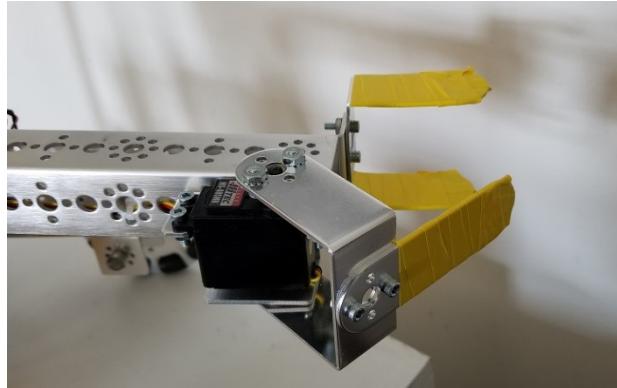


Fig. 8: Insulating portions of the robot that touch other metallic objects on the field can help.

In past seasons, teams who have done this have observed reductions in the frequency and severity of ESD events on their robots.

#### 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.



Fig. 9: The REV Resistive Grounding Strap (REV-31-1269) is an approved grounding cable.

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



Fig. 10: The REV-31-1385 adapter is approved for use with REV's Resistive Grounding Strap.

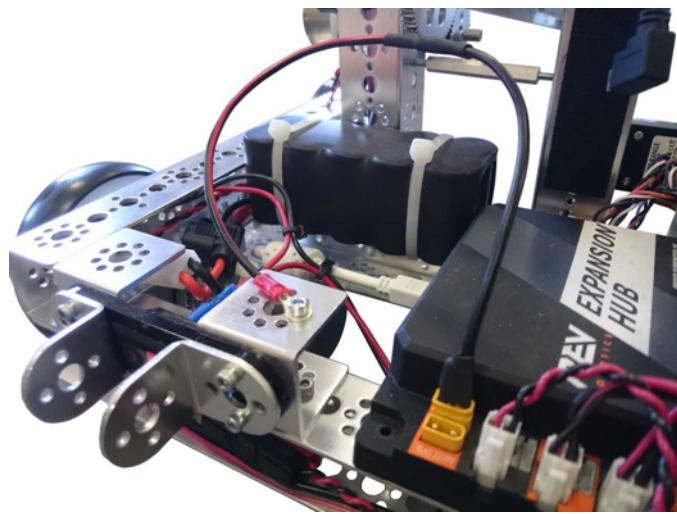


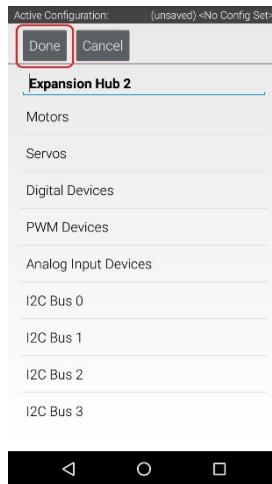
Fig. 11: Ground the electronics to the frame using a FIRST-approved cable.

## Saving the Configuration Information

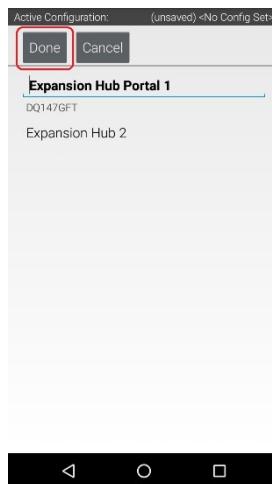
Once you have configured your hardware, you must save the information to the configuration file. If you do not save this information, it will be lost and the robot controller will be unable to communicate with your hardware.

### Saving the Configuration Information Instructions

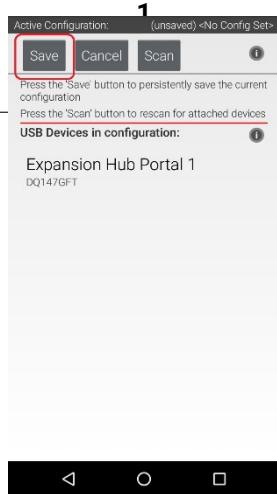
1. Press the **Done** button to go up one level in the configuration screens.



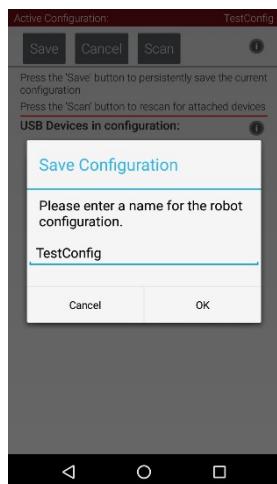
2. Press the **Done** button again to return to the highest level in the configuration screens.



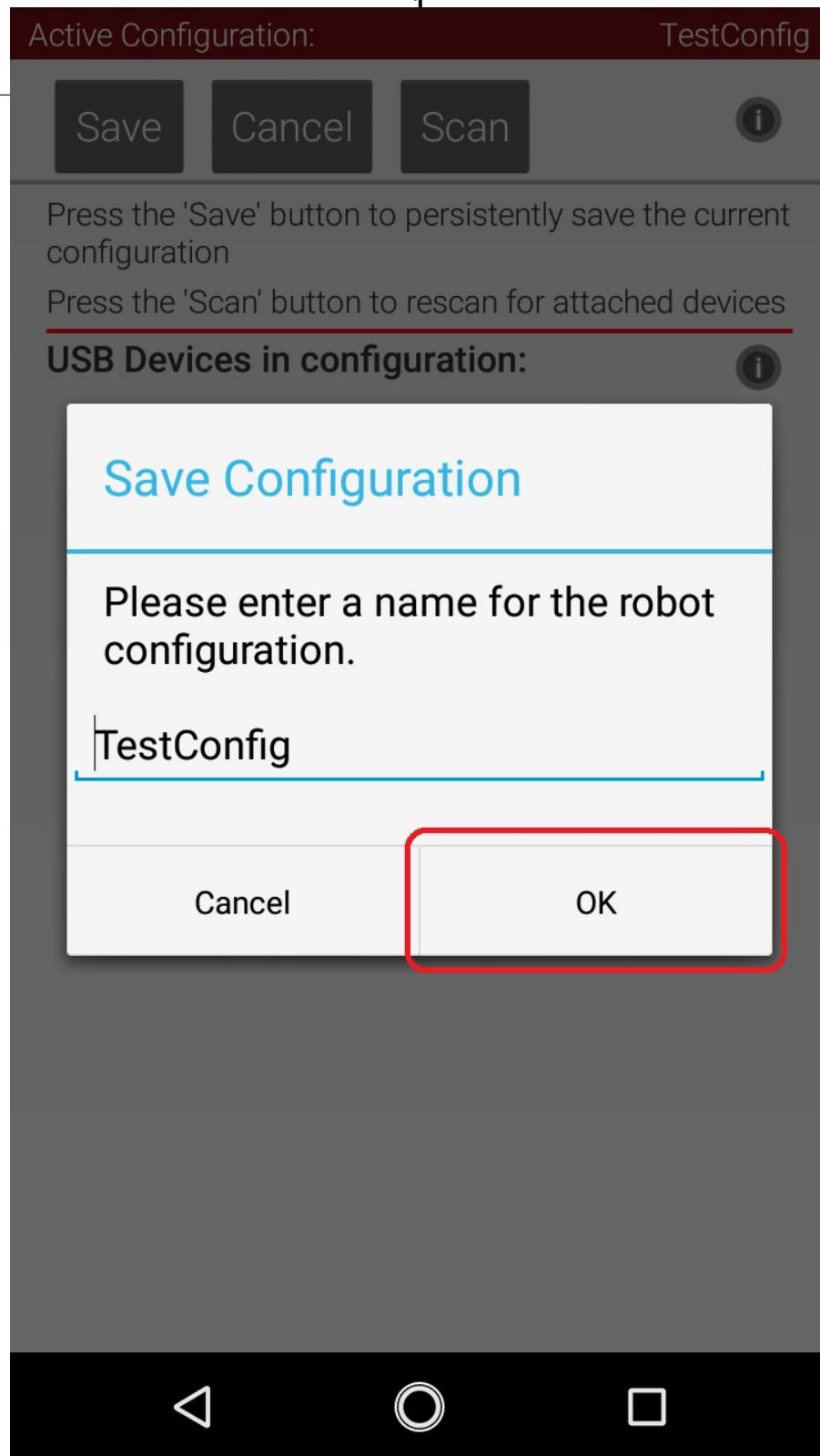
3. Press the **Save** button.



4. When prompted, specify a configuration file name using the touchscreen's keypad (use "TestConfig" for this example).



5. Press the **OK** button to save your configuration information using that file name.



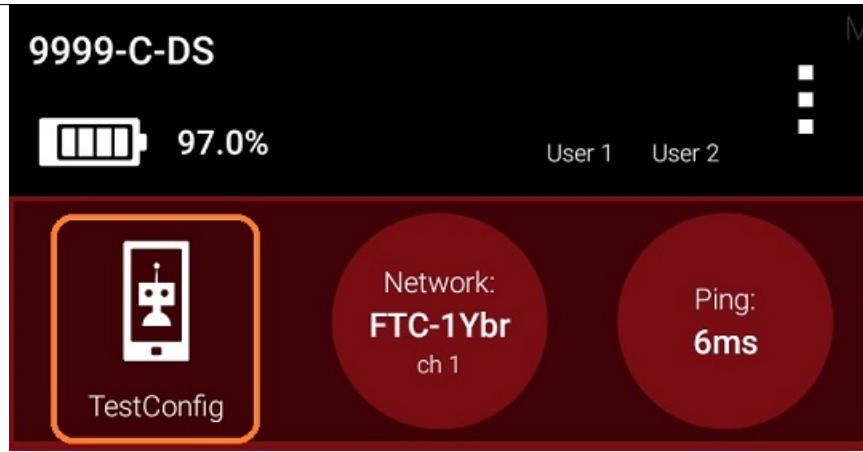
- After the configuration file has been saved, touch the Android back-arrow button to return to the main screen of the app.

**New****Available configurations:**

TestConfig

**Edit****Activate****Delete****Configure from Template**

7. Verify that the configuration file is the active configuration file on the main DRIVER STATION screen.



### 1.2.3 Connecting to the Program & Manage Server Blocks

#### Installing a Javascript Enabled Browser

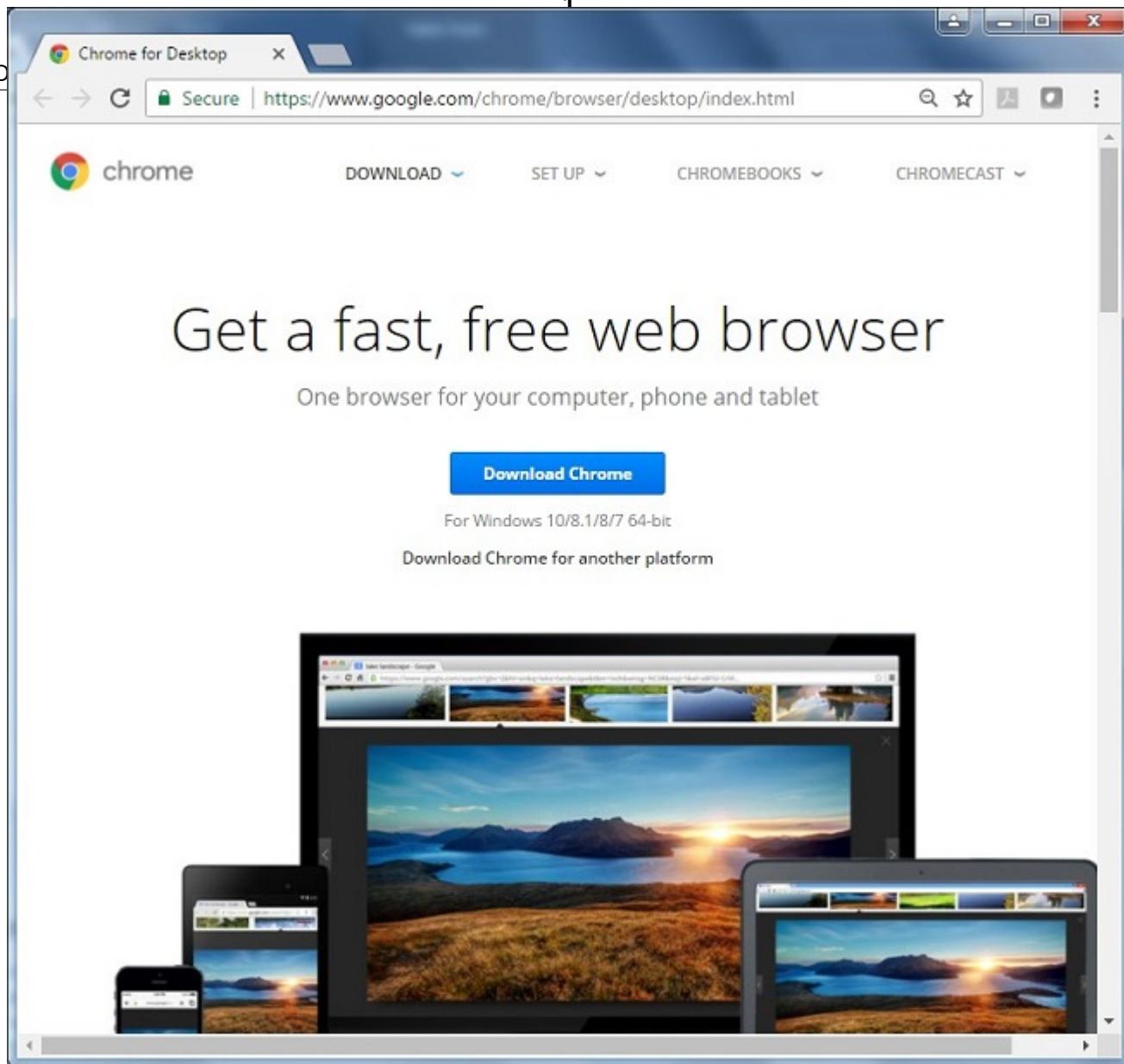
In order to be able to program your Robot Controller using the Blocks Programming Tool or the OnBot Java Programming tool, your laptop will need a Javascript-enabled browser. Both tools are Javascript applications that are served up by the Program and Manage server of the Robot Controller.

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.

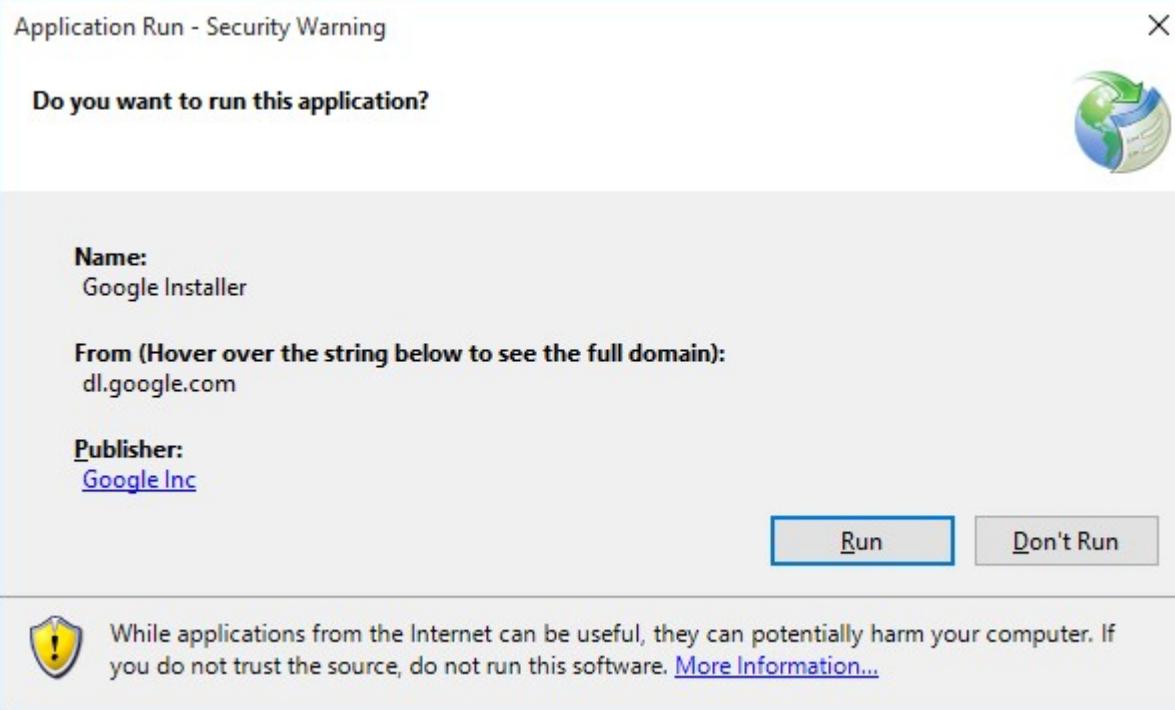
#### Installing a Javascript-Enabled Browser Instructions

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.

## 1 Connecting Your Laptop to the Program & Manage Network Instructions

FTC Docs

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.



The screenshot shows the FIRST robot controller console application interface. At the top, it displays the FIRST logo and the text "robot controller console". On the right side, there is a menu icon consisting of three horizontal lines. The main content area is divided into two sections:

- Robot Controller Connection Info**:
  - The connected robot controller, 9999-C-RC, resides on the wireless network named: **DIRECT-XK-9999-C-RC**
  - The passphrase for this network is: **ZU7if0hB**
  - To remotely connect to the controller, connect your laptop's wireless adapter to this network, using the passphrase to gain access. Once connected, enter the following address into your web browser:  
**http://192.168.49.1:8080**
- Robot controller status:**
  - Server OK (Running since Sep 05, 8:23 AM)**
  - Active connections:**

DriverStation #1	connection.html
RobotController #1	connection.html
Windows #1	java/editor.html

At the bottom of the screen, there is a black navigation bar with three white icons: a left arrow, a circle, and a square.

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"



### Robot Controller Connection Info

The connected robot controller, 9999-C-RC, resides on the wireless network named:

DIRECT-XK-9999-C-RC

The passphrase for this network 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".

The screenshot shows the FIRST robot controller console interface. At the top, it says "FIRST® robot controller console". Below that, in a box, it says "Robot Controller Connection Info". It states: "The connected robot controller resides on the wireless network named: FTC-1Ybr". Underneath, it says: "The passphrase for this network is: password". The text "FTC-1Ybr" and "password" are highlighted with red boxes.

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.



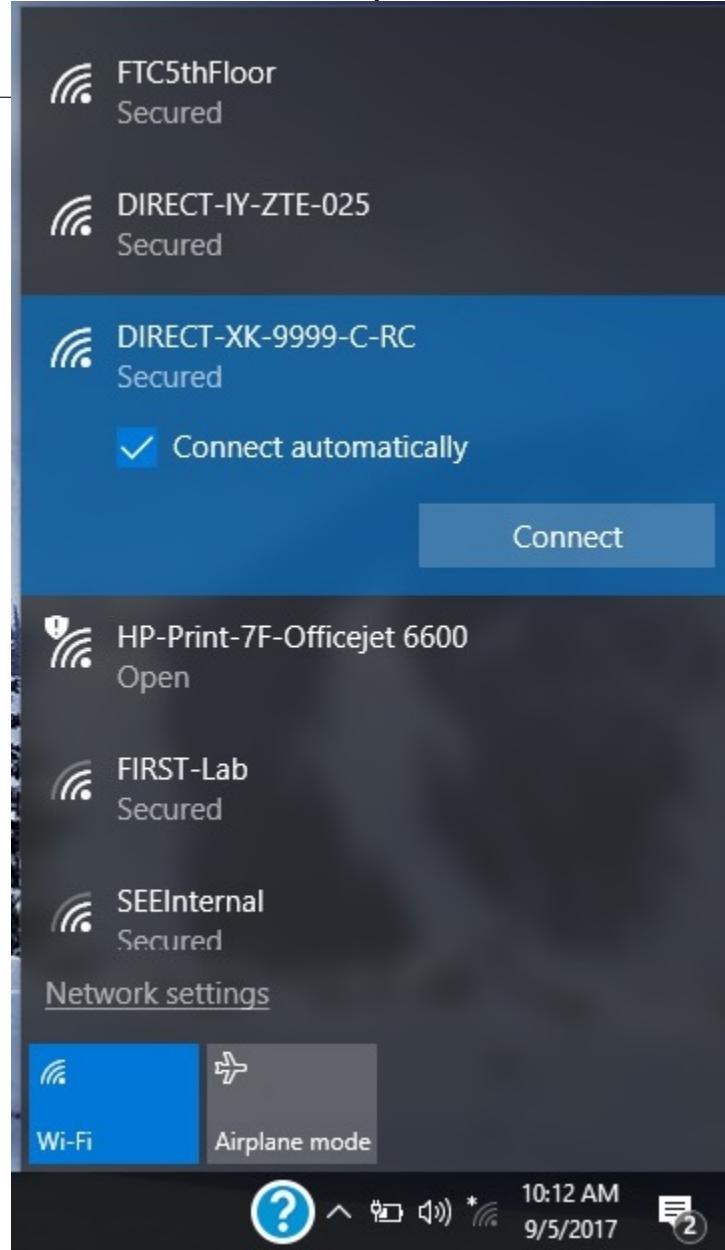
Click on Wi-Fi symbol to display  
Wi-Fi settings for your  
Windows 10 computer.

5. Look for the wireless network that matches the name displayed on the Program & Manage screen.



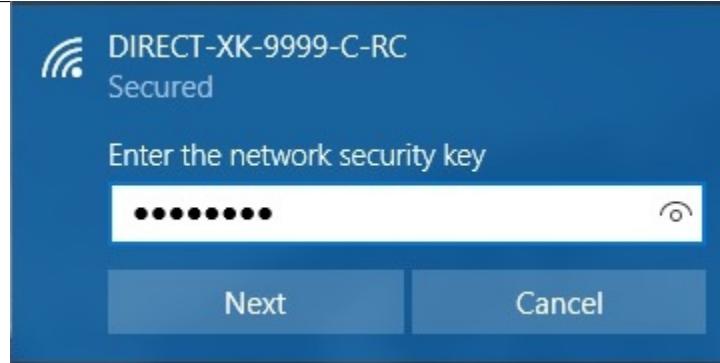
In this example, the name of the wireless network for the Android Robot Controller is “DIRECT-XK-9999-C-RC” and the network is visible in the list displayed on the Windows 10 computer.

- Once you have found the target network in the list, click on it to select it.



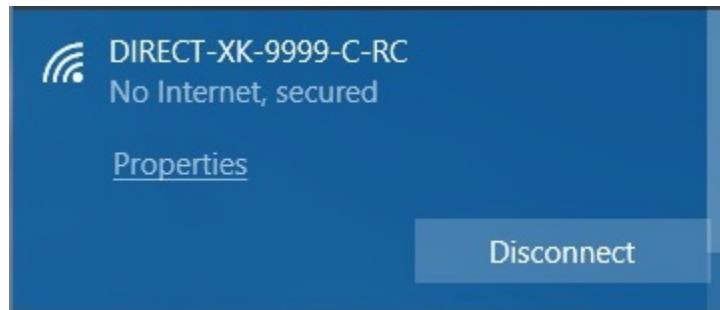
Press the Connect button to connect to the network.

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.

- 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:

- Is the Robot Controller running and connected to the DRIVER STATION?
- 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.

## 1.2.4 Writing an Op Mode Blocks

### Creating Op Modes Blocks

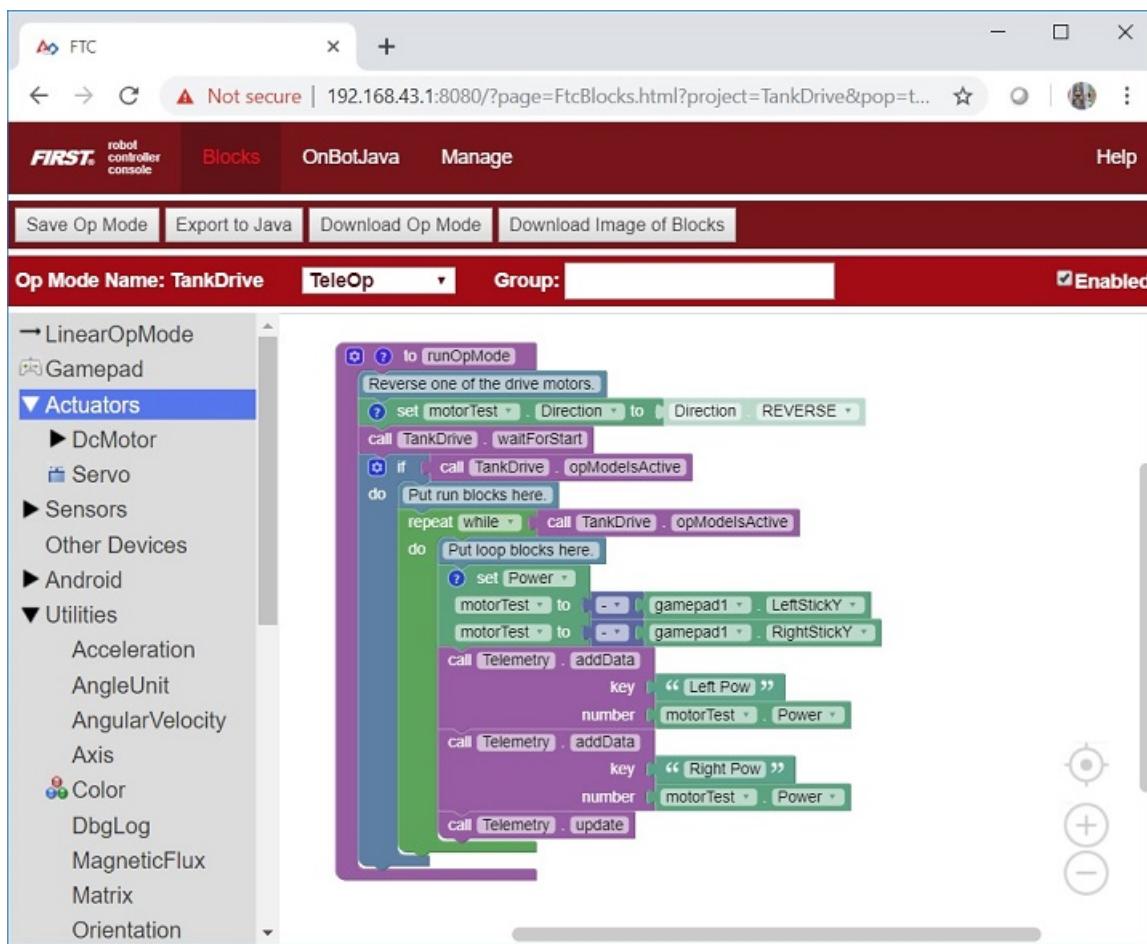
#### What's an Op Mode?

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.



Laptop



WiFi Connection



Robot Controller

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.



Laptop



WiFi Connection



Control Hub

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 Program & 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").

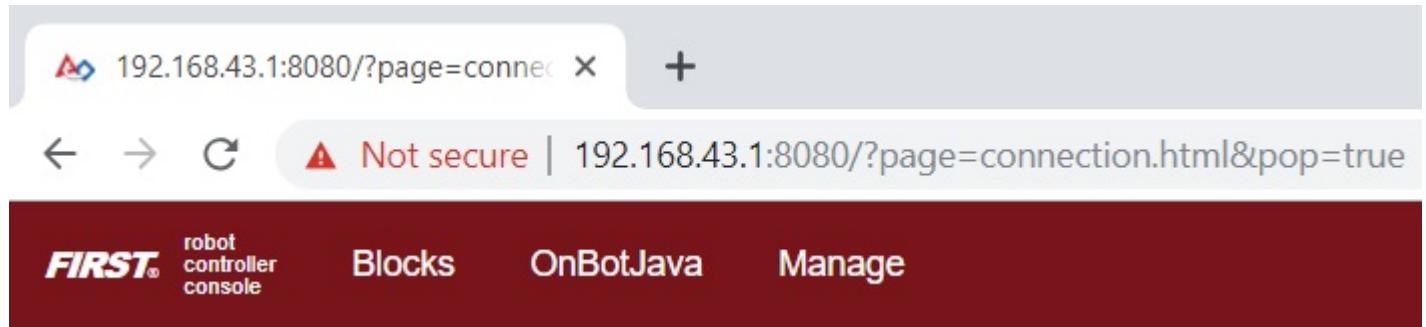
To *remotely* connect to the controller, connect your laptop's wireless adapter to this network, using the passphrase to gain access. Once connected, enter the following address into your web browser:

**http://192.168.43.1:8080**

Robot controller status:

**Server OK (Running since Dec 31, 7:00 PM)**

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.

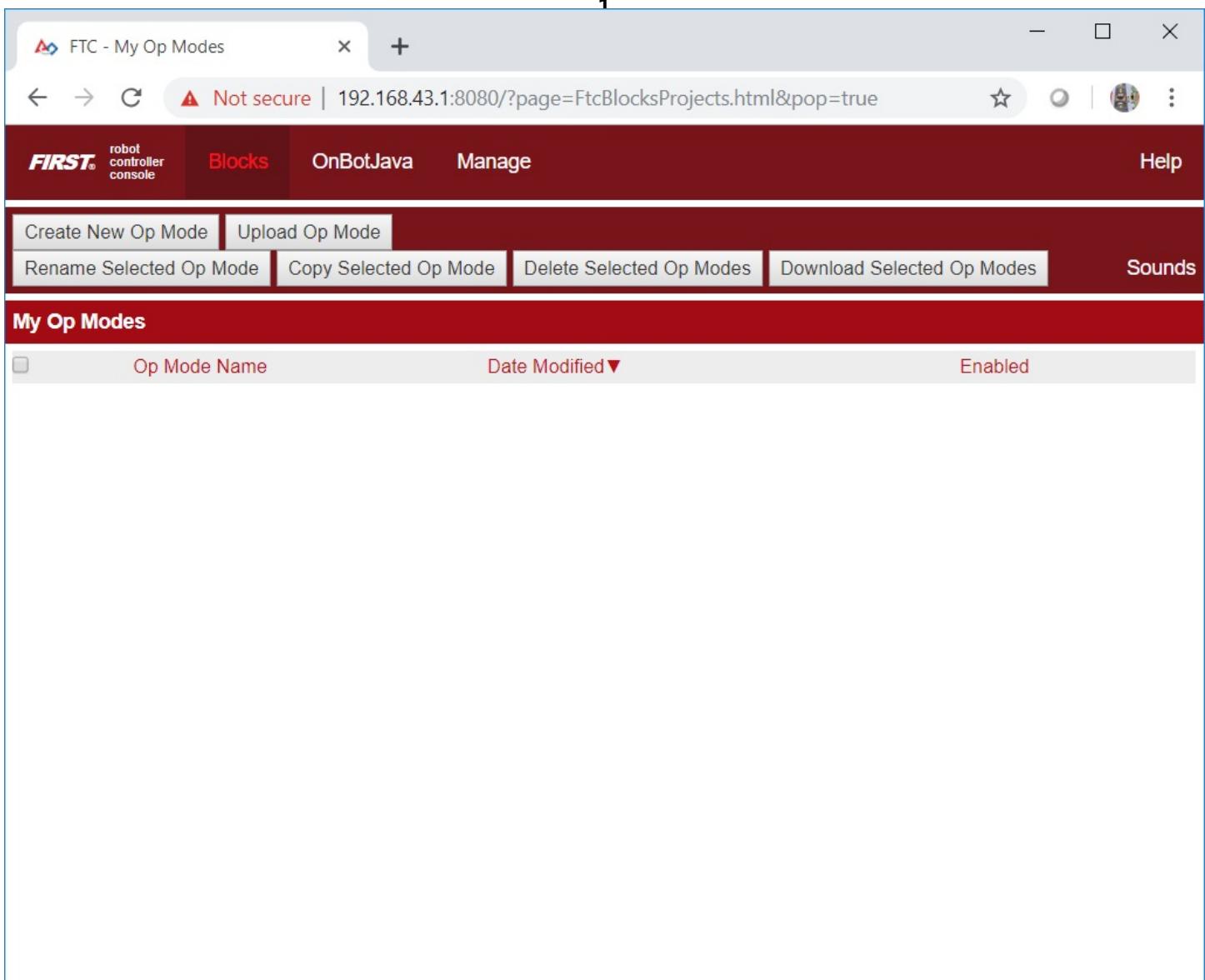
The screenshot shows a web browser window with the URL `192.168.43.1:8080/?page=connection.html&pop=true`. The page title is "Not secure". The top navigation bar includes links for FIRST, robot controller console, Blocks, OnBotJava, Manage, and Help. The main content area is titled "Robot Controller Connection Info". It displays the following information:

- The connected robot controller resides on the wireless network named: **FTC-1Ybr**
- The passphrase for this network is: **password**
- Robot controller status: **Server OK (Running since Dec 31, 7:00 PM)**
- Active connections: Windows #1 connection.html

3. Press the **Blocks** link towards the top of the Console to navigate to the main Blocks Programming screen.

The screenshot shows a web browser window with the URL `192.168.43.1:8080/?page=connection.html&pop=true`. The page title is "Robot Controller Connection Info". The browser status bar indicates "Not secure". The navigation bar includes back, forward, and refresh buttons. The main menu at the top has four items: "FIRST® robot controller console" (disabled), "Blocks" (highlighted with an orange border), "OnBotJava", and "Manage". Below the menu, the page content starts with the heading "Robot Controller Connection Info". It states that the connected robot controller resides on the wireless network named "FTC-1Ybr".

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.

## Create New Op Mode

Op Mode Name:

Sample:  ▾

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.

Op Mode Name: MyFIRSTOpMode    TeleOp    Group:   Enabled

This function is executed when this Op Mode is selected from the Driver Station.

```

    to runOpMode
      Put initialization blocks here.
      call MyFIRSTOpMode . waitForStart
      if call MyFIRSTOpMode . opModelsActive
        do Put run blocks here.
          repeat while call MyFIRSTOpMode . opModelsActive
            do Put loop blocks here.
              call Telemetry . update
  
```

Notice that the left-hand side of the browser 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.

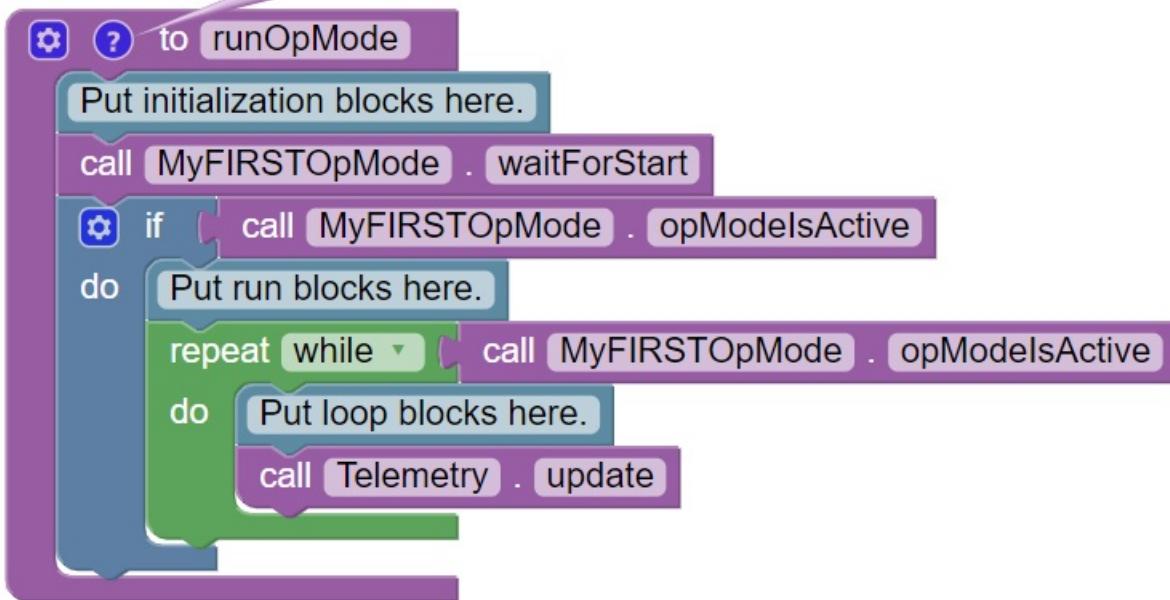
## Examining the Structure of Your Op Mode

FTC Docs

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.

FTC Programming Resources, 154

This function is executed when this Op Mode is selected from the Driver Station.



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.



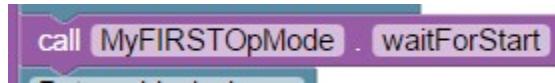
If you think about an op mode as a list of instructions for the robot, this set of instructions will be executed by the robot whenever a team member selects the op mode called “MyFIRSTOpMode” from the list of available op modes for this Robot Controller.

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.

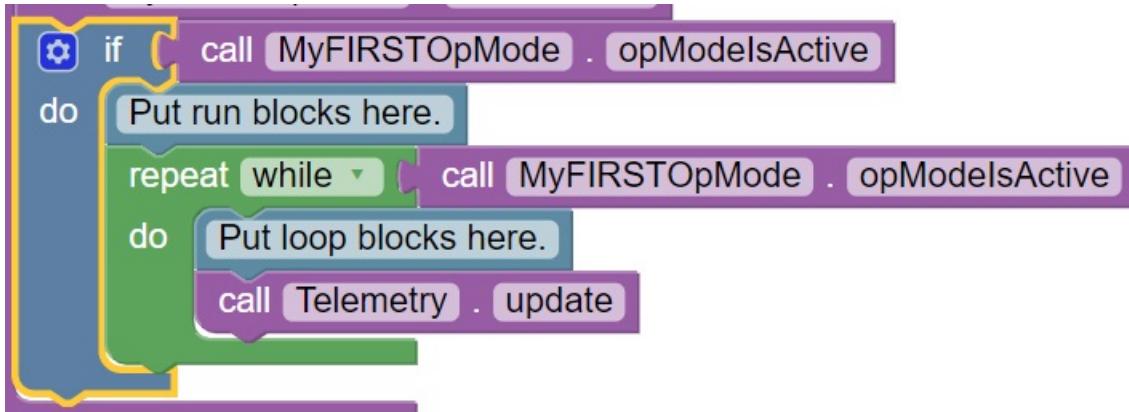
Put initialization blocks here.

Any programming blocks that are placed after the “Put initialization blocks here” comment (and before the “call MyFIRSTOpMode.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 MyFirstOpMode.opModelsActive” will be executed sequentially by the Robot Controller after the Start button has been pressed.

The green block labeled “repeat while call MyFirstOpMode.opModelsActive” 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.opModelsActive” 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.opModelsActive” 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.

---

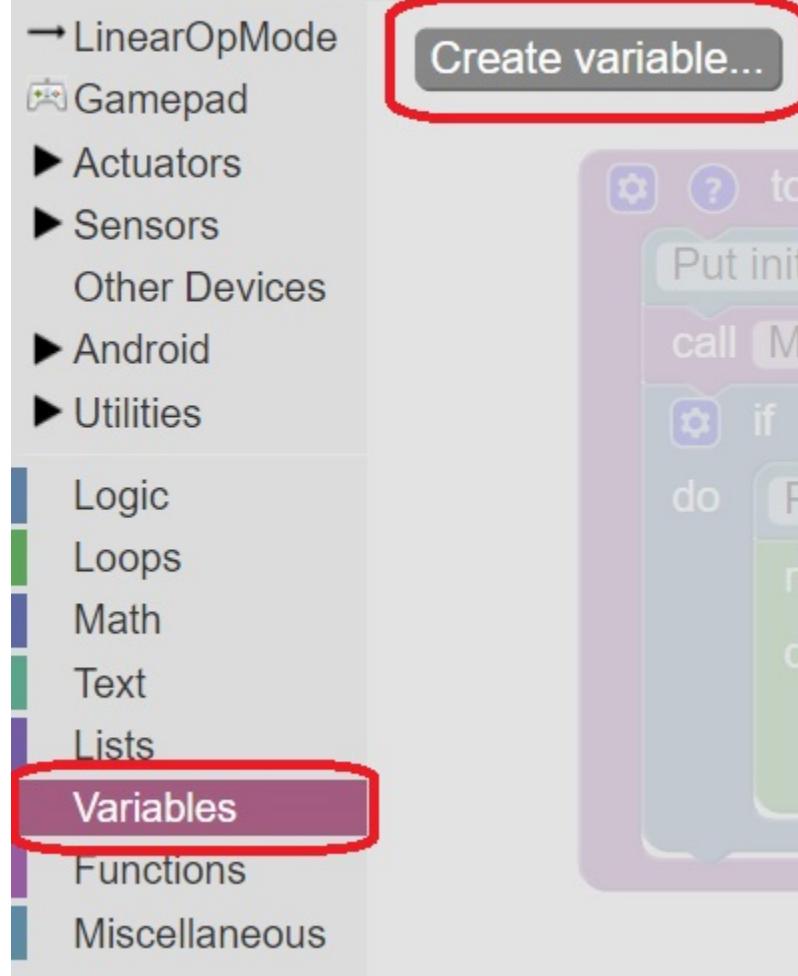
**Important:** The programming blocks for user configured devices (motors, servos and sensors) will only be visible in the Blocks tool if there is an active configuration file with the configured devices included in the file. If a type of device is not included in the active configuration file, then 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.

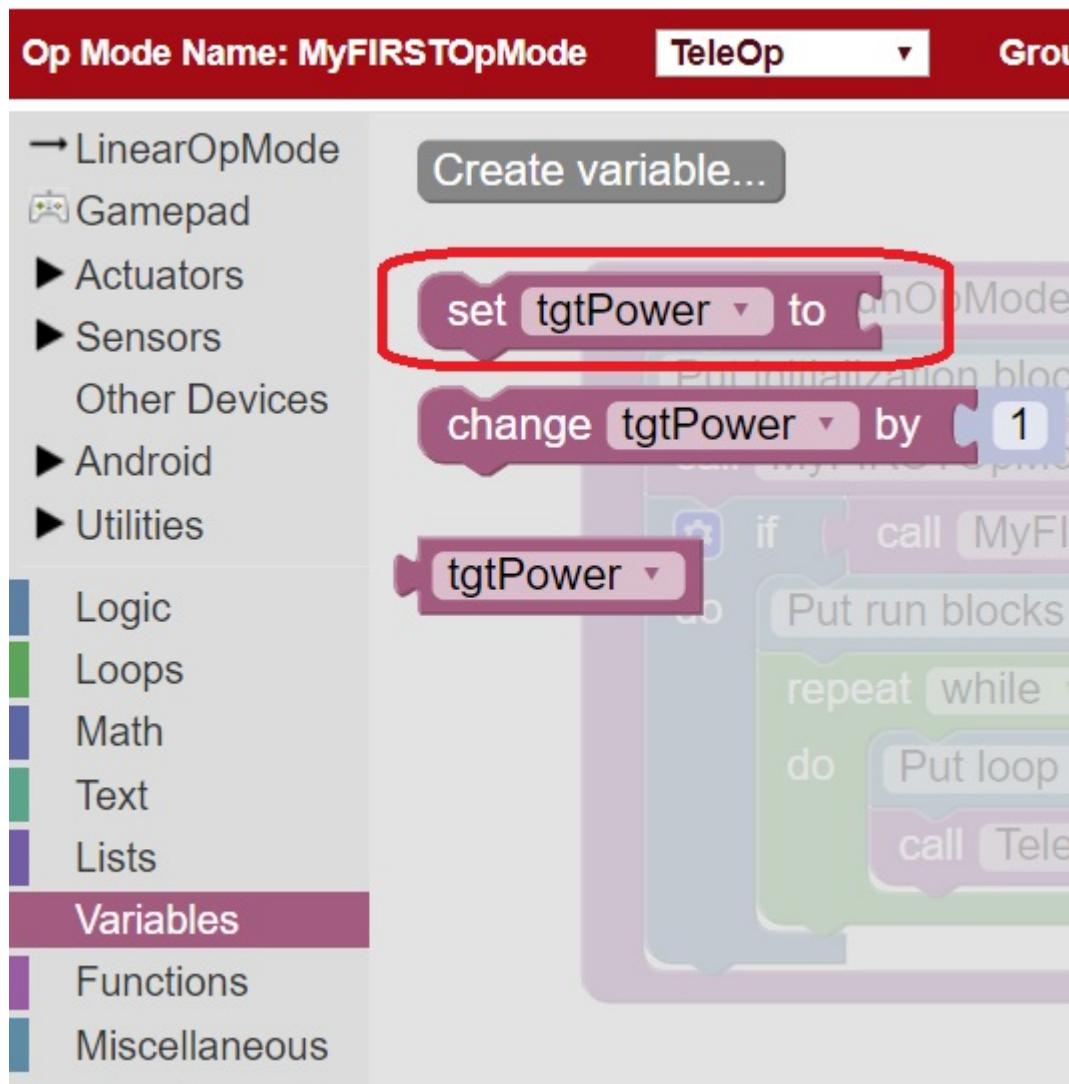
New variable name:

tgtPower

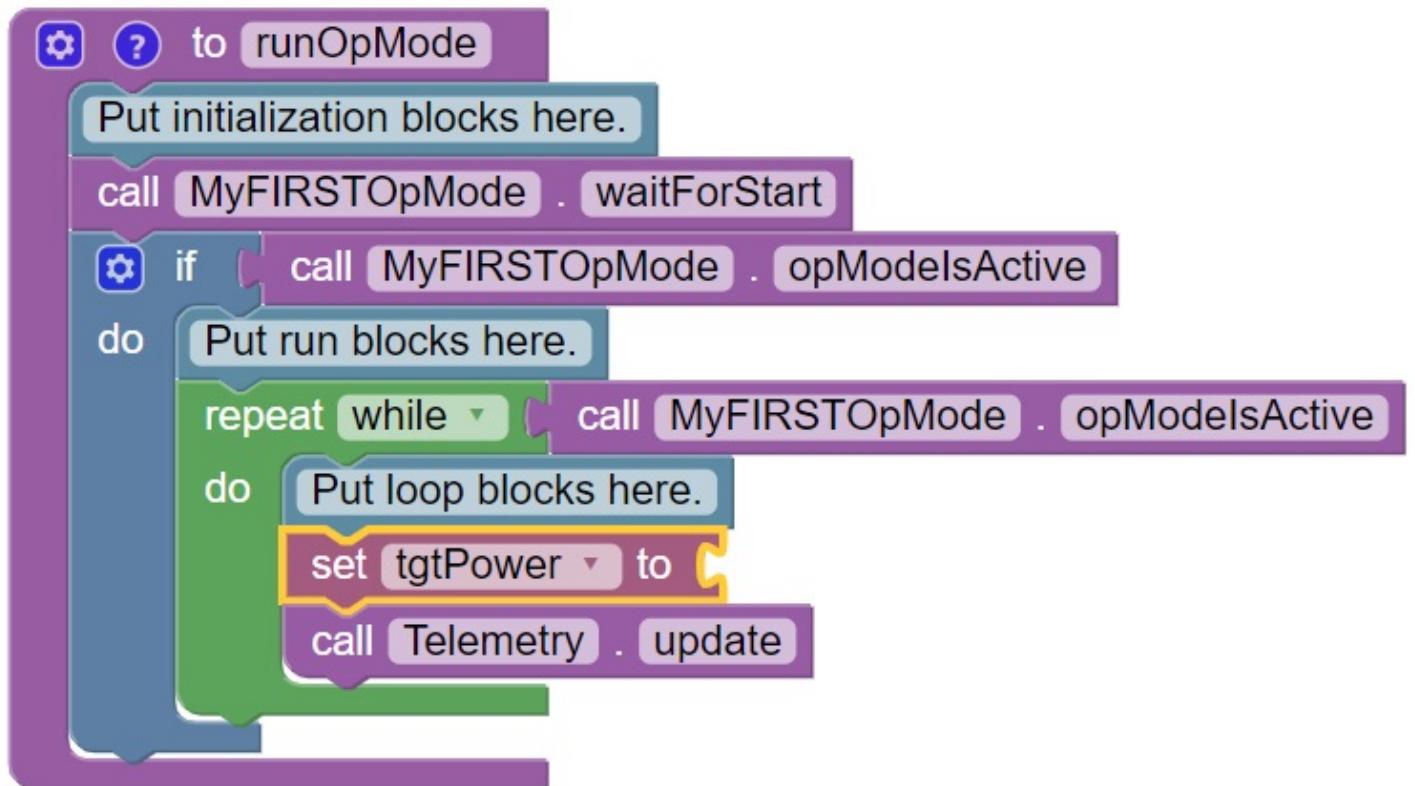
OK

Cancel

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.

→ LinearOpMode

**Gamepad**

► Actuators

► Sensors

Other Devices

► Android

► Utilities

Logic

Loops

Math

Text

Lists

Variables

Functions

Miscellaneous

gamepad1 . DpadDown

gamepad1 . DpadLeft

Put initialization blocks here.

gamepad1 . DpadRight

WaitForStart

Mode . opMode

gamepad1 . DpadUp

MyFIRSTOpMode

gamepad1 . Guide

loop blocks

gamepad1 . LeftBumper

here.

gamepad1 . LeftStickButton

set tgtPower

to

update

gamepad1 . LeftStickX

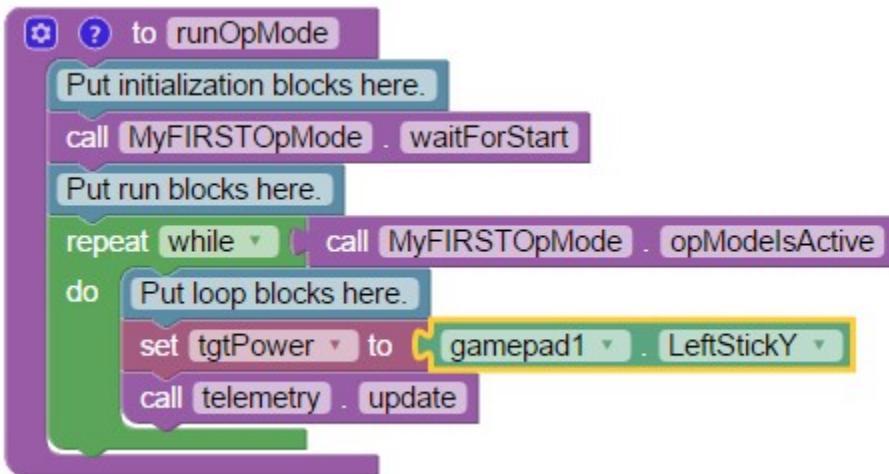
gamepad1 . LeftStickY

Returns a numeric value between -1.0 and +1.0  
representing the left analog stick vertical axis value.

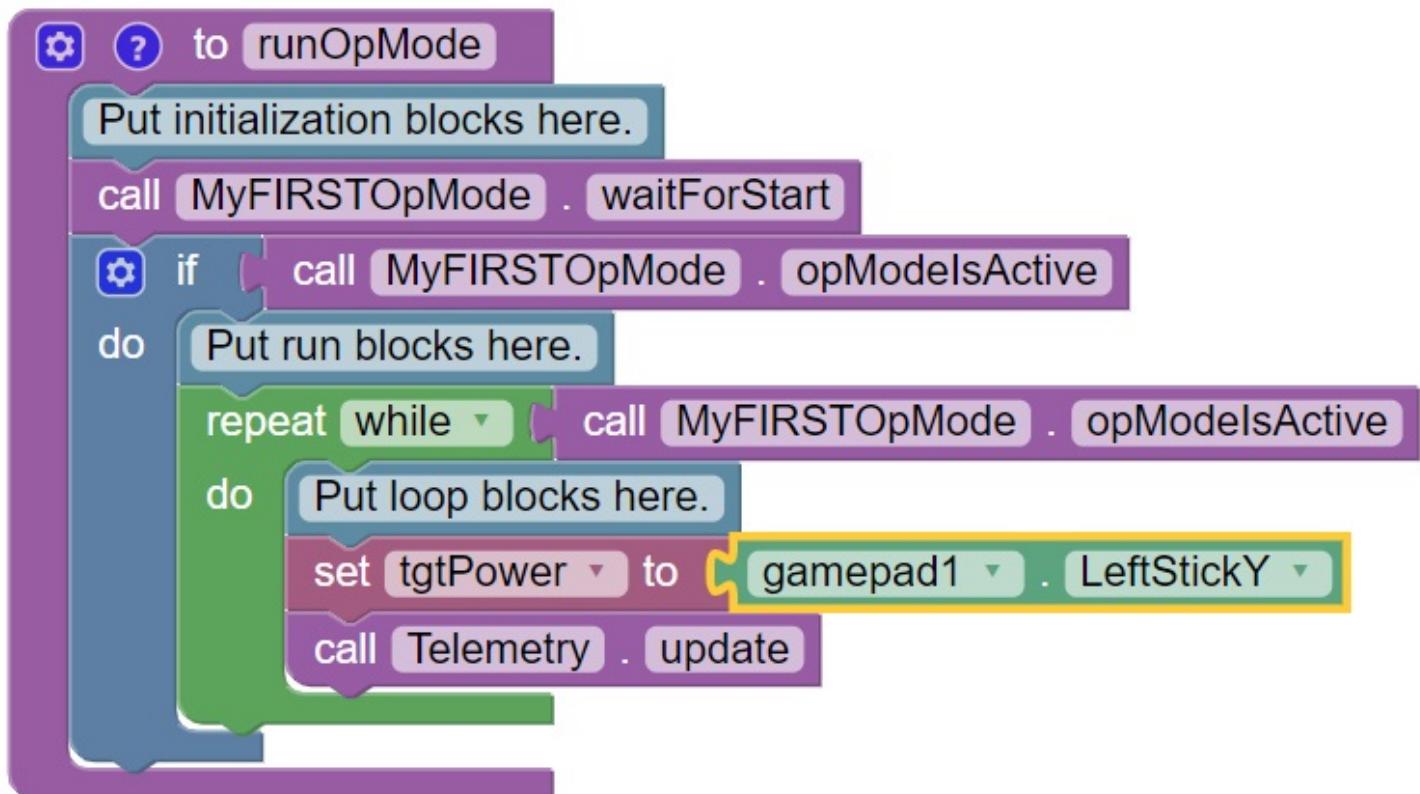
gamepad1 . LeftTrigger

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.

- 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.

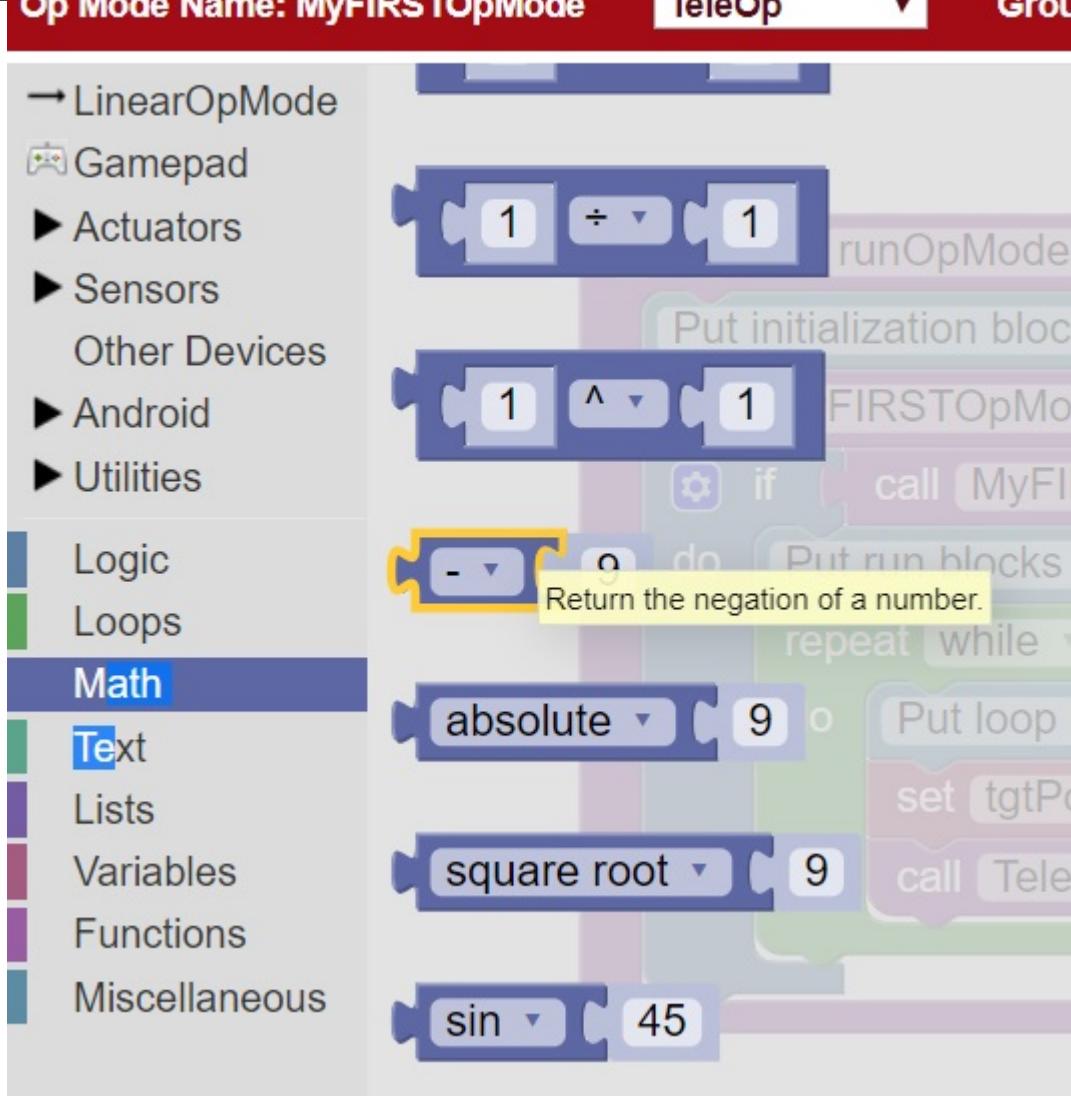


This means that for the blocks shown in our example, if the left joystick is pushed to the top, the variable `tgtPower` will have a value of -1.

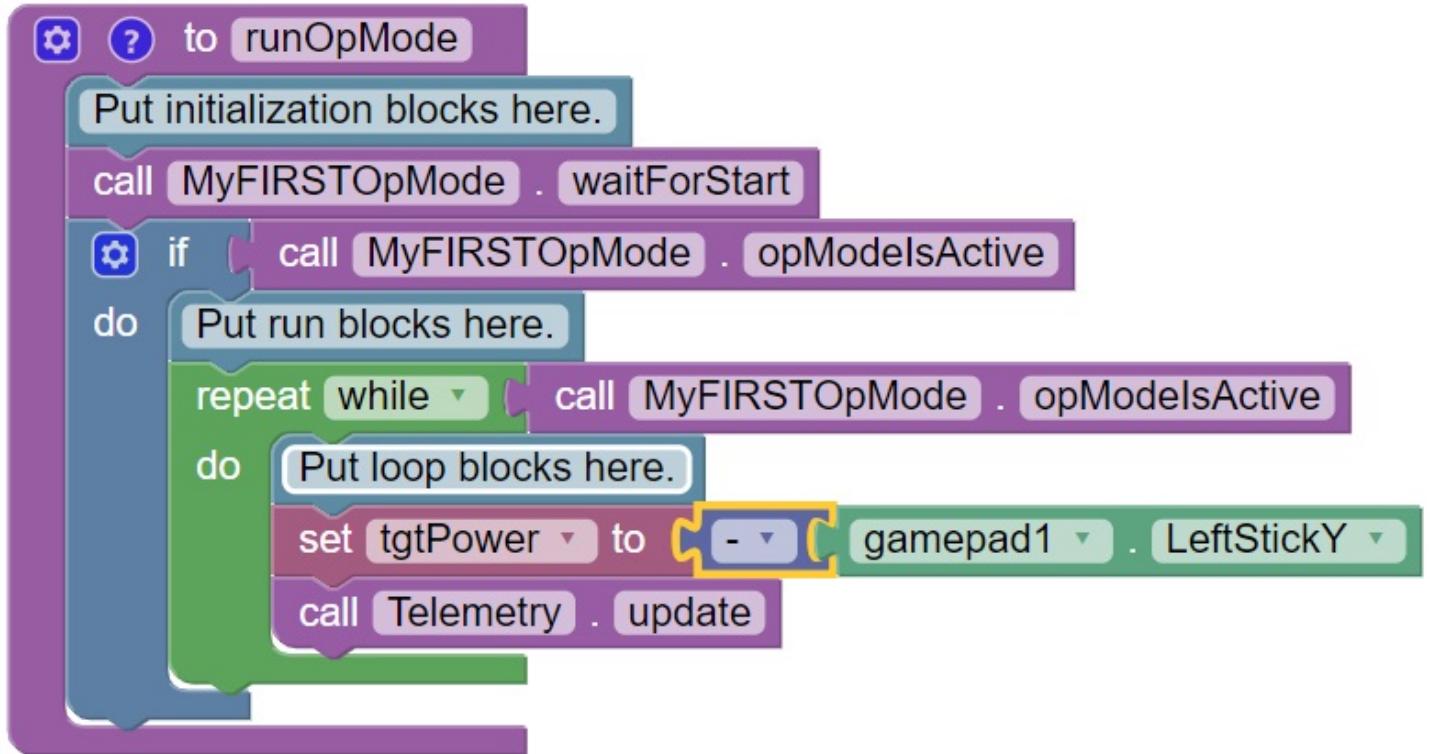
7. Click on the "Math" category for the programming blocks and select the negative symbol (" $-$ ").

FTC Docs

Resources, 163

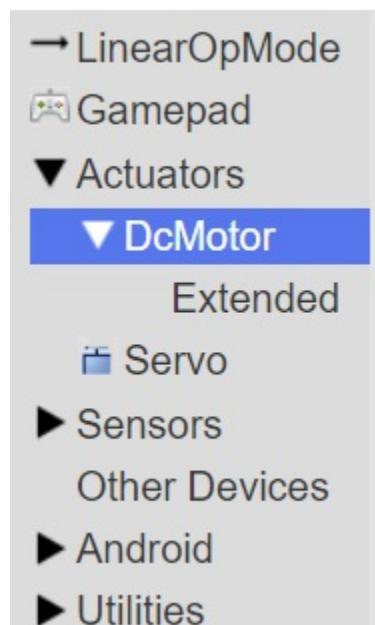


8. Drag the negative symbol (also known as a "negation operator") to the left of the "gamepad1.LeftStickY" block. It should click in place after the "set tgtPower to" block and before the "gamepad1.LeftStickY" block.

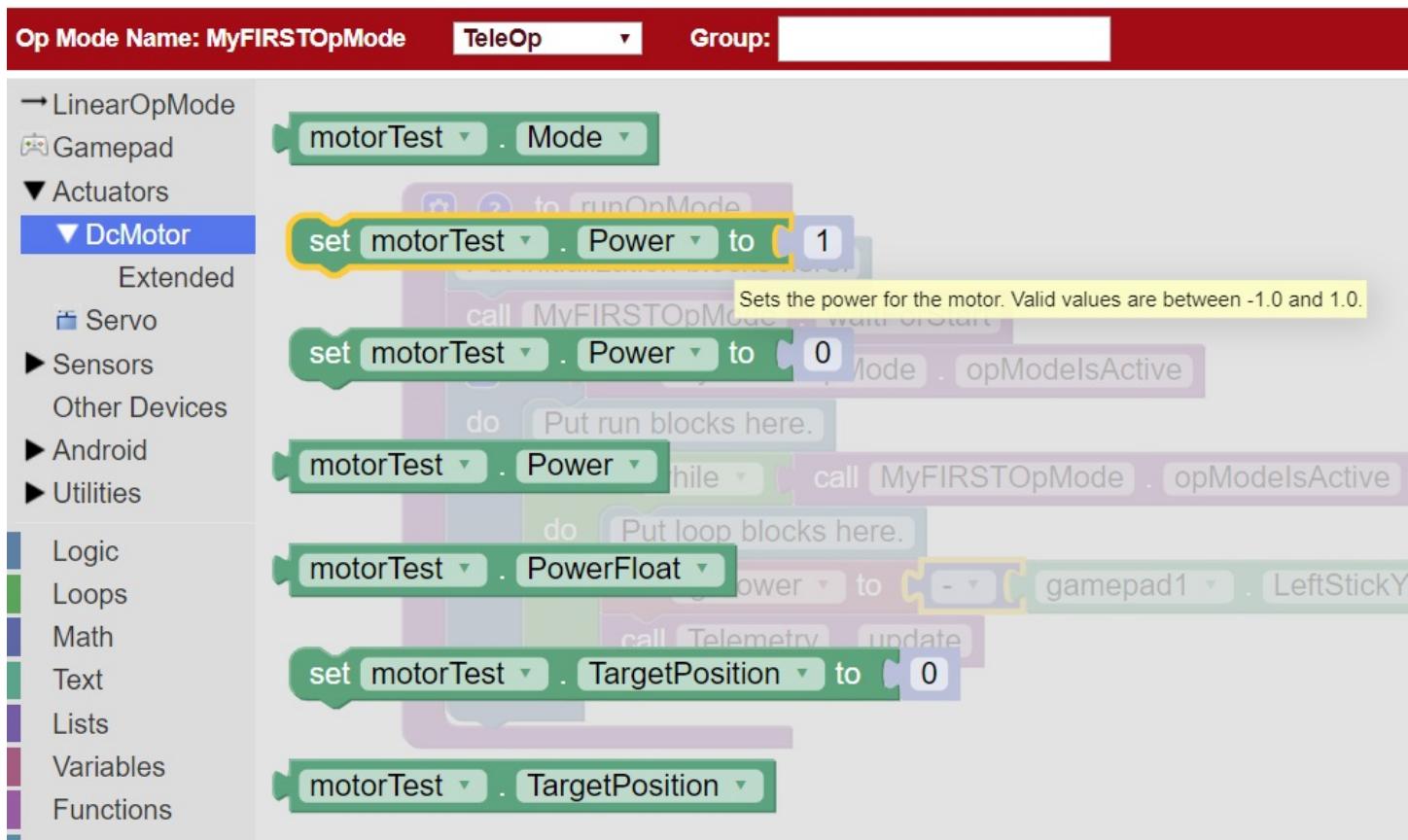


With this change, the variable `tgtPower` will be set to `+1` if the left joystick is in its topmost position and will be set to `-1` if the joystick is in its bottommost position.

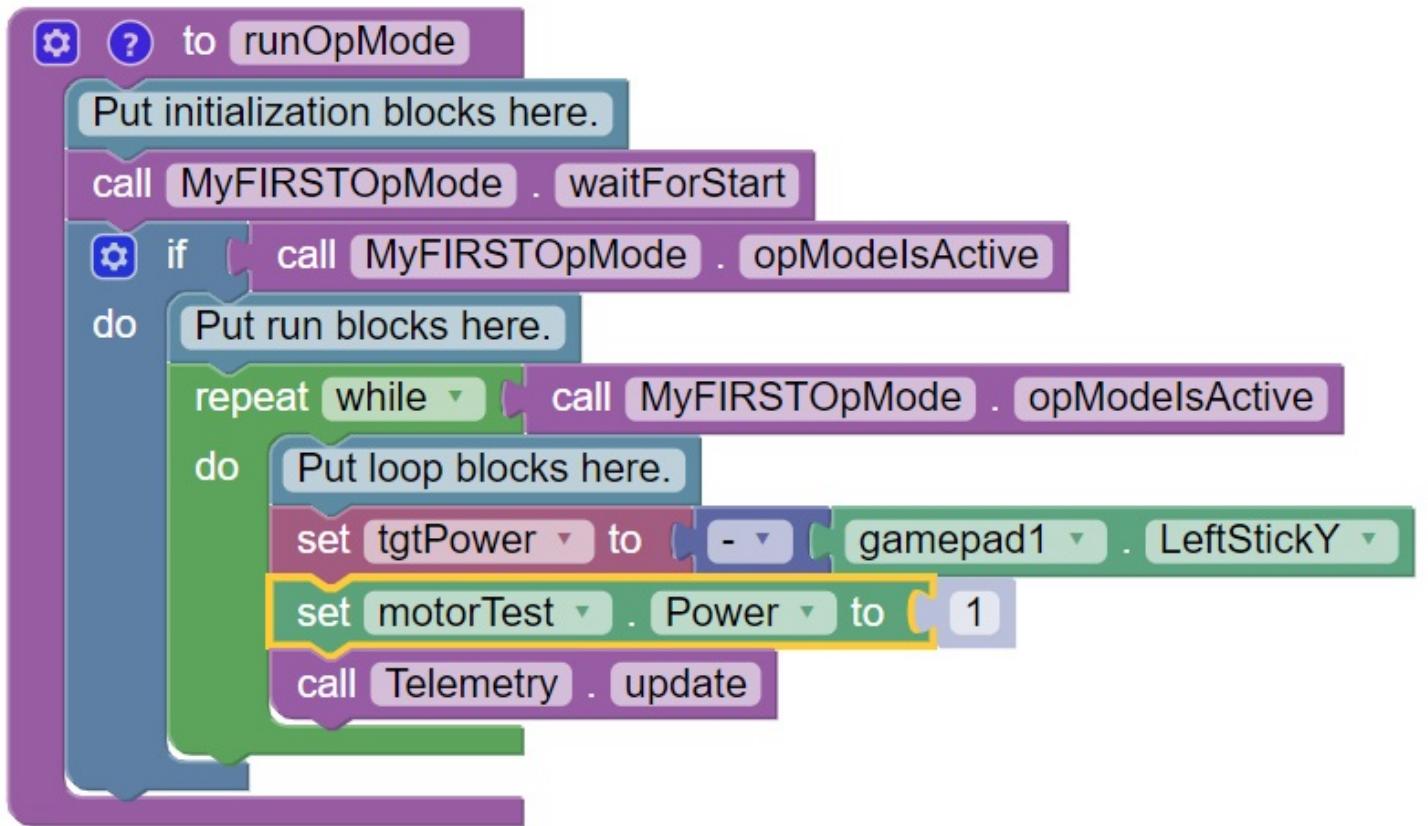
9. Click on the “Actuators” category of blocks. Then click on the “DcMotor” category of blocks.



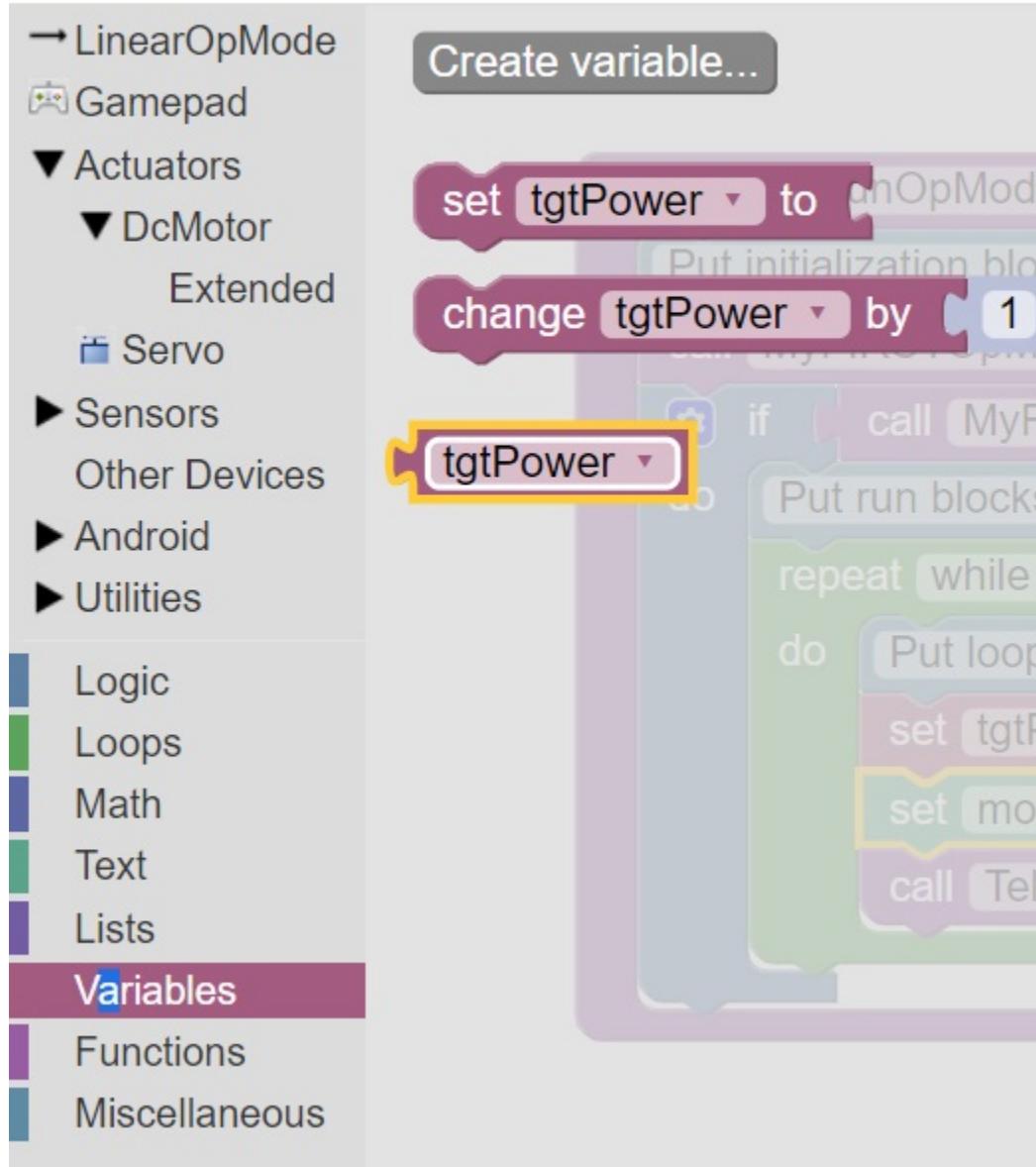
10. Select the “set motorTest.Power to 1” programming block.



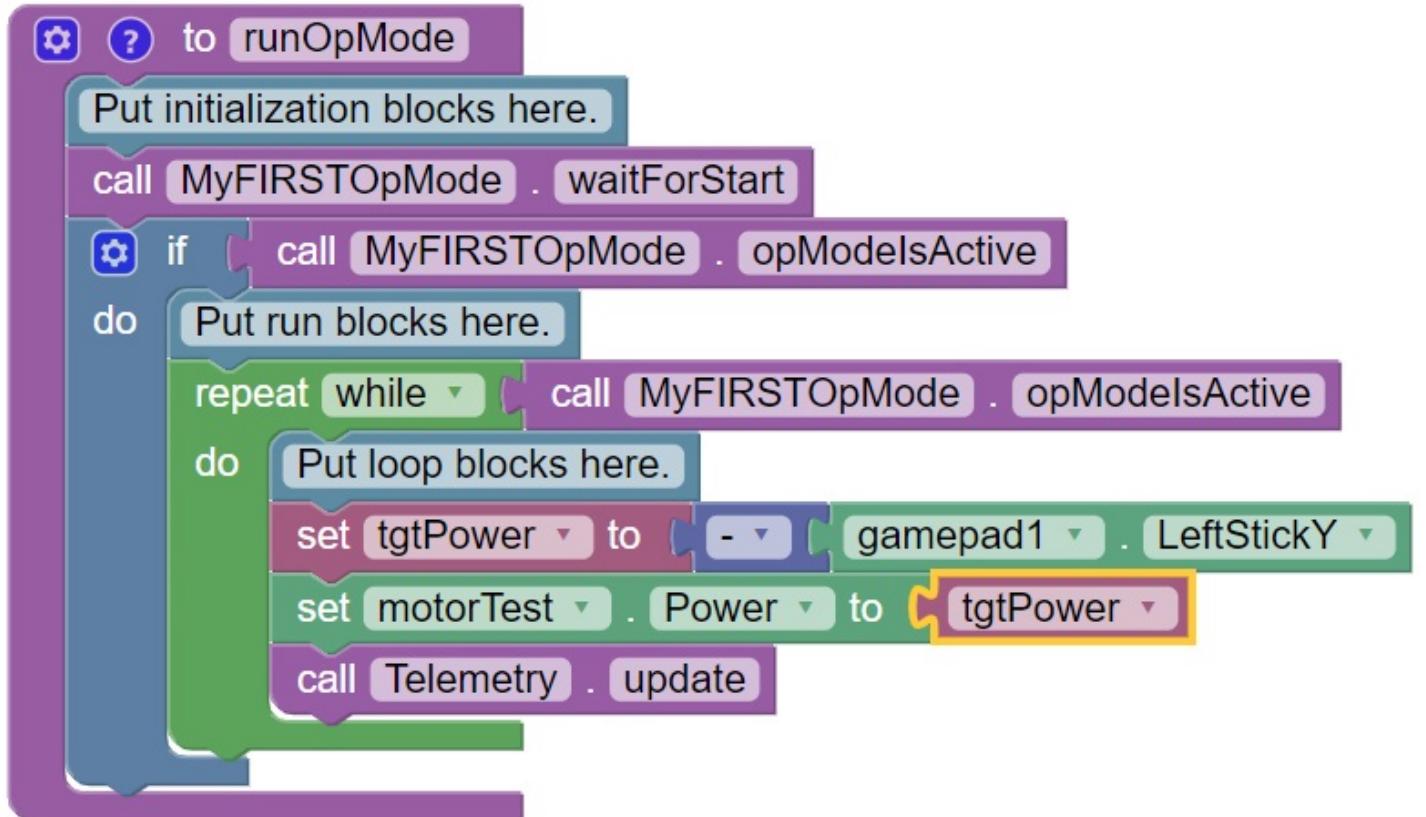
11. Drag and place the “set motorTest.Power to 1” block so that it snaps in place right below the “set tgtPower to” block.



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.

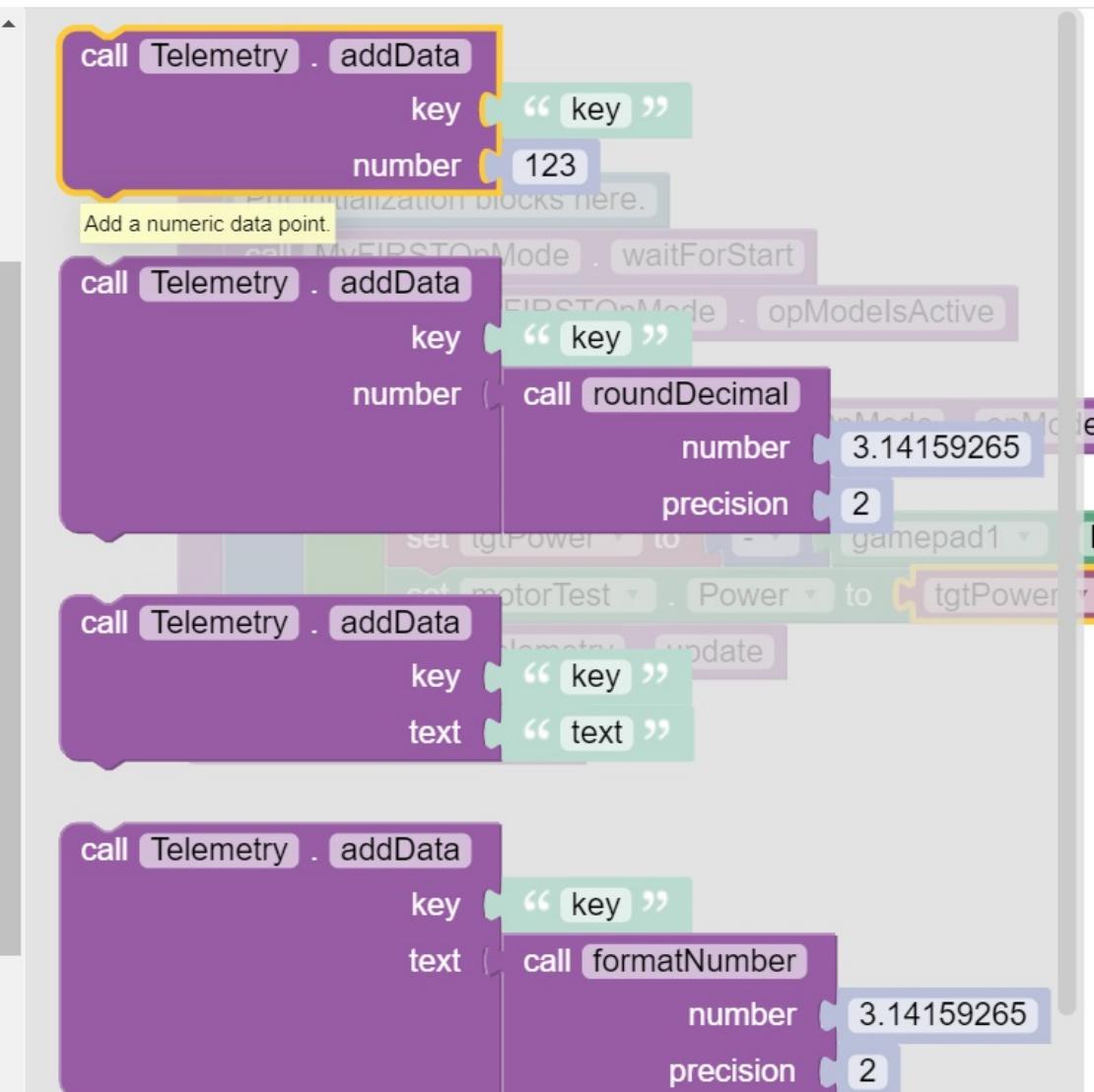
Op Mode Name: MyFIRSTOpMode

TeleOp

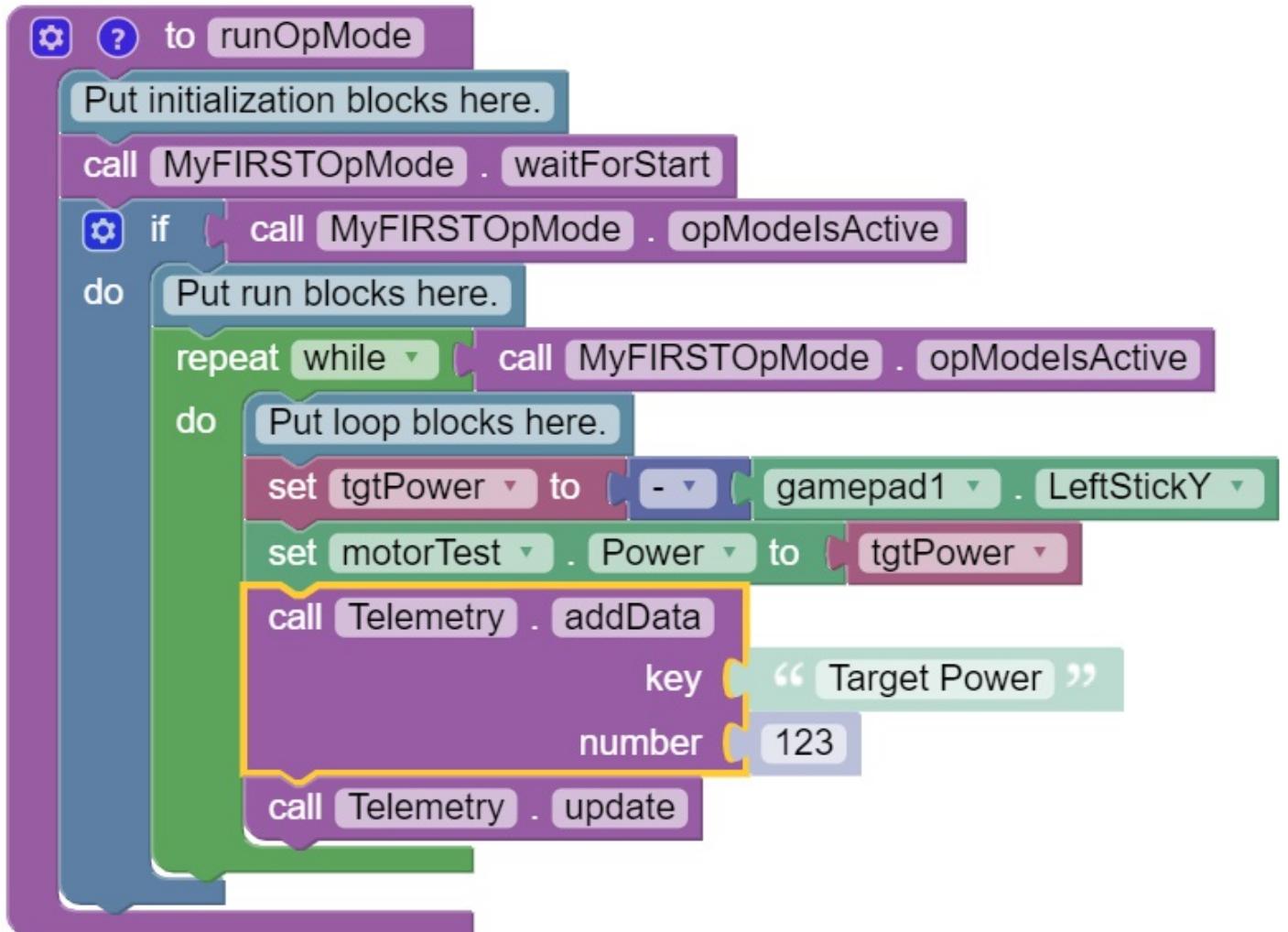
Group:

1

- Android
- ▼ Utilities
  - Acceleration
  - AngleUnit
  - AngularVelocity
  - Axis
  - Color
    - DbgLog
    - MagneticFlux
    - Matrix
    - Orientation
    - PIDFCoefficients
    - Position
    - Quaternion
    - Range
  - Telemetry
  - Temperature
- Time
  - Vector
  - Velocity
- Vuforia
- Logic
- Loops
- Math

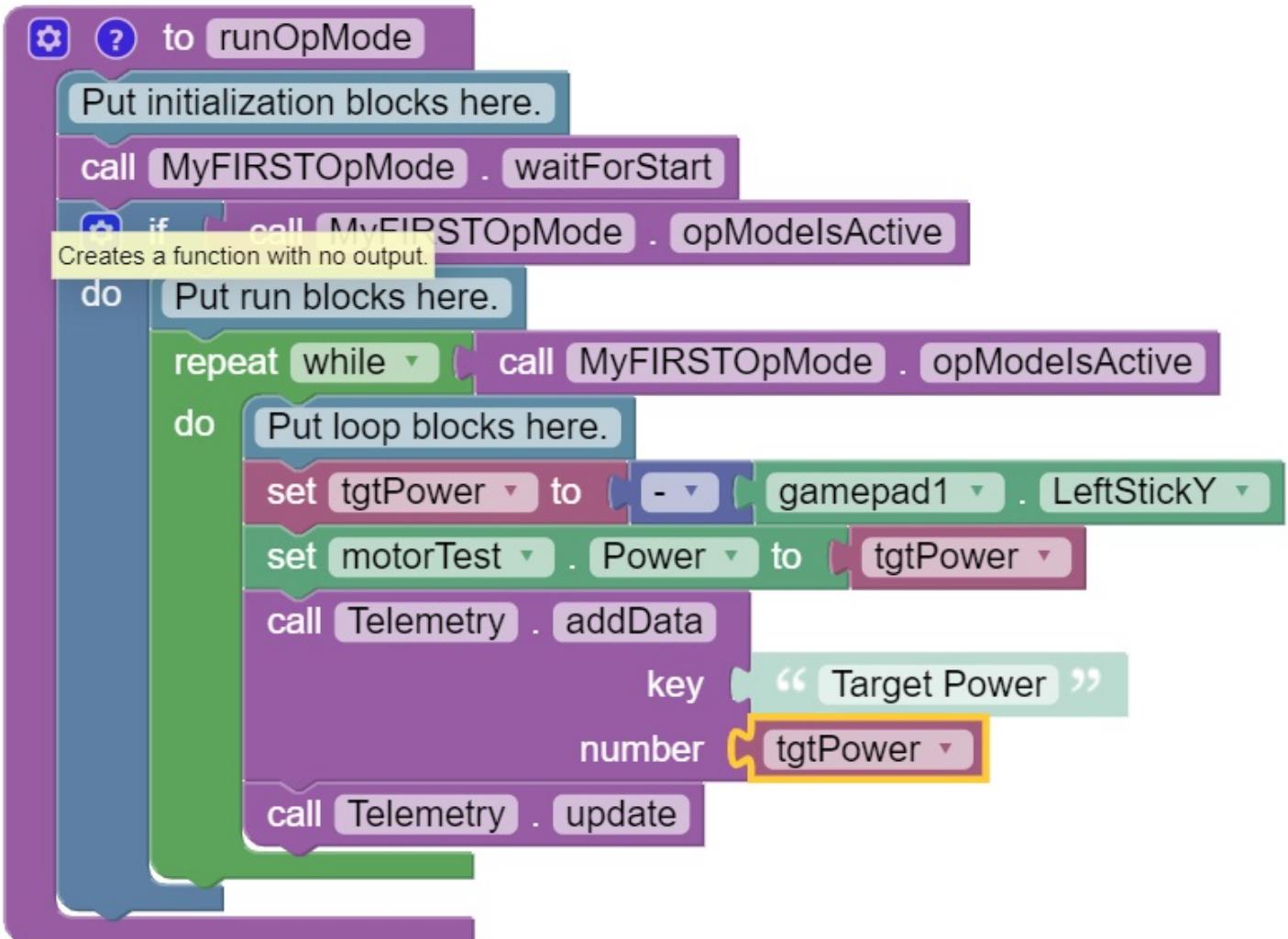


2. Drag the “call telemetry.addData(key, number)” block and place it below the “set motor1.Power to” block. Click on the green text block “key” and highlight the text and change it to read “Target Power”.



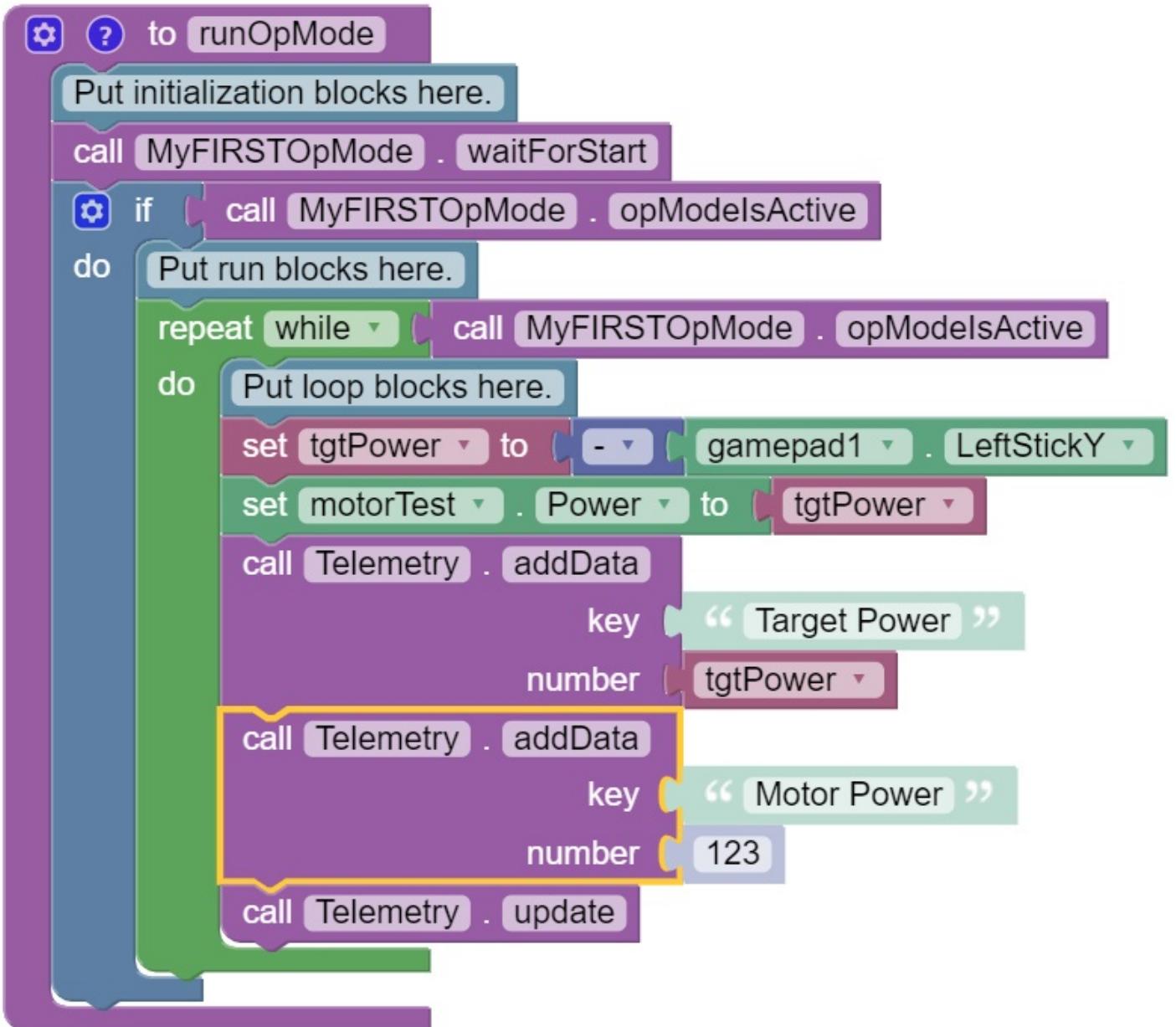
Note that the “call telemetry.update” block is an important block. Data that is added to the telemetry buffer will not be sent to the DRIVER STATION until the “telemetry.update” method is called.

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".



The image shows a Scratch script for the "MyFIRSTOpMode" mode. It starts with a "when green flag clicked" event. Inside, it calls "MyFIRSTOpMode . waitForStart". Then, it enters a "repeat (while)" loop. Inside the loop, it checks if "opModelsActive" is true. If so, it enters a "do" loop. Inside the "do" loop, it sets "tgtPower" to the value of "gamepad1 . LeftStickY" minus a value from a blue variable. It then sets "motorTest . Power" to "tgtPower". It calls "Telemetry . addData" twice: once with key "Target Power" and number "tgtPower", and once with key "Motor Power" and number "123". Finally, it calls "Telemetry . update".

```
when green flag clicked
  [MyFIRSTOpMode . waitForStart]
  [if (opModelsActive) then
    [do
      (repeat (while (opModelsActive)
        [set tgtPower v to (gamepad1 . LeftStickY) - (10)]
        [set motorTest . Power v to tgtPower]
        [call Telemetry . addData "Target Power" (tgtPower)
         "Motor Power" (123)]
        [call Telemetry . update]
      ) until (opModelsActive))
    end]
  end]
```

5. Find and click on the “DcMotor” subcategory. Look for the green programming block labeled “motorTest.Power”.

Op Mode Name: MyFIRSTOpMode

TeleOp

Group:

→ LinearOpMode

Gamepad

Actuators

DcMotor

Servo

Sensors

Other Devices

Android

Utilities

Acceleration

AngleUnit

AngularVelocity

Avg

motorTest . Mode

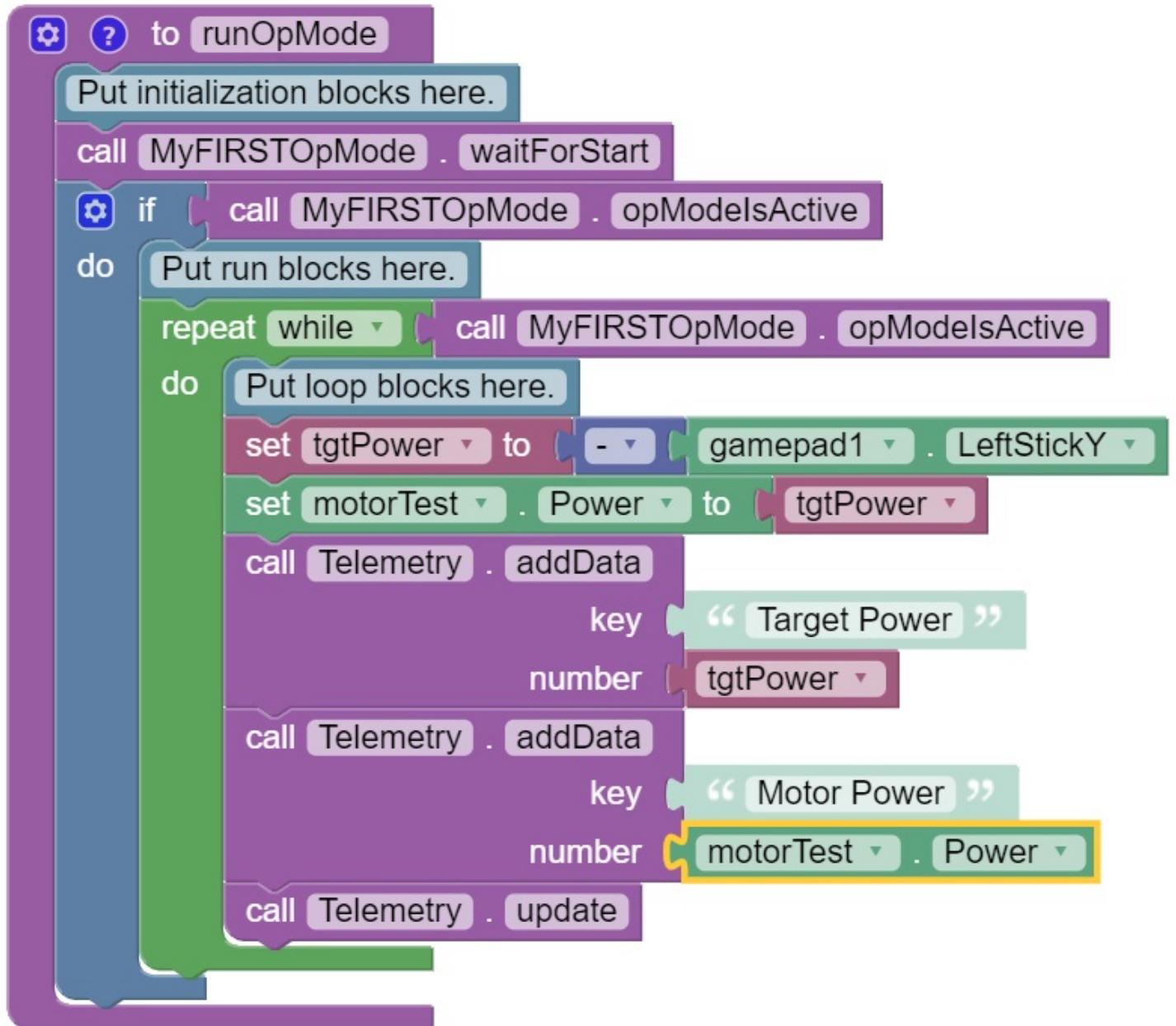
set motorTest . Power to 1

set motorTest . Power to 0

motorTest . Power

motorTest . PowerFloat

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

FTC Docs

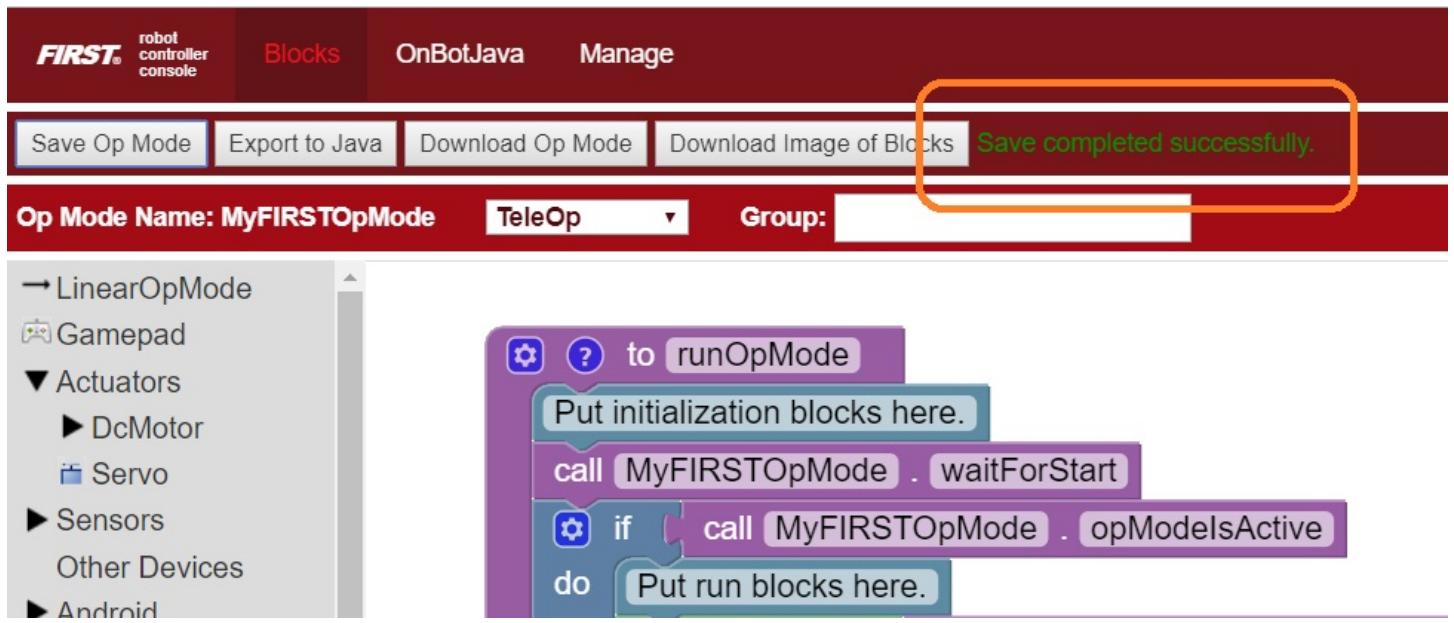
After you have modified your op mode, it is very important to save the op mode to the Robot Controller.

[FTC Programming Resources, 175](#)

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.

The screenshot shows the OnBotJava Blocks interface. At the top, there are tabs for FIRST robot controller console, Blocks, OnBotJava, and Manage. Below the tabs, there are buttons for Save Op Mode, Export to Java, Download Op Mode, and Download Image of Blocks. A green success message 'Save completed successfully.' is displayed in a box. The main area shows the op mode configuration for 'MyFIRSTOpMode' set to 'TeleOp'. The left sidebar lists various device categories: LinearOpMode, Gamepad, Actuators (DcMotor, Servo), Sensors, Other Devices, and Android. The right side shows the block-based code for the 'runOpMode' loop, which includes initialization blocks, a call to 'waitForStart', and a conditional loop for 'opModelsActive'.

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 op mode requires input from a gamepad, then you will need to connect a Logitech F310 gamepad to the DRIVER STATION. Note that you can have up to two gamepads connected through a USB hub to a DRIVER STATION. 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 Op Mode Instructions

~~FTC Docs~~ Before you connect your gamepad to the phone, verify that the switch on the bottom of the gamepad is set to the ~~FTC Programming Resources~~ <sup>1.7.7</sup> X position.



2. Connect the gamepad to the DRIVER STATION using the Micro USB OTG adapter cable.

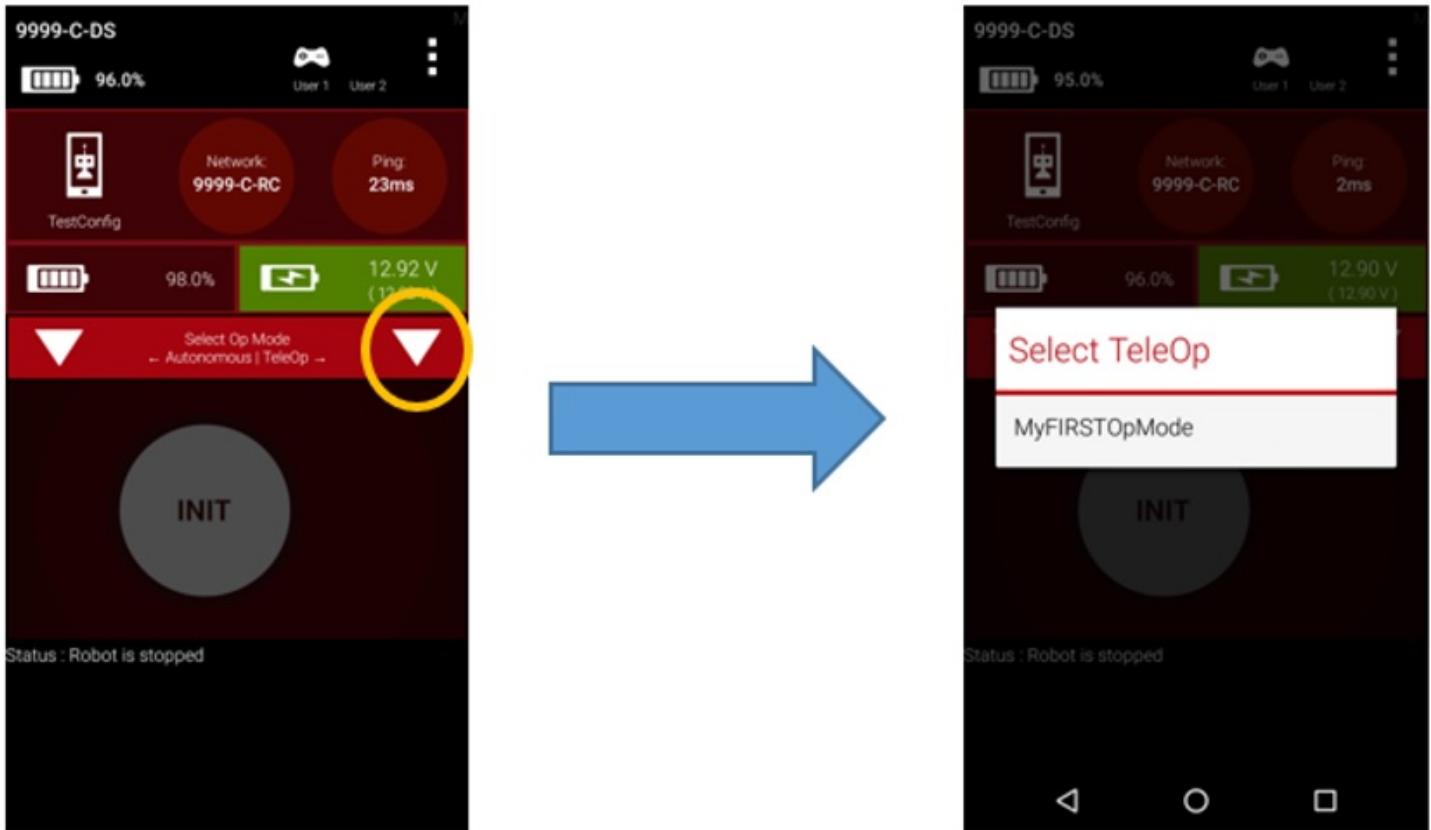


3. For the examples in this wiki, the op modes are looking for input from the gamepad designated as the user or driver #1. Press the Start button and the A button simultaneously on the Logitech F310 controller to designate your gamepad as user #1.



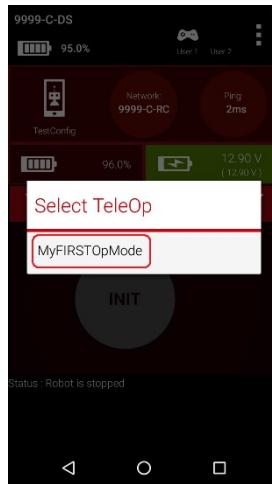
Note that pushing the Start button and the B button simultaneously would designate the gamepad as user #2.

~~4. On the DRIVER STATION screen, touch the triangular-shaped, “TeleOp” dropdown list button to display a list available op modes. You should see your recently saved op mode among the list of available op modes that reside on your Robot Controller.~~



Note that the word “TeleOp” is short for “Tele-Operated” and it implies a driver controlled op mode (i.e., an op mode that gets input from a human driver).

5. Select “MyFIRSTOpMode” to load your op mode on the Robot Controller.



Note that even though you are using the DRIVER STATION to select the op mode, the actual op mode instructions will be executed on the Robot Controller phone.

- Press the INIT button to initialize your op mode.



- Push the Start button (designated by the triangular-shaped symbol) to start the op mode run.



- 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 lower left hand corner of the screen.



If you want to stop your op mode, 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.

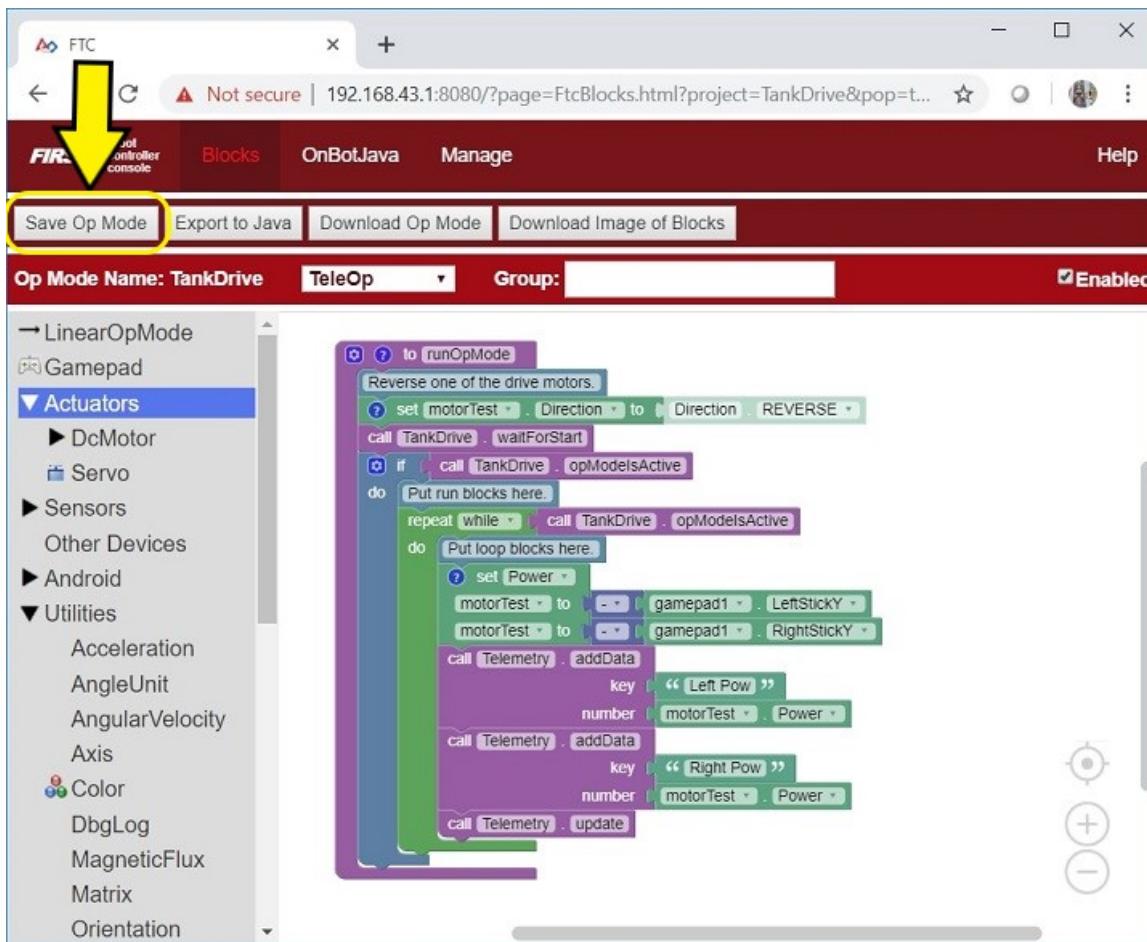


Fig. 12: Saving the OpMode within the Blocks Editor

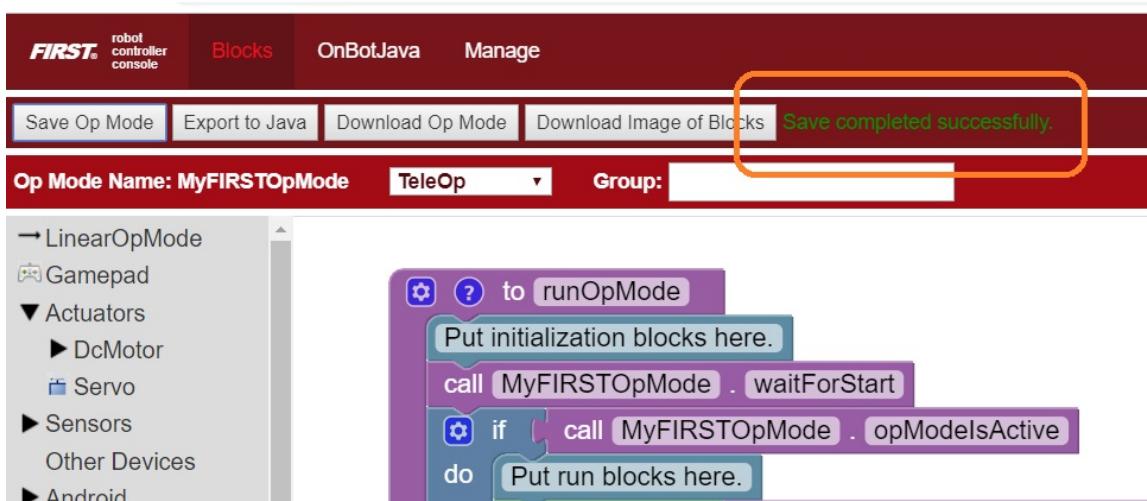


Fig. 13: Message indicating OpMode has been Saved

## 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.

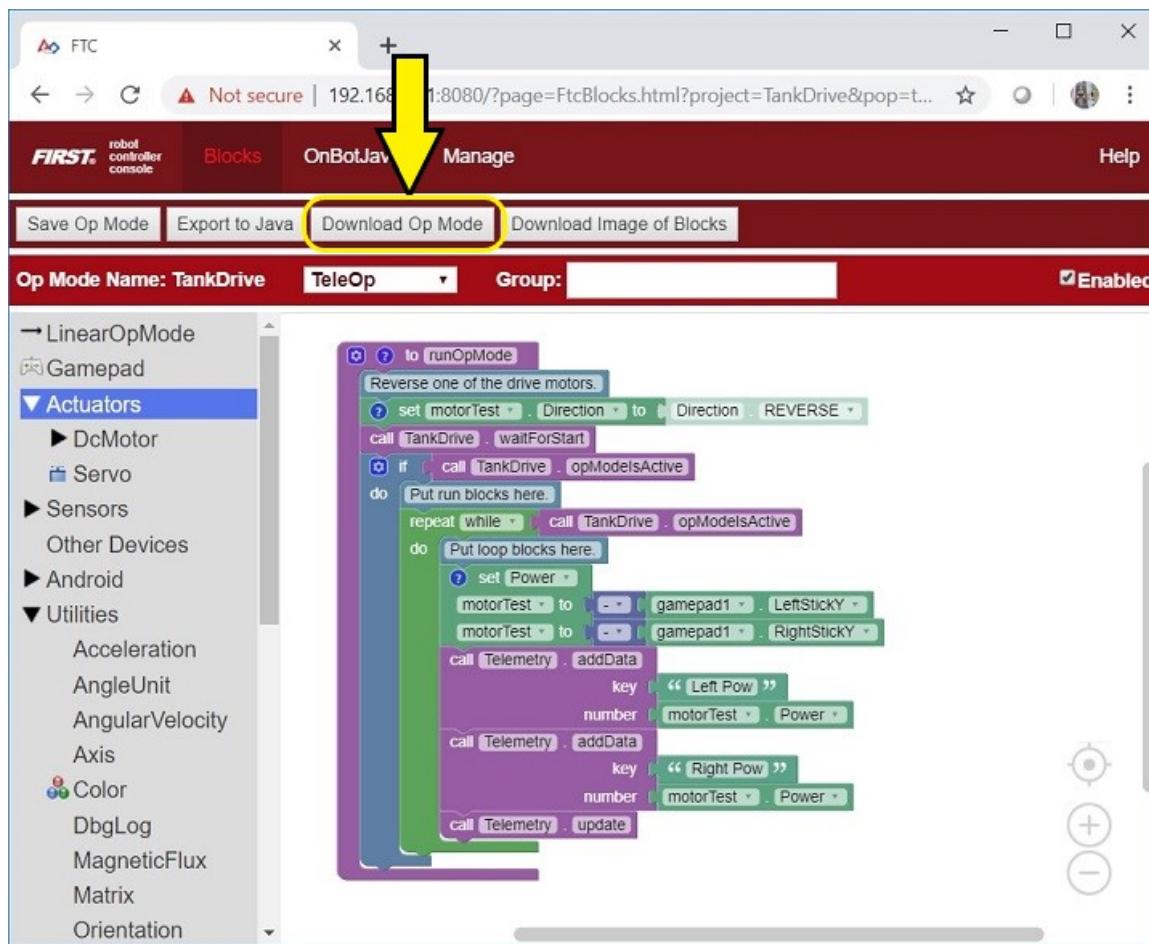


Fig. 14: Downloading a Blocks program

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).

## 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.

<input type="checkbox"/> Op Mode Name	Date Modified▼	Enabled
<input type="checkbox"/> new IMU sample application	November 30, 2022, 2:29:39 PM	<input checked="" type="checkbox"/>
<input type="checkbox"/> Old IMU sample code	November 12, 2022, 6:02:35 AM	<input checked="" type="checkbox"/>
<input type="checkbox"/> TensorFlow Sample OpMode	October 31, 2022, 2:11:38 PM	<input checked="" type="checkbox"/>
<input type="checkbox"/> Mecanum Drive	October 31, 2022, 2:05:29 PM	<input checked="" type="checkbox"/>

Fig. 15: 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.

<input type="checkbox"/> Op Mode Name	Date Modified▼	Enabled
<input type="checkbox"/> new IMU sample application	November 30, 2022, 2:29:39 PM	<input checked="" type="checkbox"/>
<input type="checkbox"/> Old IMU sample code	November 12, 2022, 6:02:35 AM	<input checked="" type="checkbox"/>
<input type="checkbox"/> TensorFlow Sample OpMode	October 31, 2022, 2:11:38 PM	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/> Mecanum Drive	October 31, 2022, 2:05:29 PM	<input checked="" type="checkbox"/>

Fig. 16: 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.

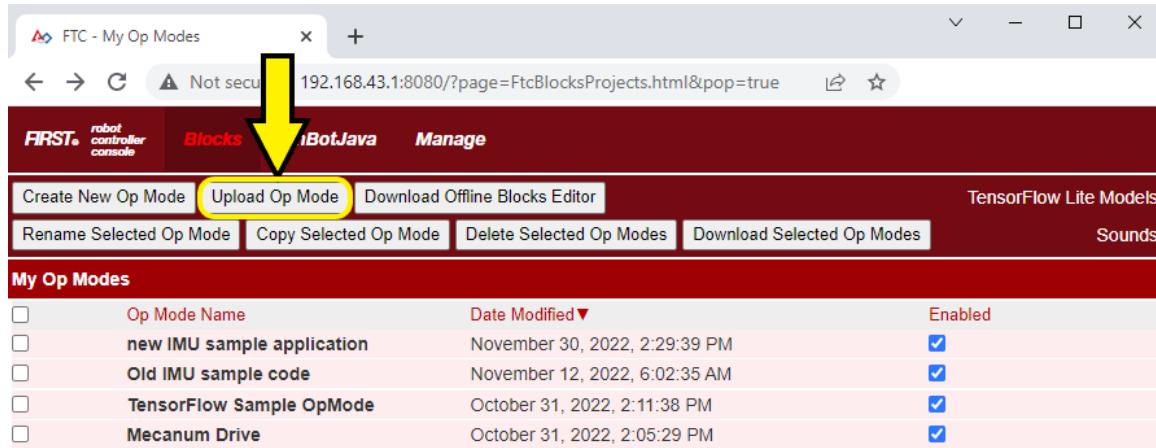


Fig. 17: Uploading Blocks Files via the Management Interface

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.

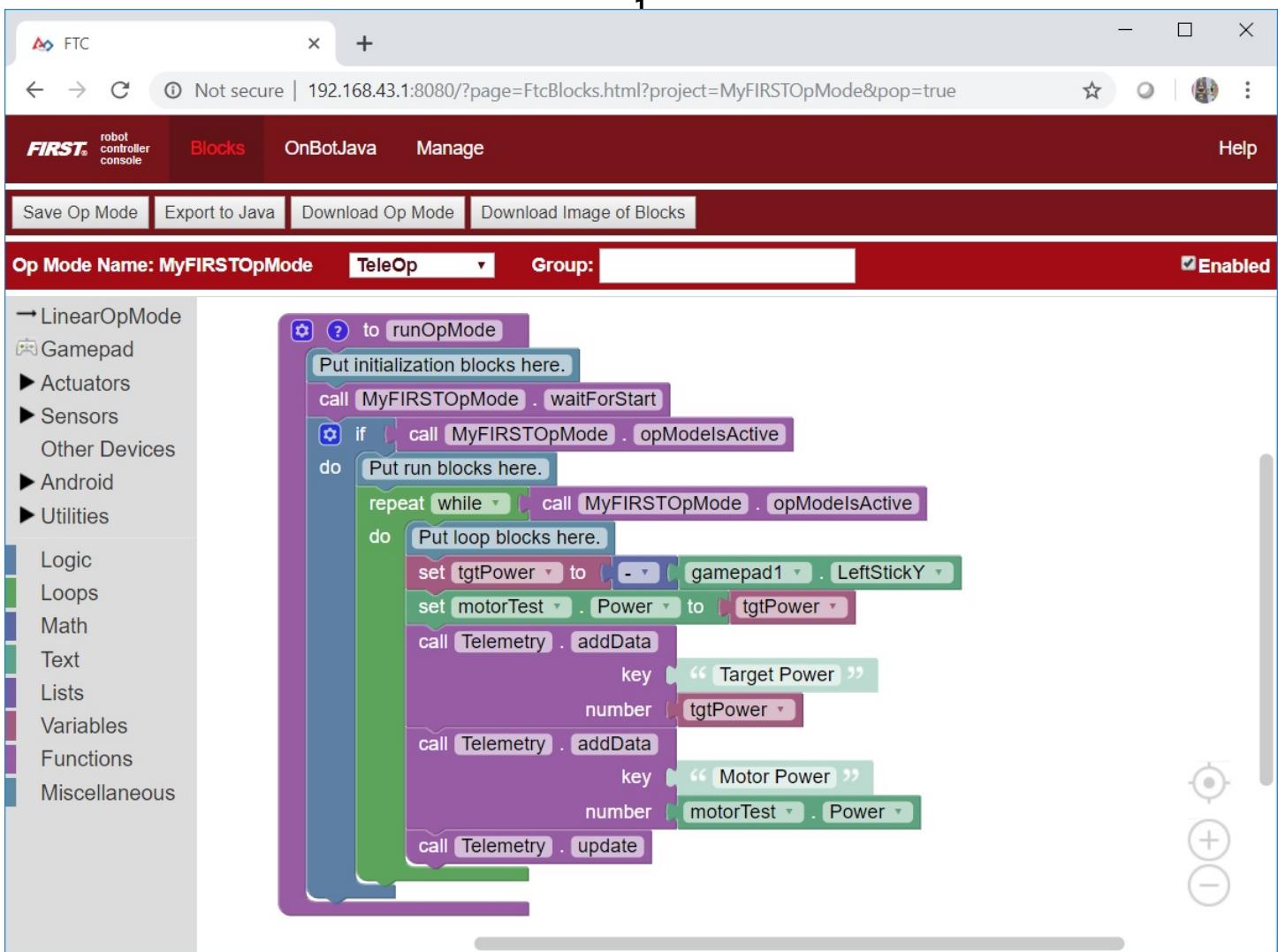


In this example, you will use the colored buttons on the right side of the F310 controller to control the position of the servo. Initially, the op mode will move the servo to the midway position (90 degrees of its 180-degree range). Pushing the yellow "Y" button will move the servo to the zero-degree 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.



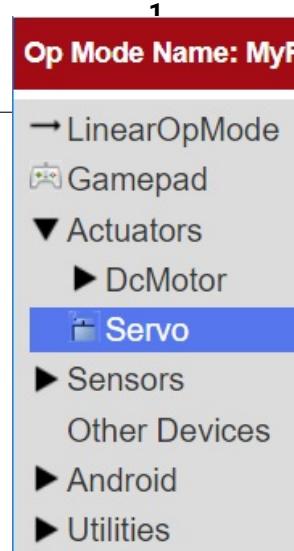
### Modifying the Op Mode to Control a Servo Motor Instructions

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.

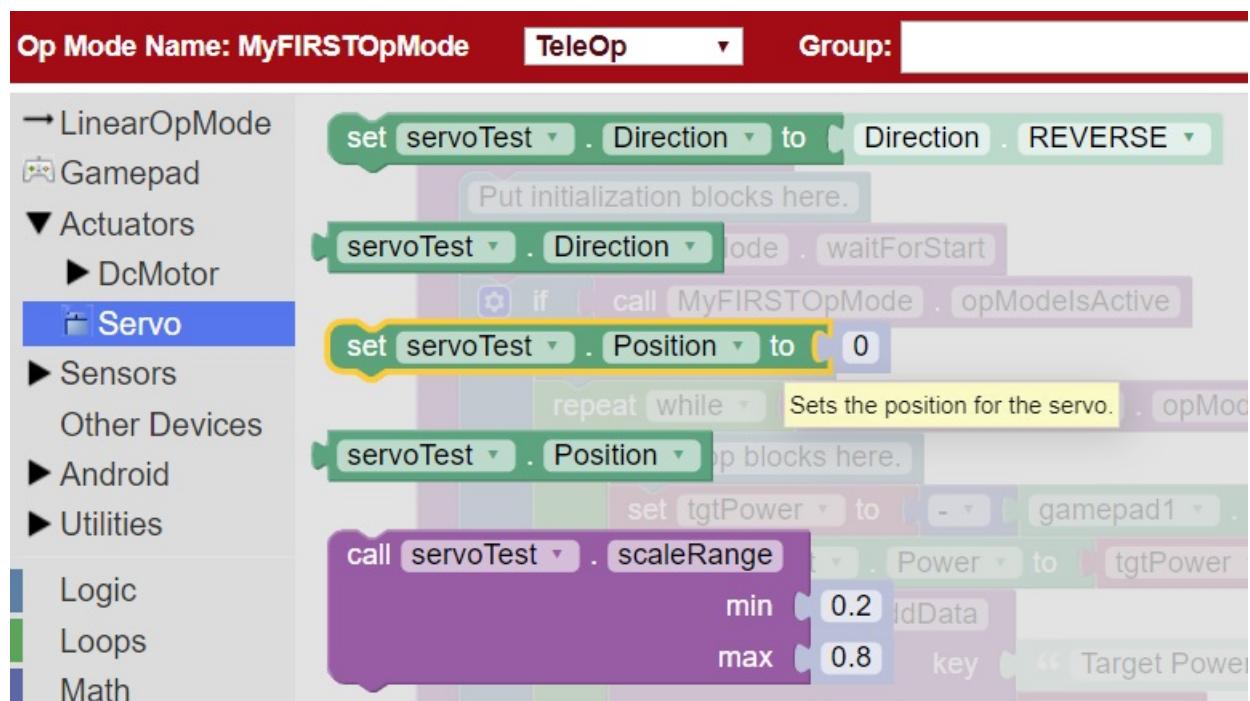


Click on the “MyFIRSTOpMode” project to open it for editing if it is not already opened.

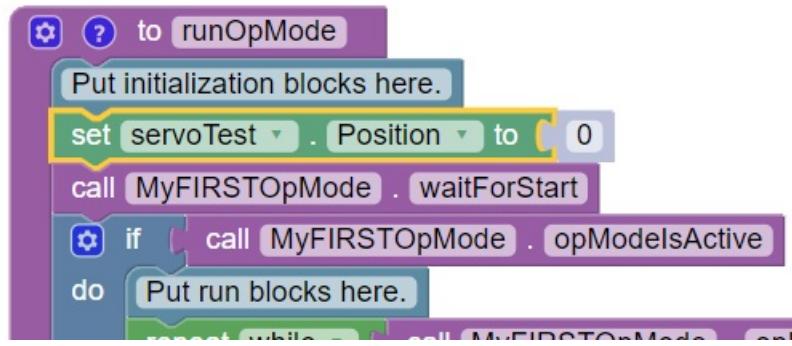
3. On the left-hand side of the screen click on the category called “Actuators” and look for the subcategory called “Servos”.



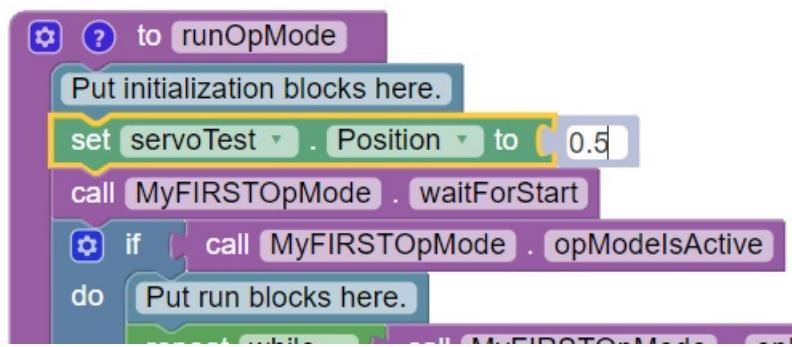
4. Select the "set servoTest.Position to" block from the list of available Servo blocks.



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.

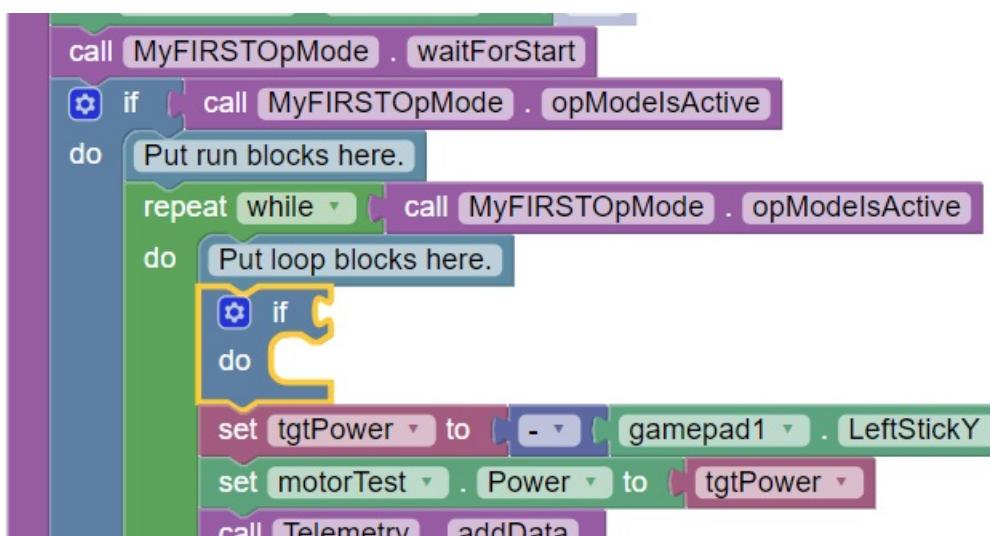


6. Click on the number block "0" and change the block's value to "0.5".



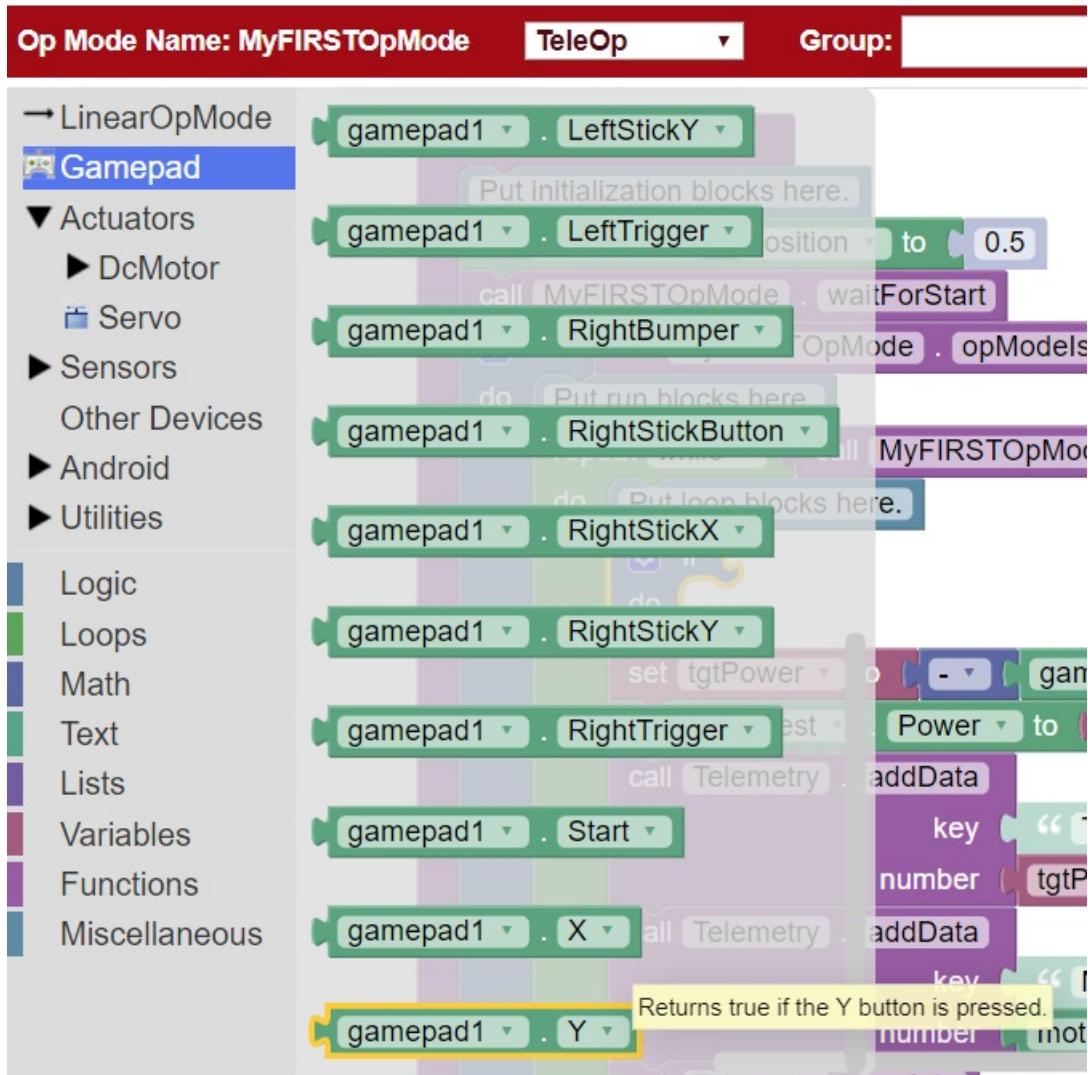
When a user selects this op mode, the servo position will initially be set to the midway point (90-degree position).

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."



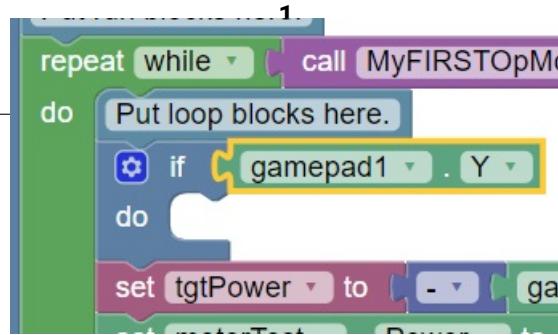
The block should click into place.

- 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.

- 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.

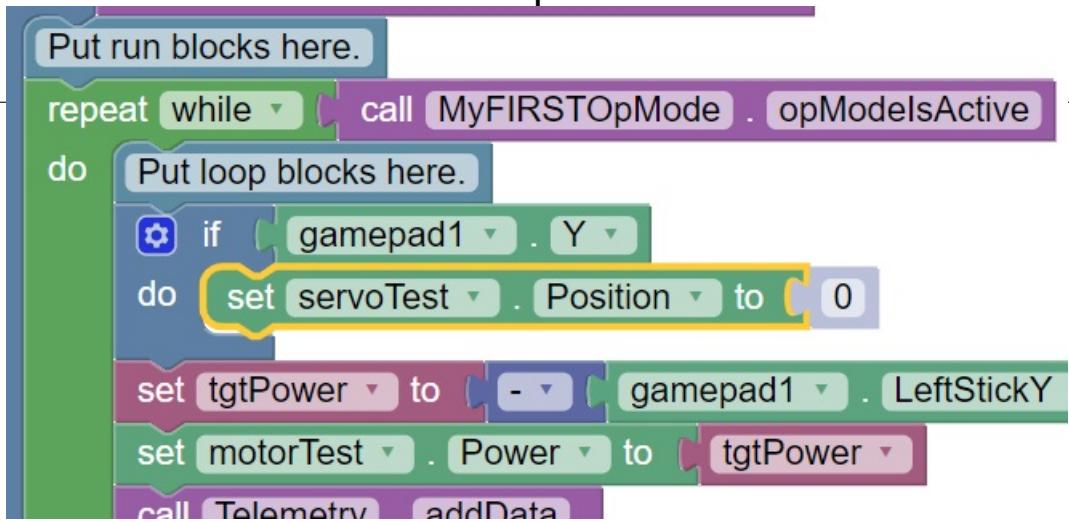
10. On the left-hand side of the screen click on the category called “Actuators” and look for the subcategory called “Servos”.



11. Select the “set servoTest.Position to” block from the list of available Servo blocks.

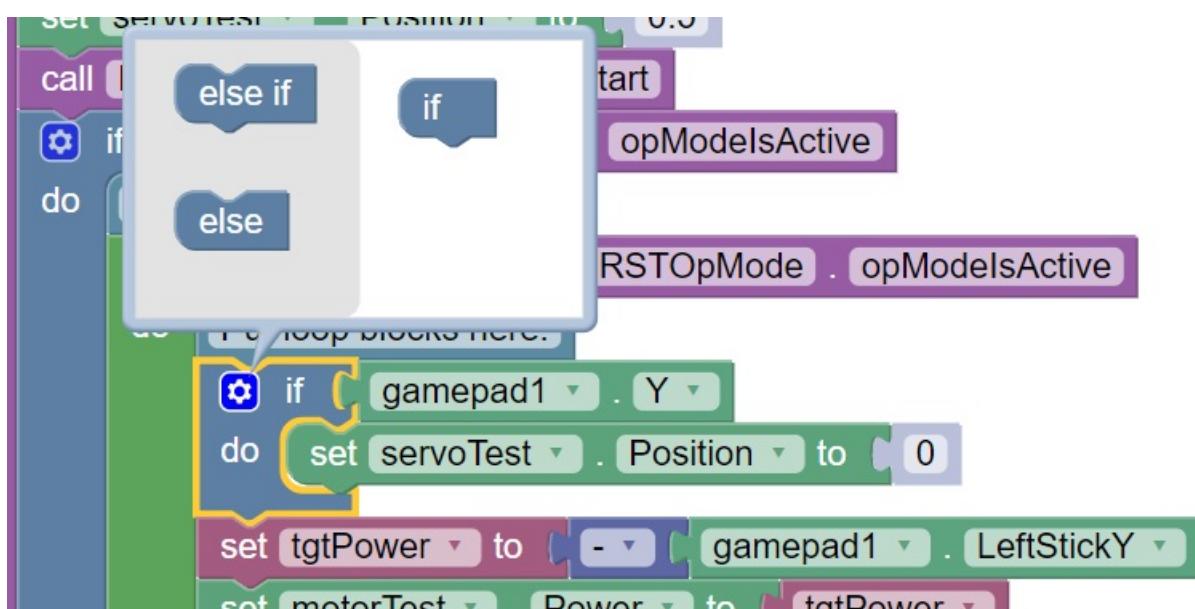


12. Drag the “set servoTest.Position to” block so that it snaps in place in the do portion of the “if do” block.

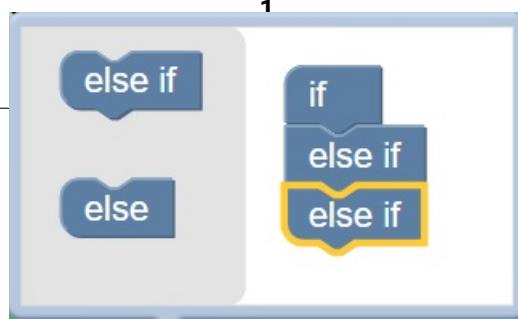


If the "Y" button is pressed on gamepad #1, the op mode will move the servo's position to the 0-degree position.

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.

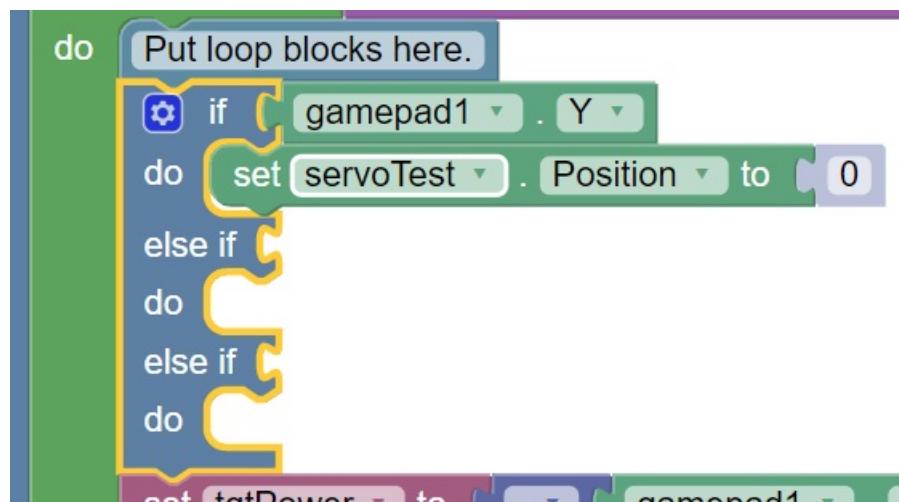


14. Drag an "else if" block from the left side of the pop-up menu and snap it into place under the "if" block.

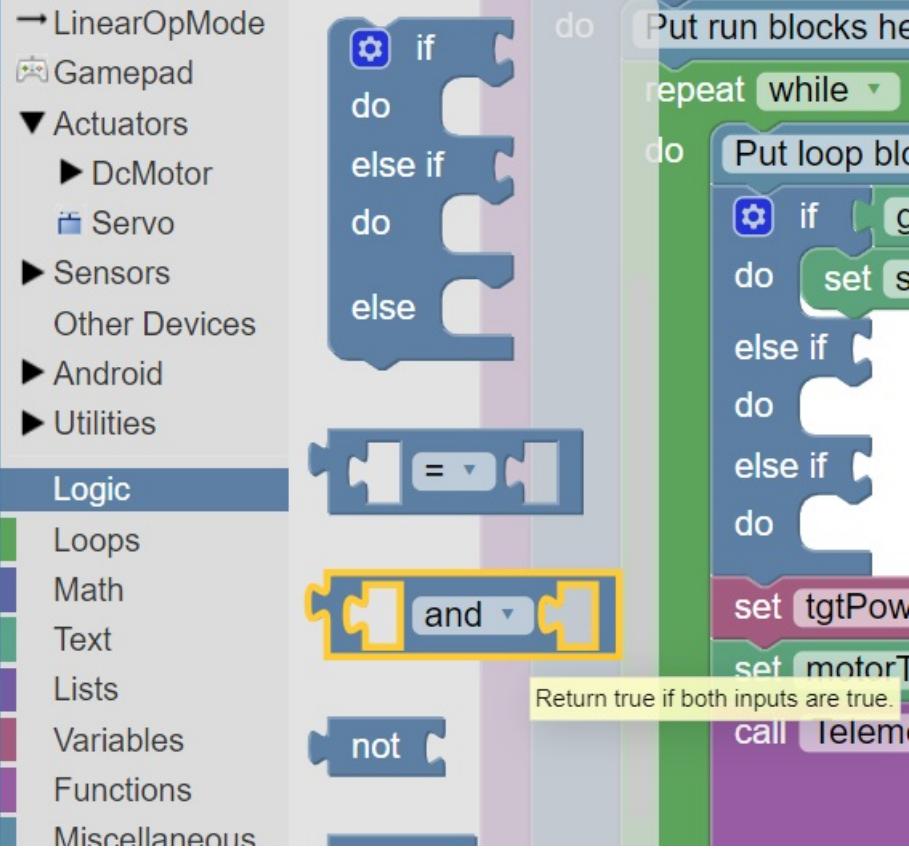


Drag a second "else if" block from the left side and snap it into place on the right side under the first "else if" block.

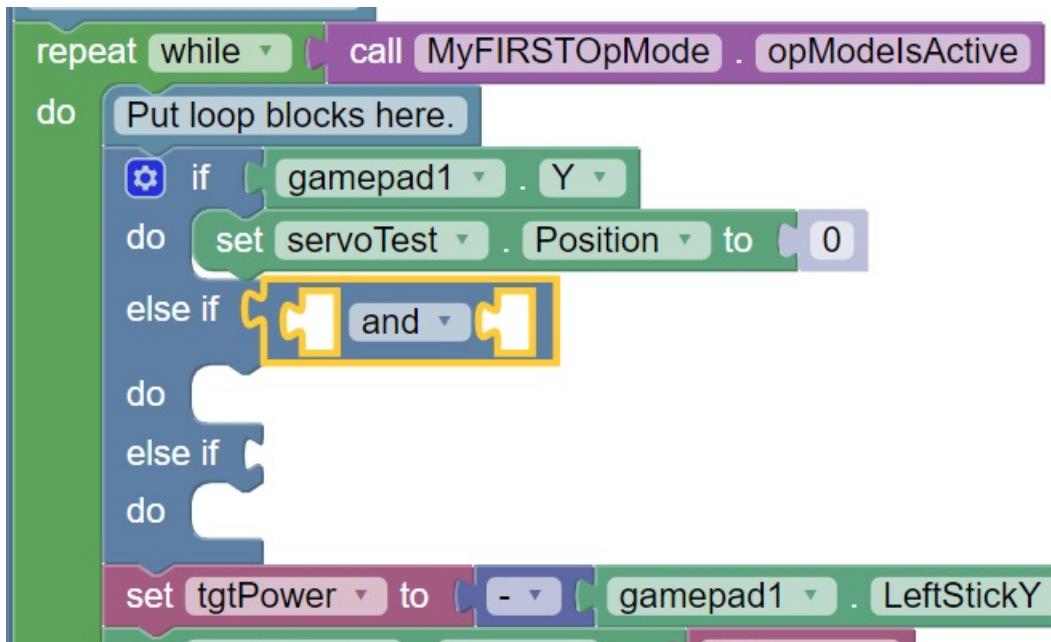
15. Click on the Settings icon to hide the pop-up menu for the "if do" block. The "if do" block should now have two "else if" test conditions added.



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.



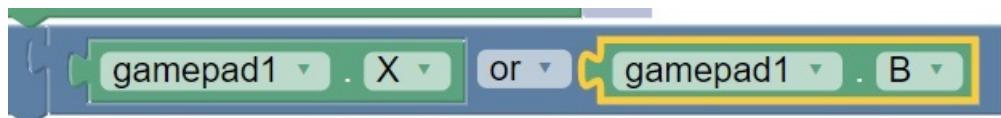
18. Click on the word “and” and select “or” from the pop-up menu to change the block to a logical “or” block.



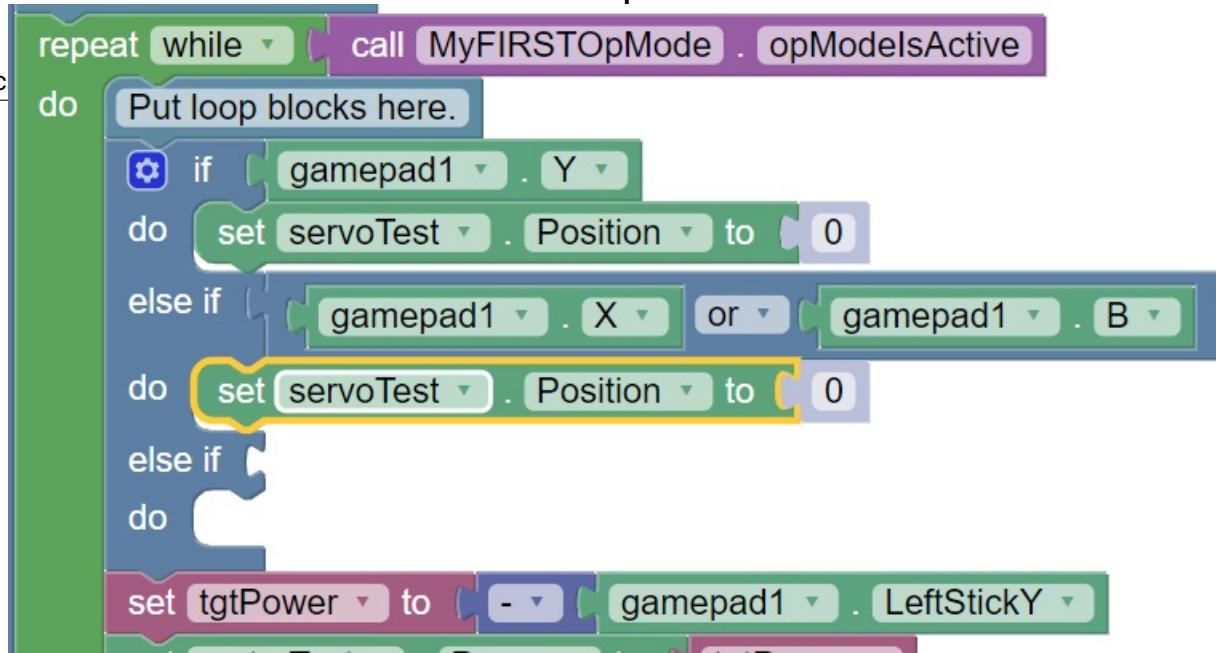
19. Click on the “Gamepad” category and select the “gamepad1.X” block. Drag the block so that it clicks in place as the first test condition of the logical “or” block.



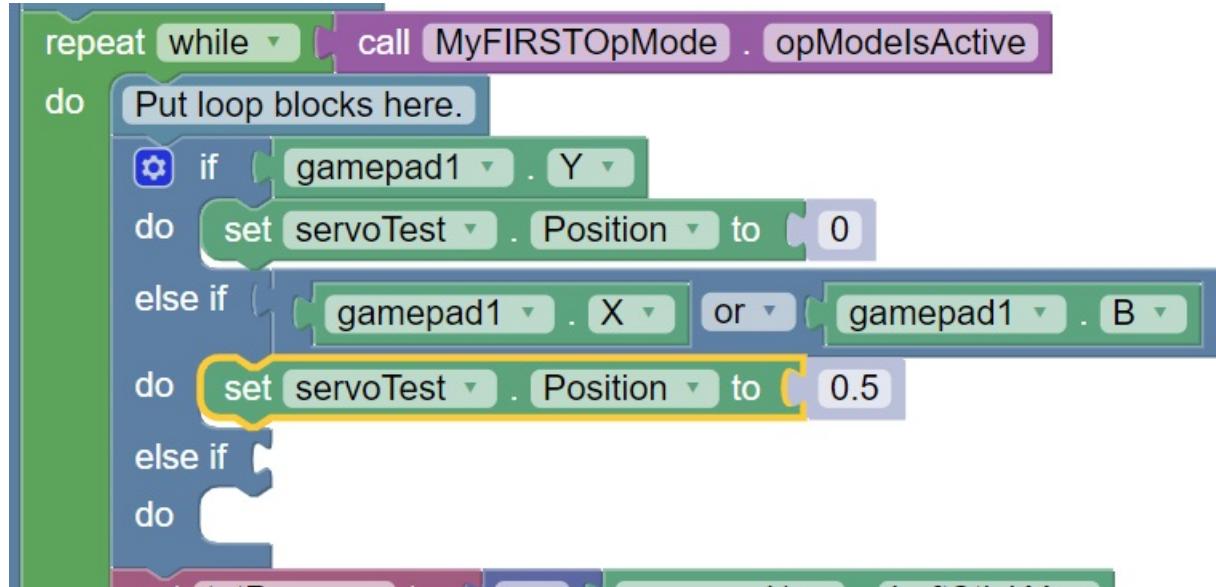
20. Click on the “Gamepad” category and select the “gamepad1.B” block. Drag the block so that it clicks in place as the second test condition of the logical “or” block.



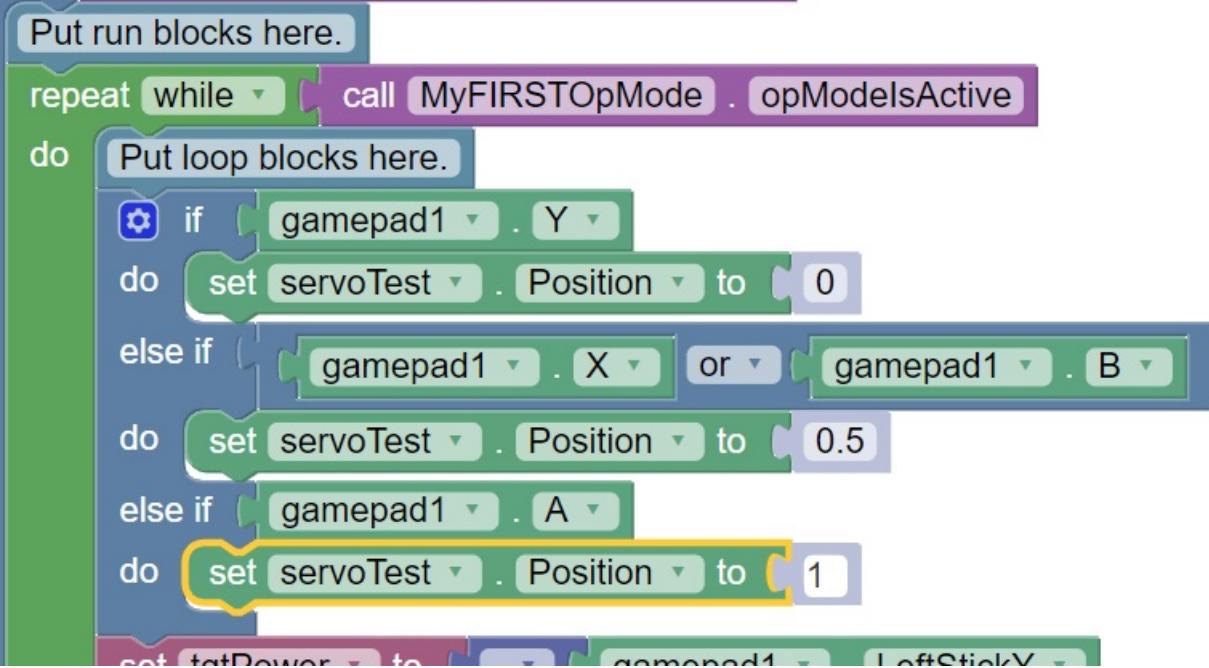
21. Select a “set servoTest.Position to” block and place it into “do” clause of the first else-if block.



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.

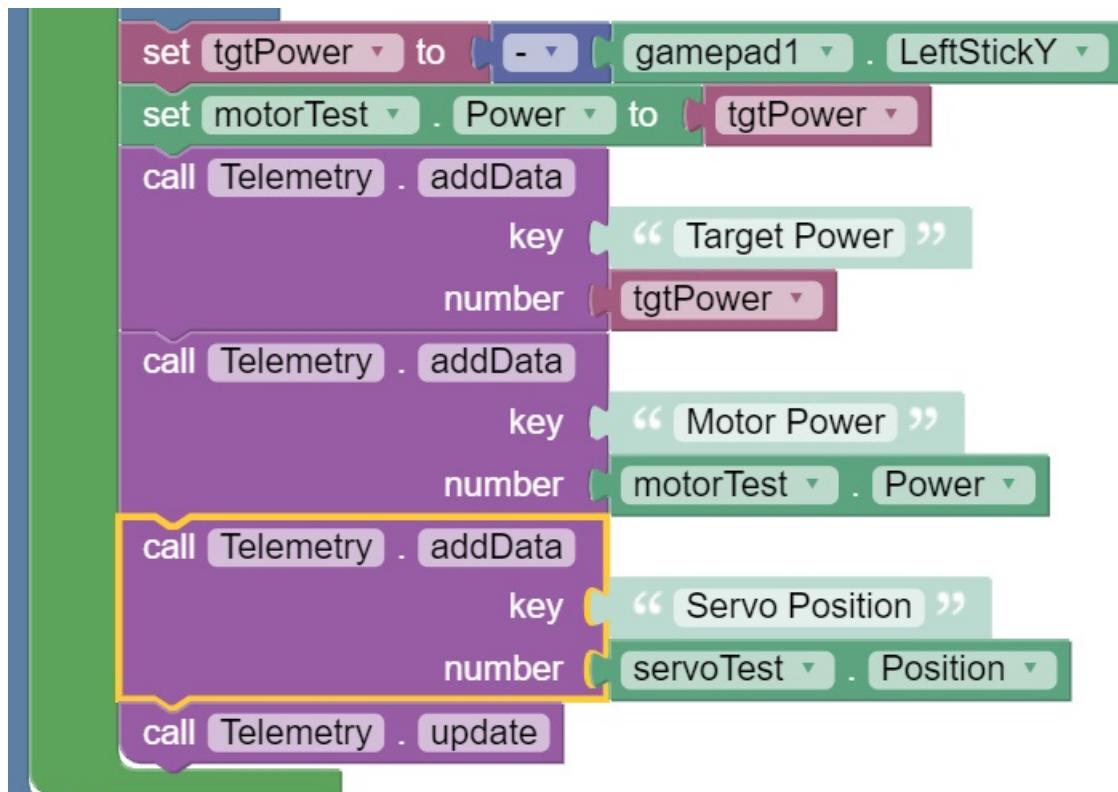


23. Use a "gamepad1.A" block as the test condition for the second "else if" block. Drag a "set servoTest.position to" block to the do clause of the second "else if" block and modify the numeric value so that the servo's position will be set to a value of 1.



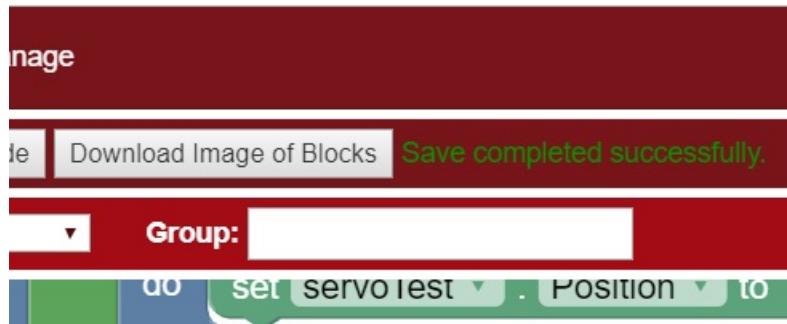
For this clause, if the "A" button is pressed on the #1 gamepad, the op mode will move the servo to the 180-degree position.

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.

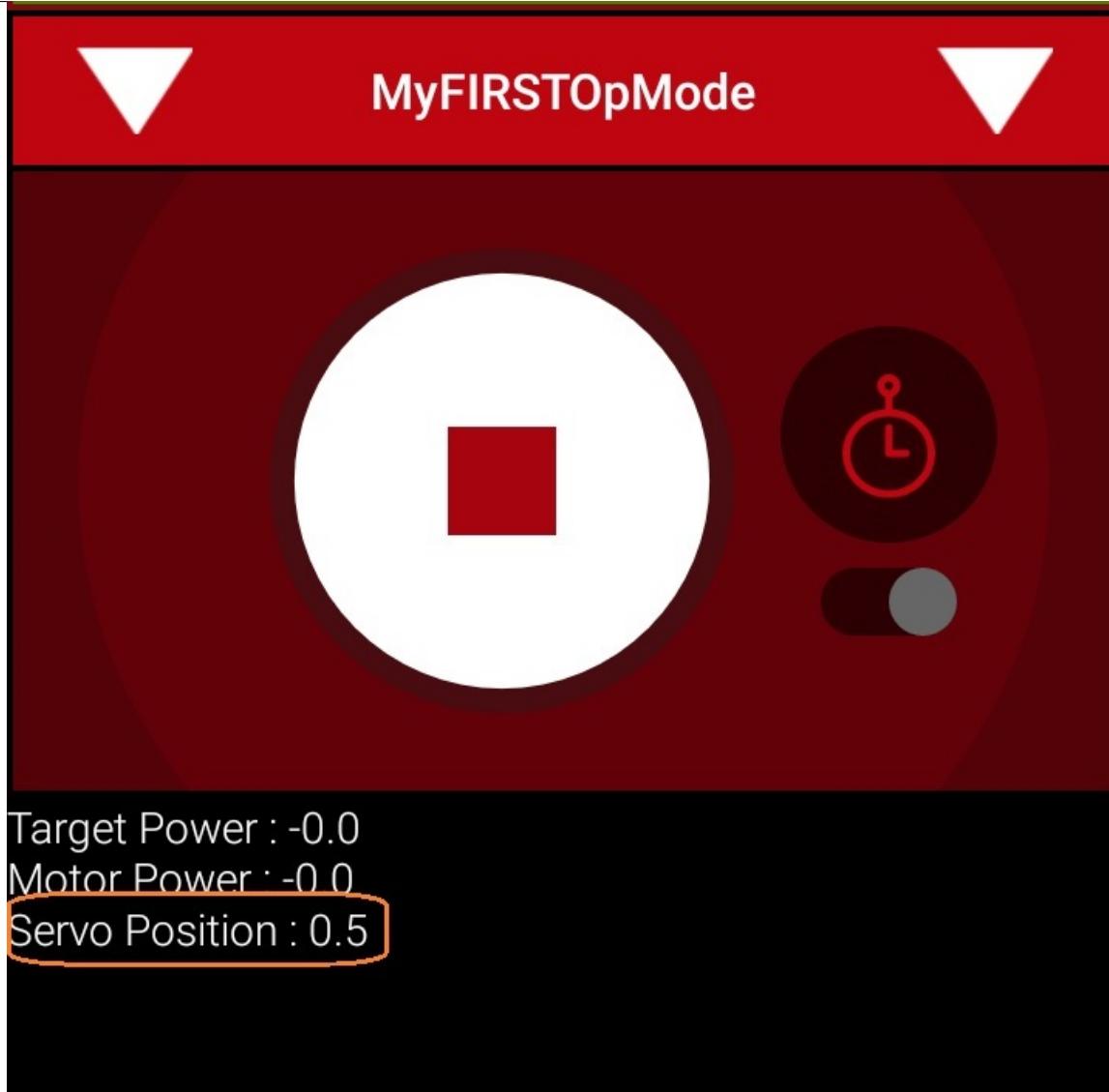


This set of blocks will send the current servo position value to the DRIVER STATION while the op mode is running.

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.

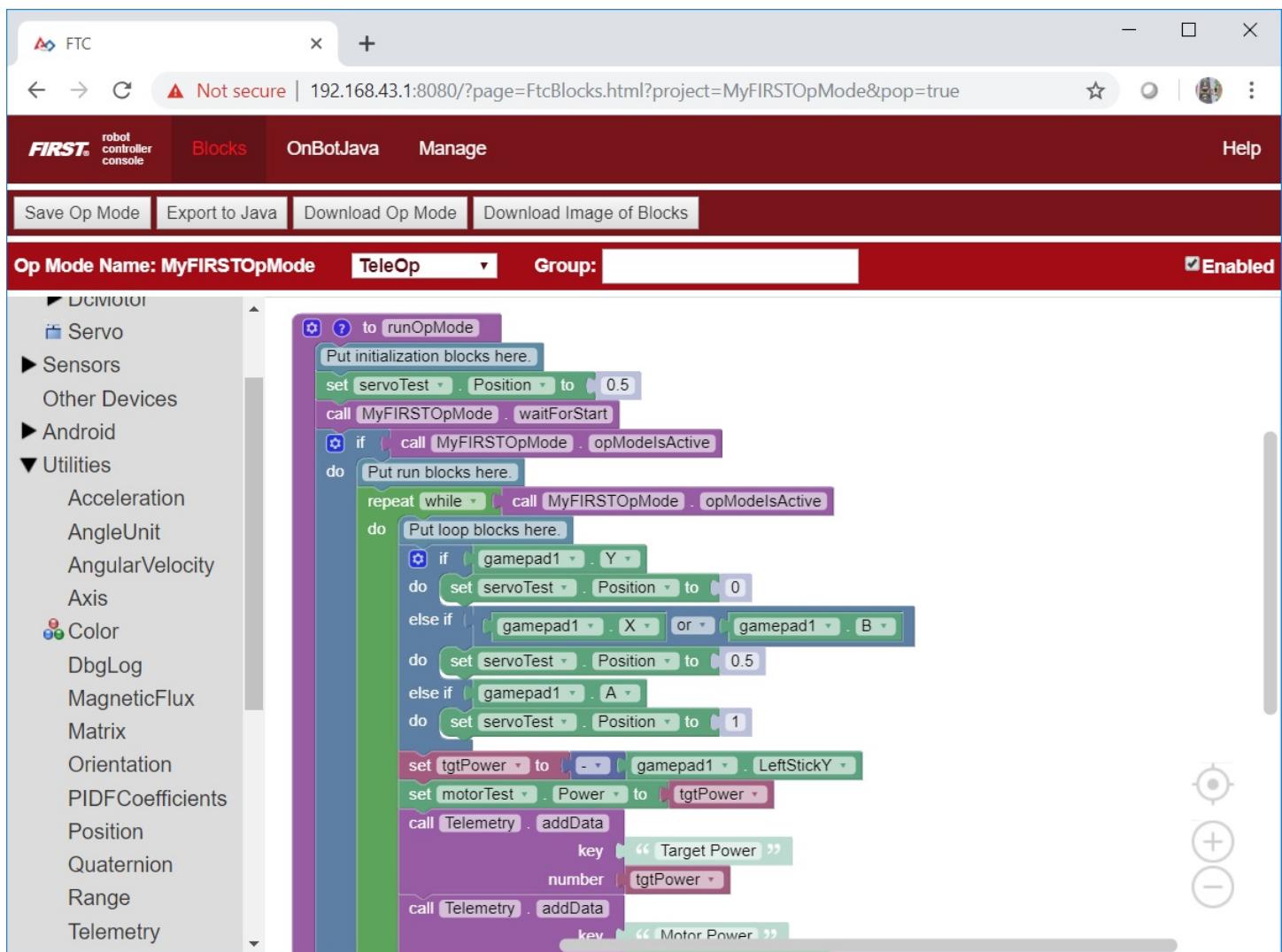
The Color-Range sensor uses reflected light to determine the distance from the sensor to the target object. It can be used to measure close distances (up 5" or more) with reasonable accuracy. Note that at the time this document was most recently edited, the REV Color-Range sensor saturates around 2" (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

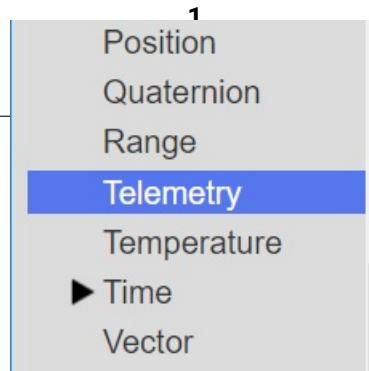
### FTC Docs

- Verify that your laptop is still connected to the Robot Controller's Program & Manage Wi-Fi network.
- 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.

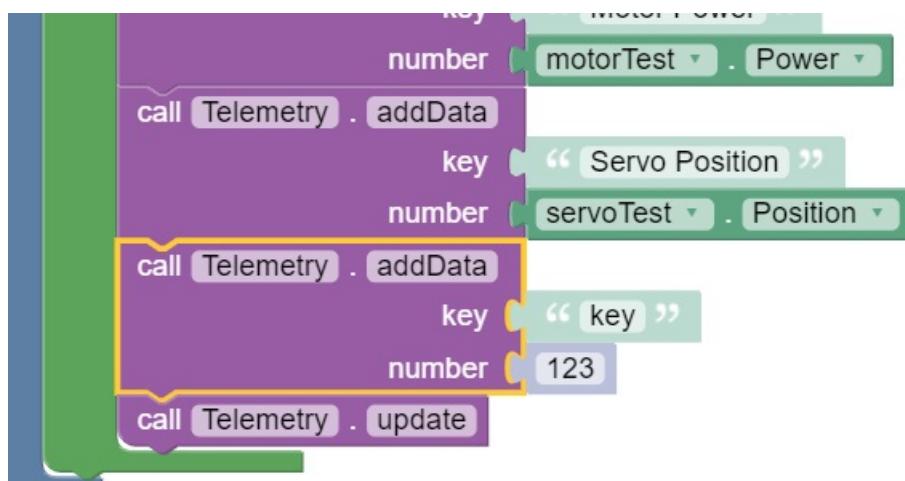


Click on the "MyFIRSTOpMode" project to open it for editing if it is not already opened.

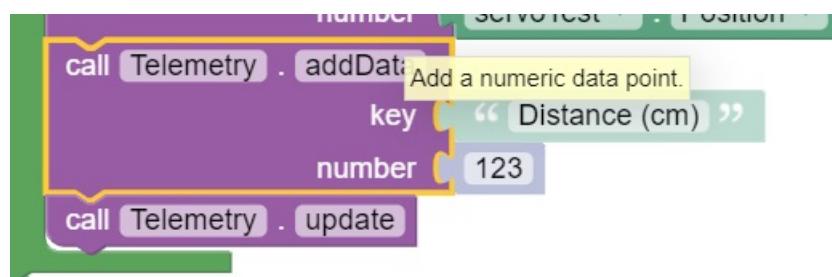
- Click on the "Utilities" category on the left-hand side of your browser. Find and click on the "Telemetry" subcategory.



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.

Op Mode Name: MyFIRSTOpMode

TeleOp

Group: 1

→ LinearOpMode

Gamepad

▼ Actuators

► DcMotor

Servo

▼ Sensors

IMU-BNO055

IMU-BNO055.Parameters

REV Color/Range Sensor

TouchSensor

VoltageSensor

Other Devices

► Android

▼ Utilities

Acceleration

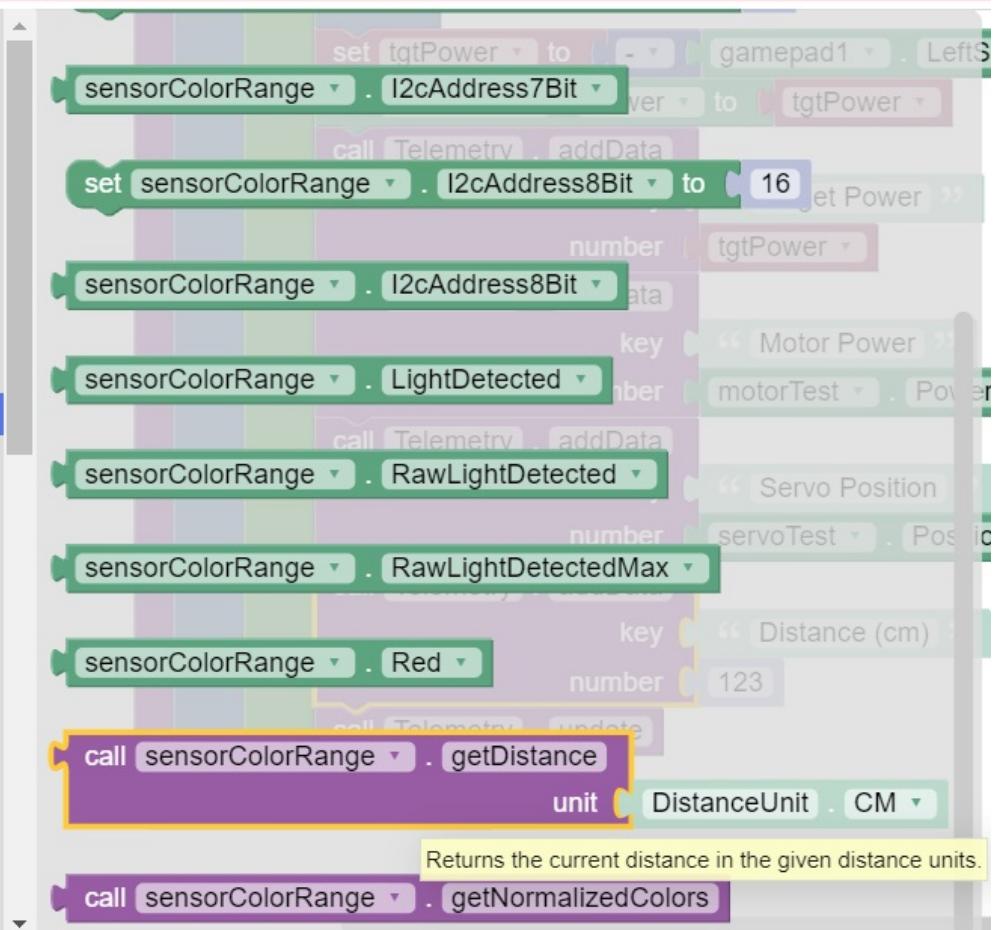
AngleUnit

AngularVelocity

Axis

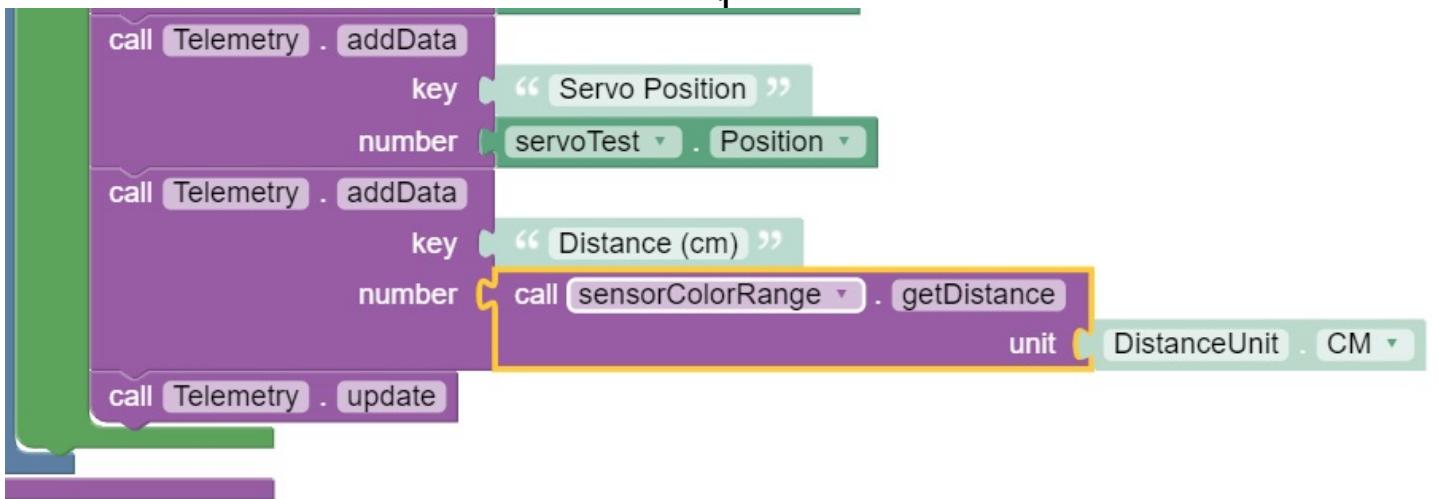
Color

Digital



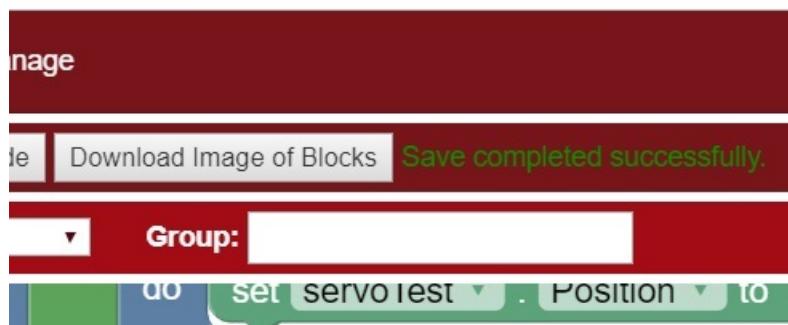
Note that earlier versions of the Blocks Programming tool refer to the REV Robotics Color-Distance Sensor as the "Lynxl2cColorRangeSensor". Newer versions of the software refer to the device as the "REV Color/Range Sensor".

7. Drag the "call sensorColorRange.getDistance" programming block to the "number" field of the "call telemetry.addData" programming block.

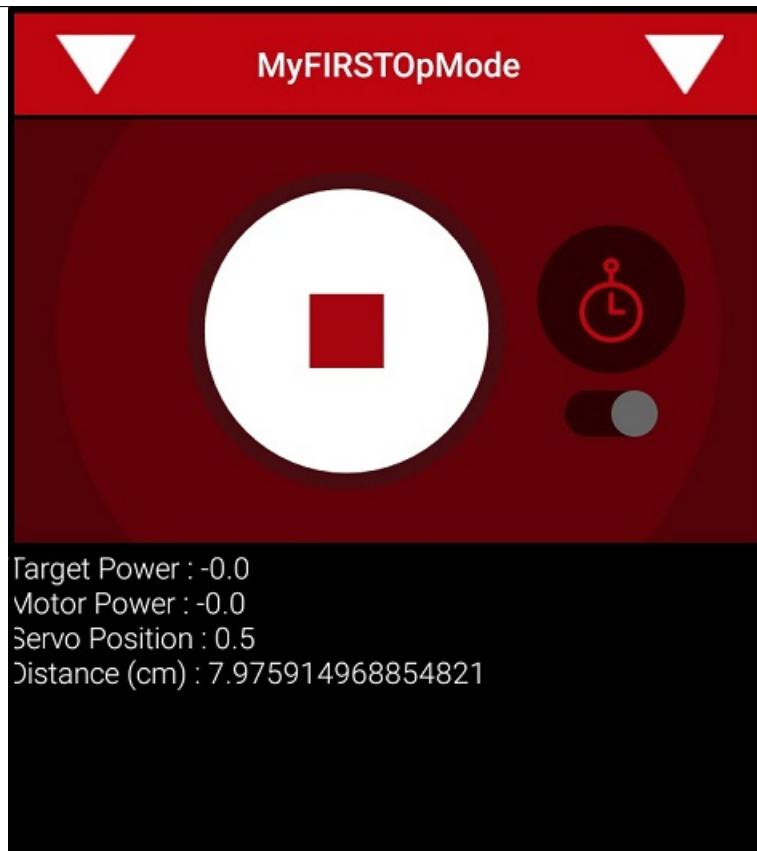


This will send the measured distance to the target in centimeters back to the DRIVER STATION.

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 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 an 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 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.

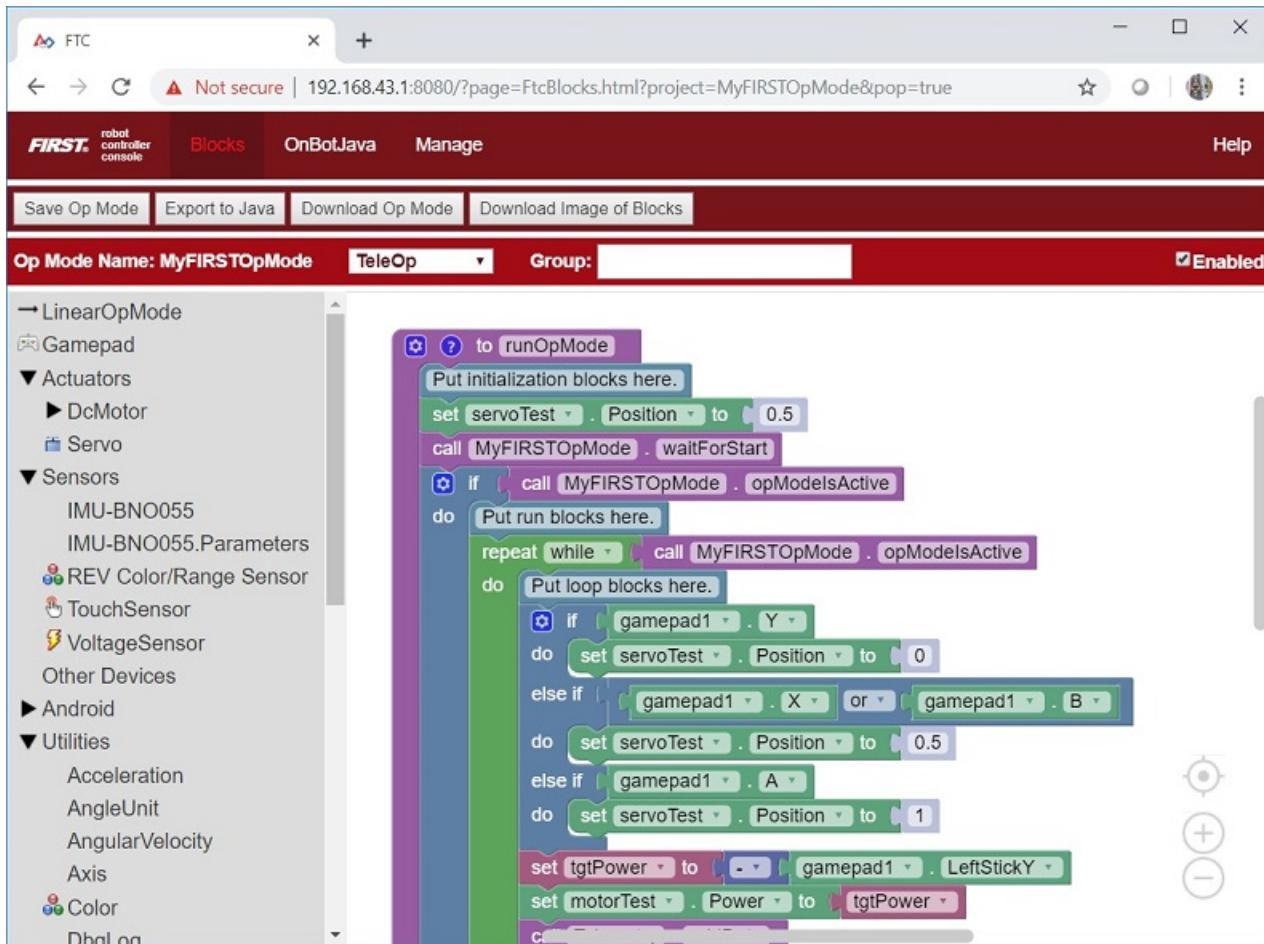
FTC Docs

Note that it will take an estimated 15 minutes to complete this task.

**FTC Programming Resources, 206**

## 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.

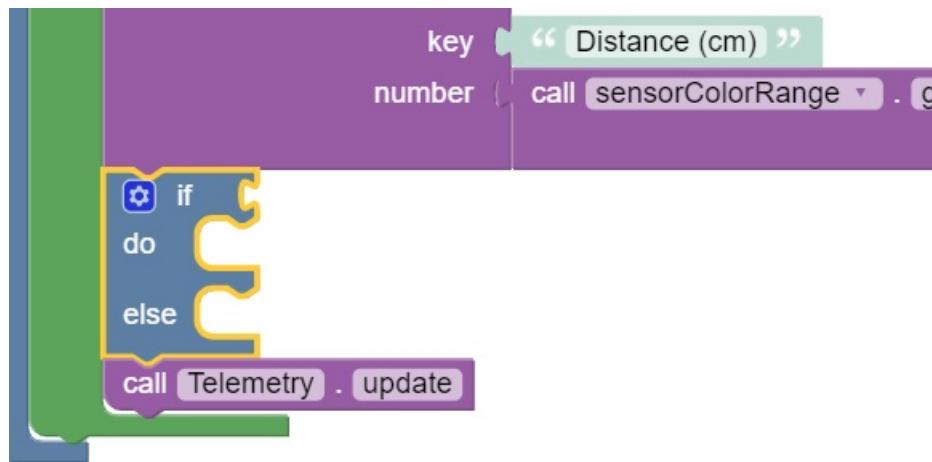


Click on the “MyFIRSTOpMode” project to open it for editing if it is not already opened.

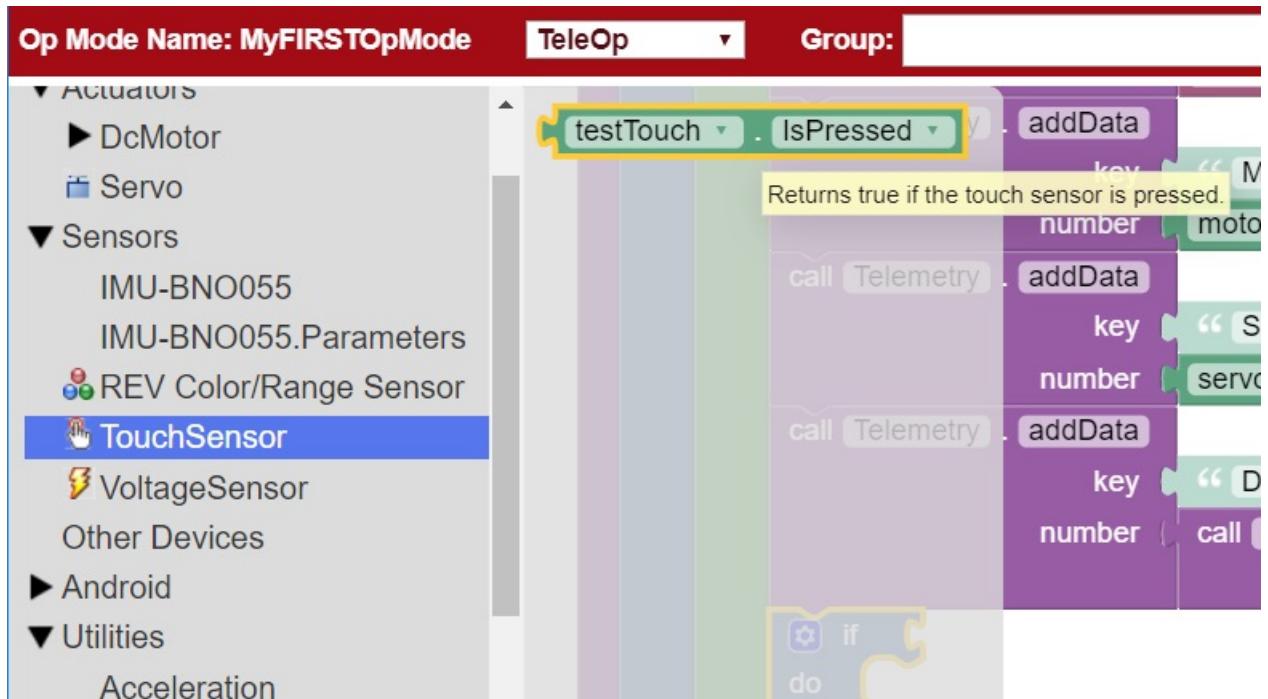
3. Click on the “Logic” category. Find and click on the “if do else” block.



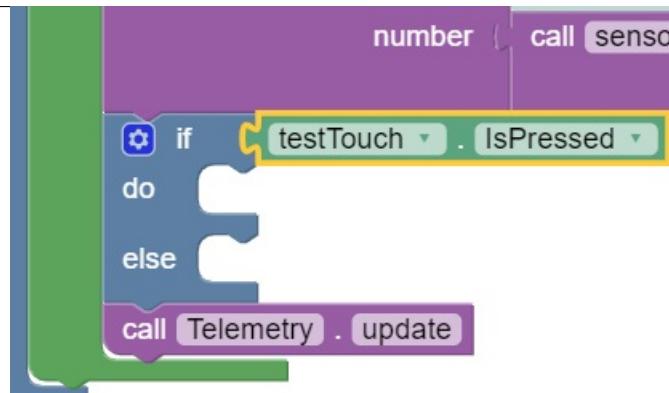
4. Drag the “if do else” block to the position before the “telemetry.update” block.



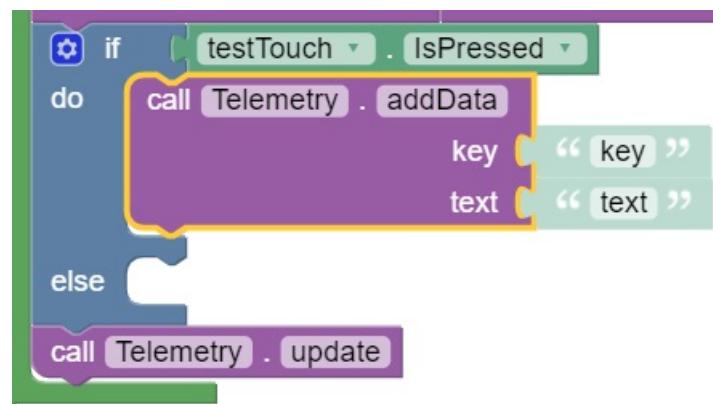
5. Click on the “Sensors” category to expand it (if it isn’t already expanded). Click on the “Touch Sensor” subcategory, then find and select the “.isPressed” block.



6. Drag the “isPressed” block to the test condition of the “if do else” programming block.

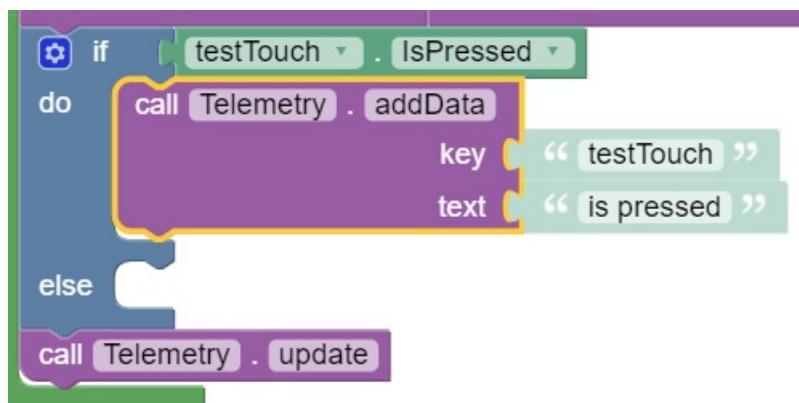


7. Click on the “Utilities” category on the left-hand side of your browser. Find and click on the “Telemetry” subcategory.



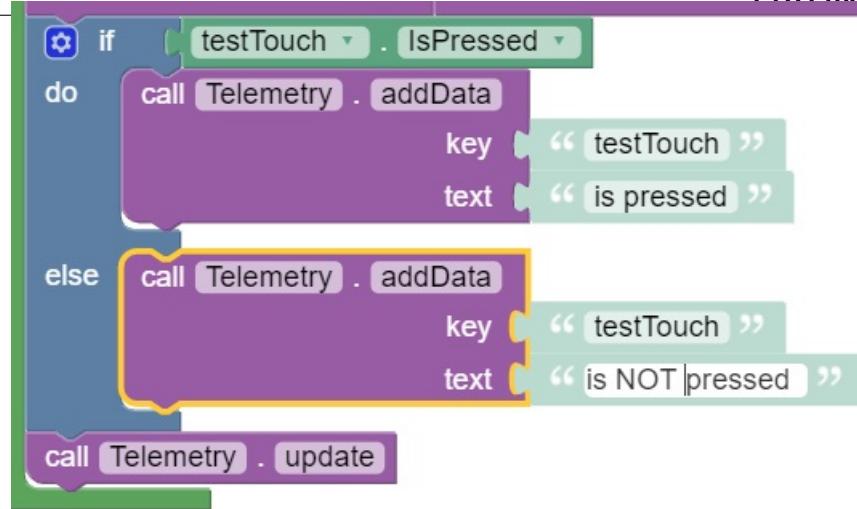
Select the “call telemetry.addData” block (the text version) and drag it to the “do” clause of the “if do else” block.

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”.<sup>1</sup>  
FTC Docs

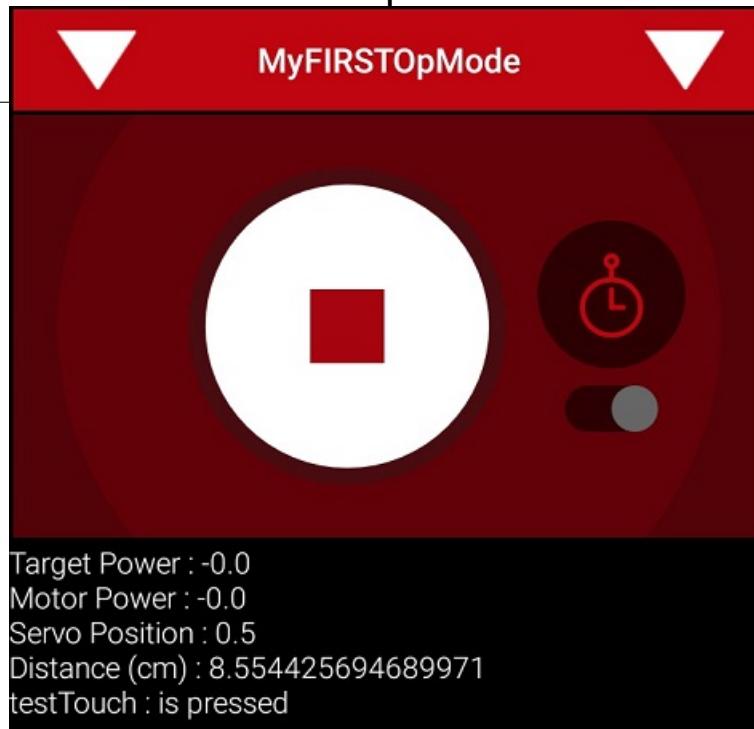
FTC Programming Resources, 209



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.

## 1.2.5 Reference Documents Blocks

### Blocks Reference Materials Blocks

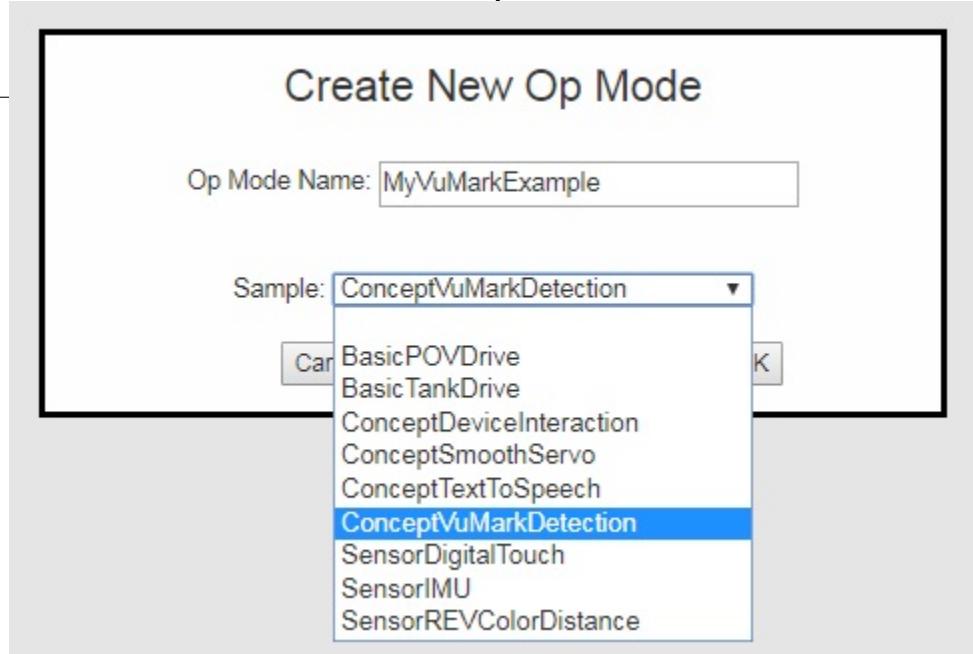
### Blocks Reference Manual

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 Expansion Hub Documentation

[REV Robotics Expansion Hub Getting Started Guide](#)

## 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

### 1.3.1 Introduction OBJ

### 1.3.2 Configuring Your Hardware OBJ

### 1.3.3 Connecting to the Program & Manage Server OBJ

### 1.3.4 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?

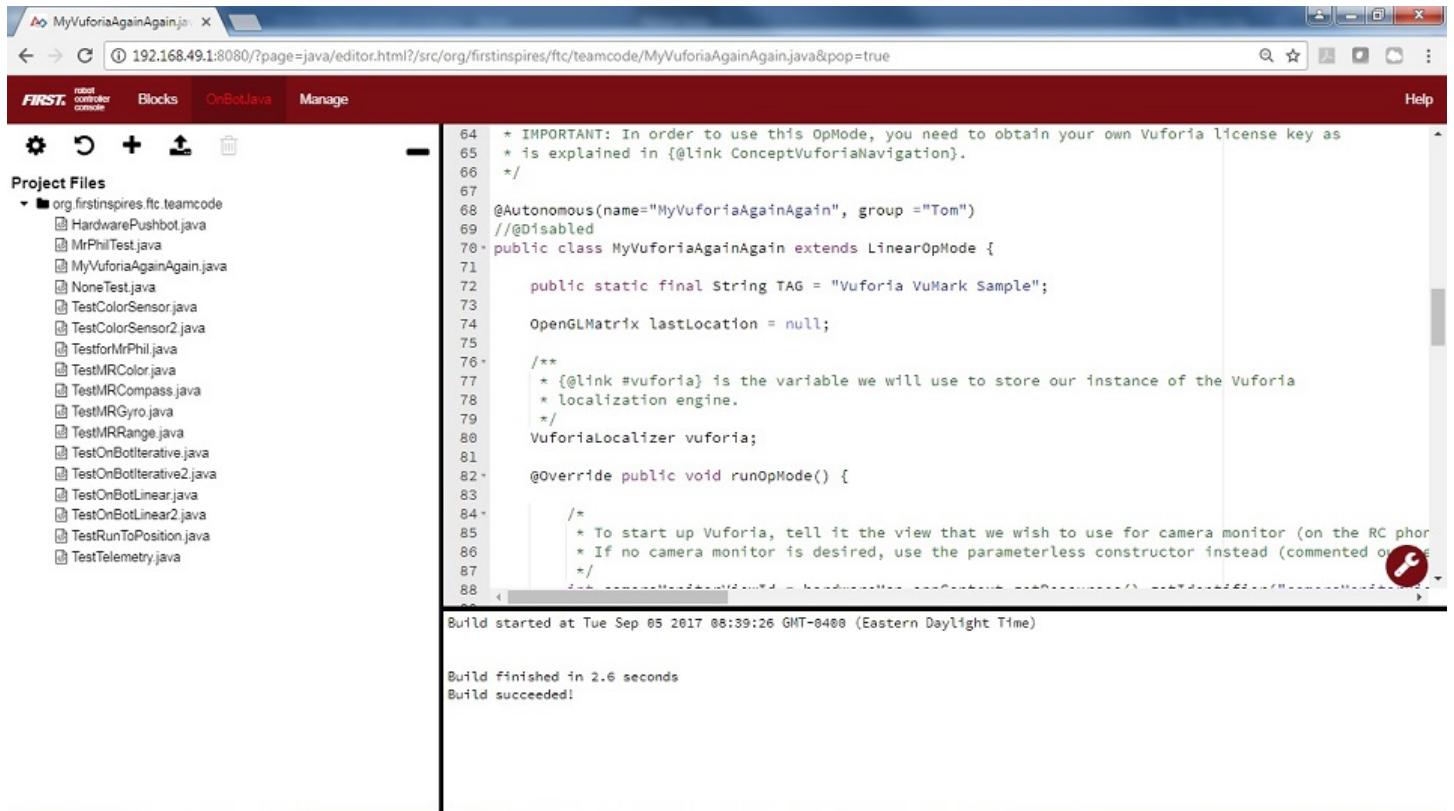
### FTC Docs

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 screenshot shows a web-based Java editor interface for the OnBot Java Programming Tool. The URL is 192.168.49.1:8080/?page=java/editor.html?/src/org/firstinspires/ftc/teamcode/MyVuforiaAgainAgain.java&pop=true. The interface includes a top navigation bar with FIRST, robot controller, Blocks, OnBotJava, Manage, and Help. Below the navigation is a toolbar with icons for settings, back, forward, search, and other functions. The main area has a code editor with the following Java code:

```

64 * IMPORTANT: In order to use this OpMode, you need to obtain your own Vuforia license key as
65 * is explained in {@link ConceptVuforiaNavigation}.
66 */
67
68 @Autonomous(name="MyVuforiaAgainAgain", group ="Tom")
69 //Disabled
70 public class MyVuforiaAgainAgain extends LinearOpMode {
71
72     public static final String TAG = "Vuforia VuMark Sample";
73
74     OpenGLMatrix lastLocation = null;
75
76     /**
77      * {@link #vuforia} is the variable we will use to store our instance of the Vuforia
78      * localization engine.
79     */
80     VuforiaLocalizer vuforia;
81
82     @Override public void runOpMode() {
83
84         /*
85          * To start up Vuforia, tell it the view that we wish to use for camera monitor (on the RC phone).
86          * If no camera monitor is desired, use the parameterless constructor instead (commented out).
87         */
88     }

```

Below the code editor, a message says "Build started at Tue Sep 05 2017 08:39:26 GMT-0400 (Eastern Daylight Time)". At the bottom, it says "Build finished in 2.6 seconds" and "Build succeeded!"

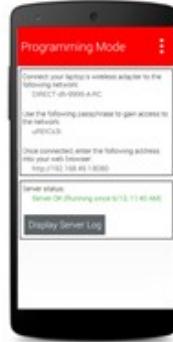
The examples in this document use a Windows laptop computer to connect to the Robot Controller. This Windows laptop computer has a Javascript-enabled web browser installed that is used to access the OnBot Java Programming Tool.



Laptop



WiFi Connection



Robot Controller

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.



Laptop



WiFi Connection



Control Hub

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 Program & 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.

**FTC Programming Resources 215**

### 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").

To *remotely* connect to the controller, connect your laptop's wireless adapter to this network, using the passphrase to gain access. Once connected, enter the following address into your web browser:

**http://192.168.43.1:8080**

Robot controller status:

**Server OK (Running since Dec 31, 7:00 PM)**

Type this web address into the address field of your browser and press RETURN to navigate to the Program & Manage web server.

FIRST robot controller console

Blocks    OnBotJava    Manage

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.

**Robot Controller Connection Info**

The connected robot controller resides on the wireless network named:  
**FTC-1Ybr**

The passphrase for this network is:  
**password**

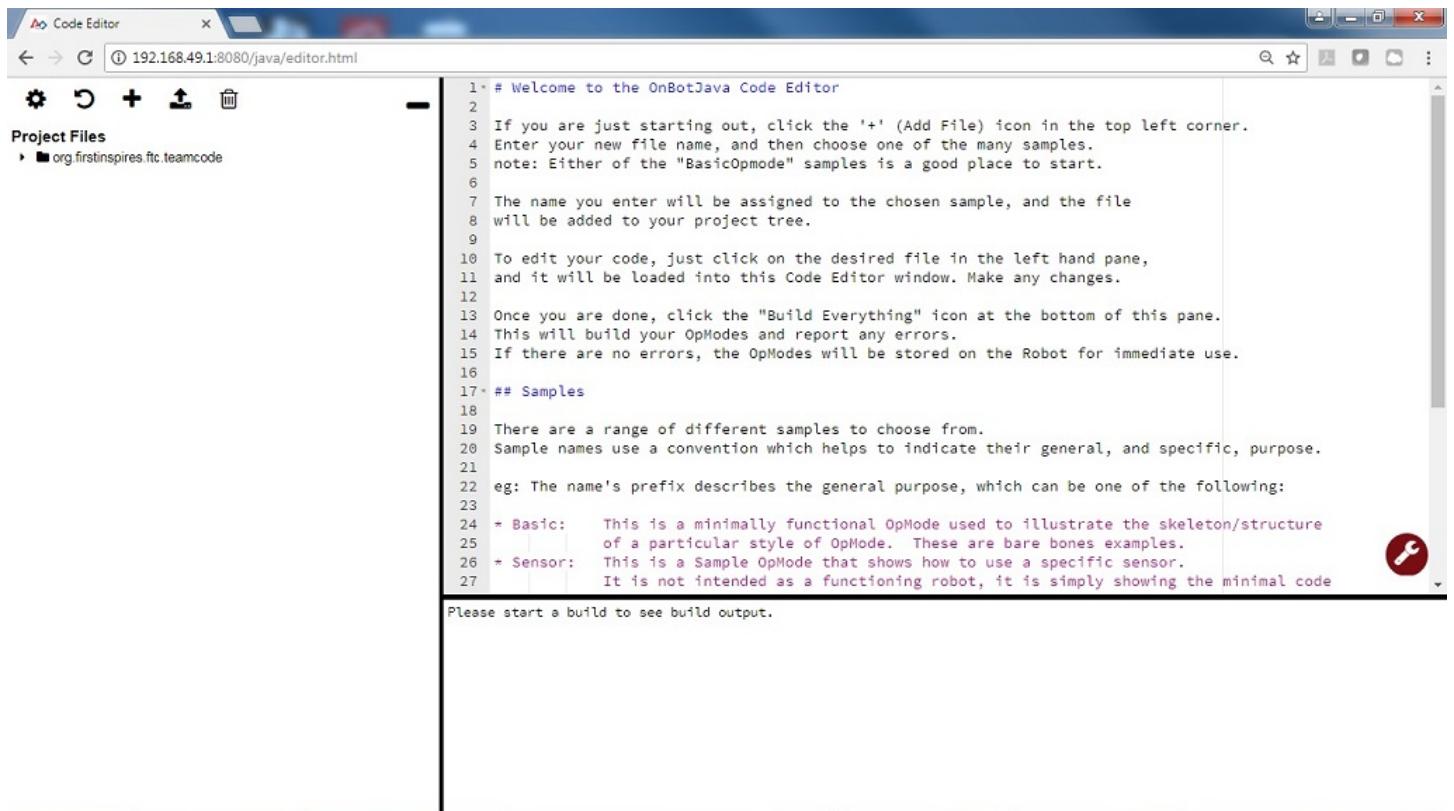
Robot controller status:  
**Server OK (Running since Dec 31, 7:00 PM)**

Active connections:  
Windows #1 connection.html

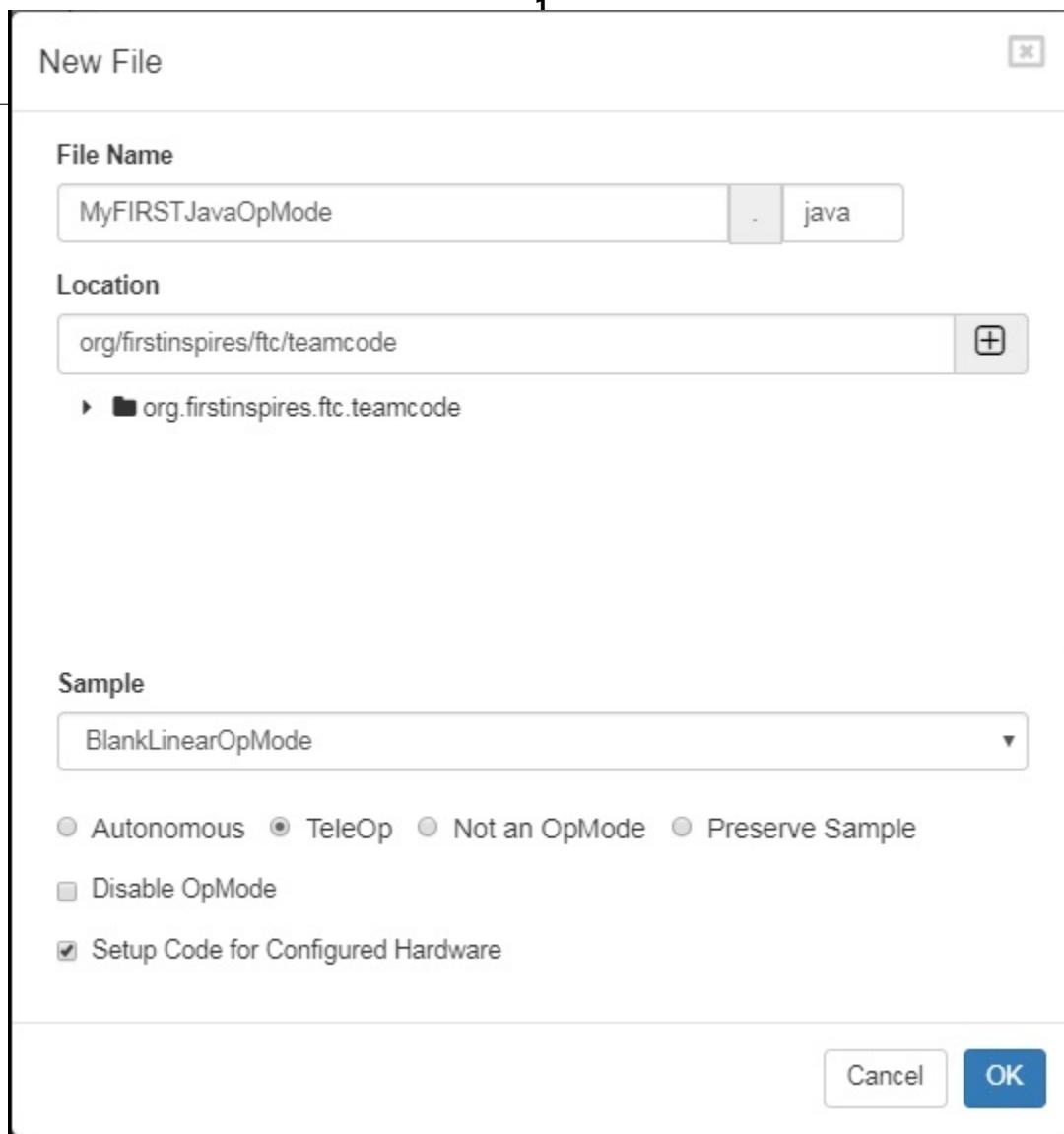
3. Click on the word OnBotJava towards the top of the screen. This will switch the browser to OnBot Java Programming mode.



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.

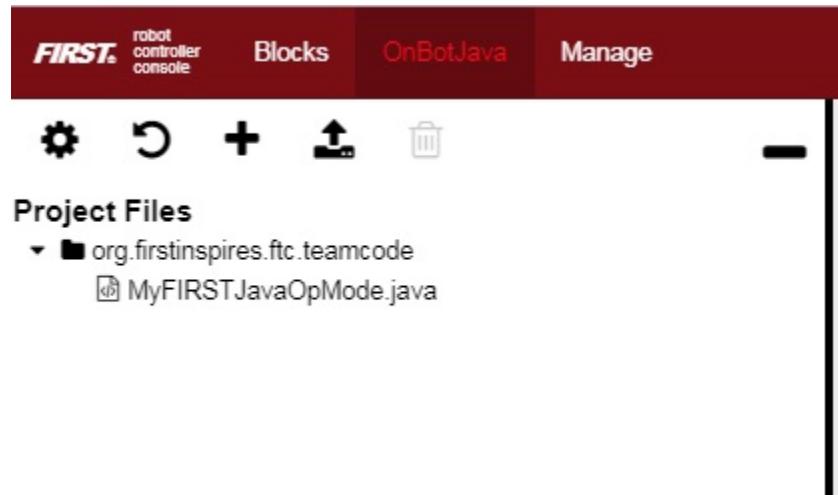
```

1 * /
2 Copyright 2017 FIRST Tech Challenge Team 9999
3
4 Permission is hereby granted, free of charge, to any person obtaining a copy of this software and
5 associated documentation files (the "Software"), to deal in the Software without restriction,
6 including without limitation the rights to use, copy, modify, merge, publish, distribute,
7 sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is
8 furnished to do so, subject to the following conditions:
9
10 The above copyright notice and this permission notice shall be included in all copies or substantial
11 portions of the Software.
12
13 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT
14 NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
15 NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
16 DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
17 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
18 */
19 package org.firstinspires.ftc.teamcode;
20
21 import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
22 import com.qualcomm.robotcore.hardware.Gyroscope;
23 import com.qualcomm.robotcore.hardware.DigitalChannel;
24 import com.qualcomm.robotcore.hardware.DistanceSensor;
25 import com.qualcomm.robotcore.hardware.Servo;
26 import com.qualcomm.robotcore.eventloop.opmode.TeleOp;
27

```



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):

```
@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:

```
@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.

```
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.

```
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.

```
@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:

FTC Programming Resources 222

```
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.

```
21 import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
22 import com.qualcomm.robotcore.hardware.Gyroscope;
23 import com.qualcomm.robotcore.hardware.DigitalChannel;
24 import com.qualcomm.robotcore.hardware.DistanceSensor;
25 import com.qualcomm.robotcore.hardware.Servo;
```

Build started at Wed Sep 06 2017 08:21:12 GMT-0400 (Eastern Daylight Time)

Build finished in 1.1 seconds  
Build succeeded!

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.

FTC Docs

FTC Programming Resources, 224

```
59
60
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/firstrinstspires/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/firstrinstspires/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/firstrinstspires/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/firstrinstspires/ftc/teamcode/MyFIRSTJavaOpMode.java line 51, column 13: ERROR: cannot find symbol
  symbol:   class Zervo
  location: class org.firstrinstspires.ftc.teamcode.MyFIRSTJavaOpMode
```

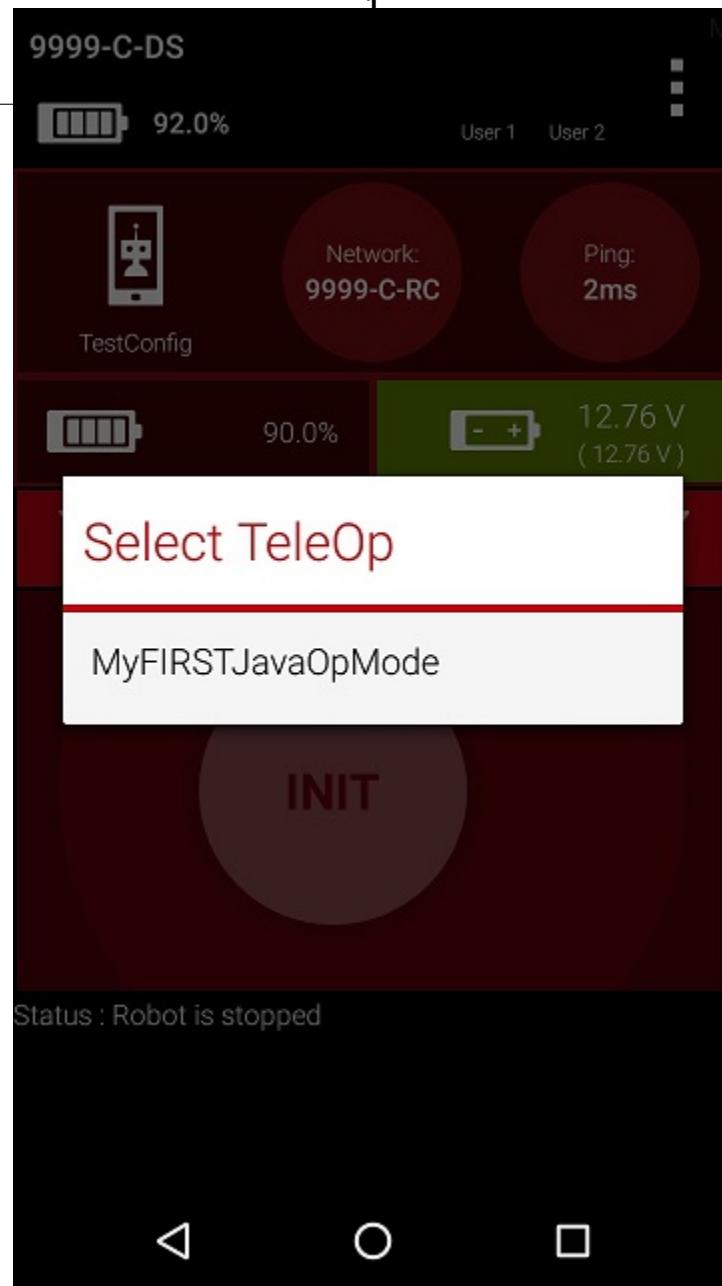
```
Build finished in 0.4 seconds
Build FAILED!
```

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.

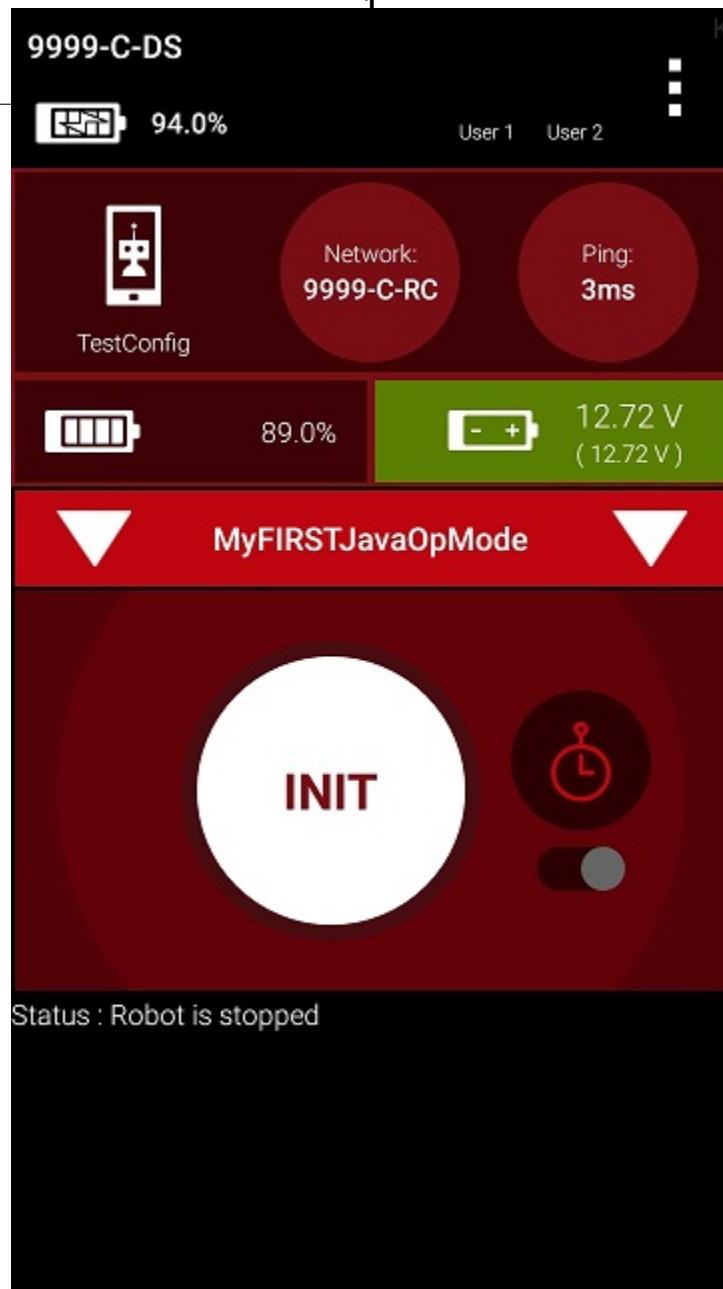
```
private Servo servoTest;
```

## 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.



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.



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.



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

### FTC Docs

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.

### FTC Programming Resources, 231

- Before you connect your gamepad to the phone, verify that the switch on the bottom of the gamepad is set to the "X" position.



Connect the gamepad to the DRIVER STATION using the Micro USB OTG adapter cable.



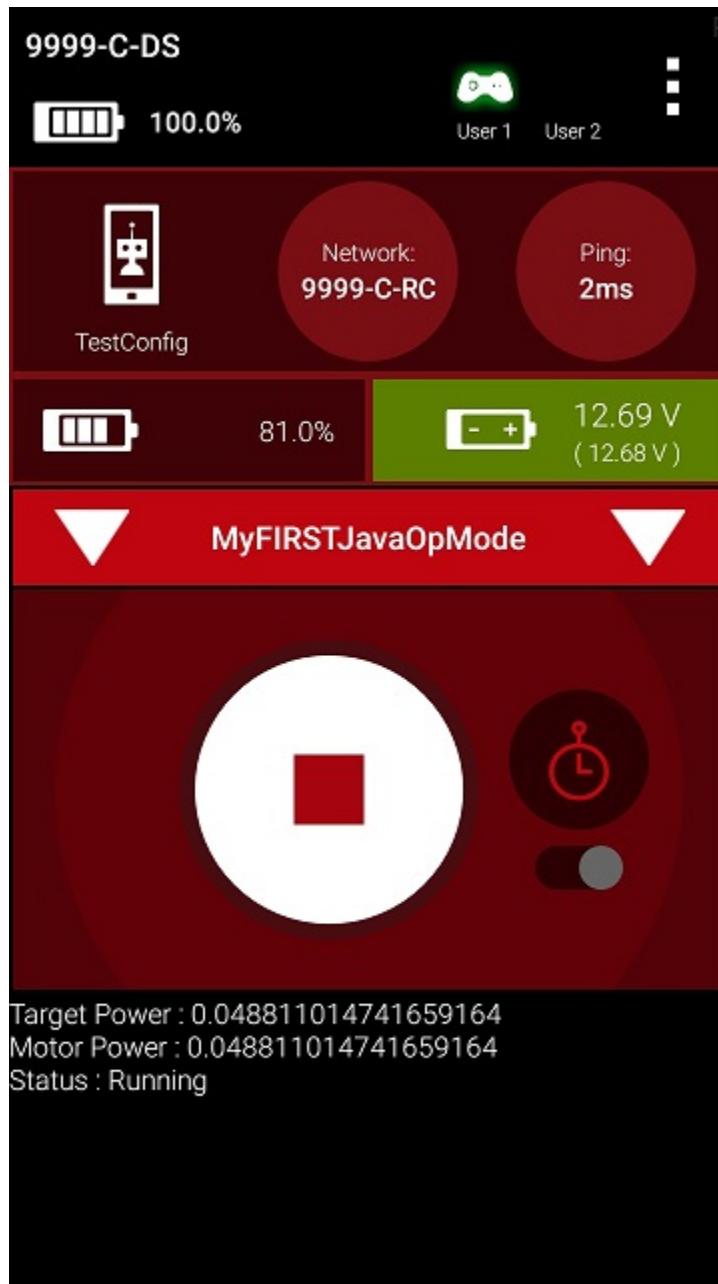
Your example op mode is looking for input from the gamepad designated as the user or driver #1. Press the Start button and the A button simultaneously on the Logitech F310 controller to designate your gamepad as user #1. Note that pushing the Start button and the B button simultaneously would designate the gamepad as user #2.



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.



## Controlling a Servo OBJ

[FTC Docs](#)

In this section, you will modify your op mode to control a servo motor with the buttons of the gamepad.

[FTC Programming Resources, 234](#)

### 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)

```
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 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 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 an 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 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”).

### 1.3.5 Reference Documentation OBJ

#### OnBot Java Reference Info OBJ

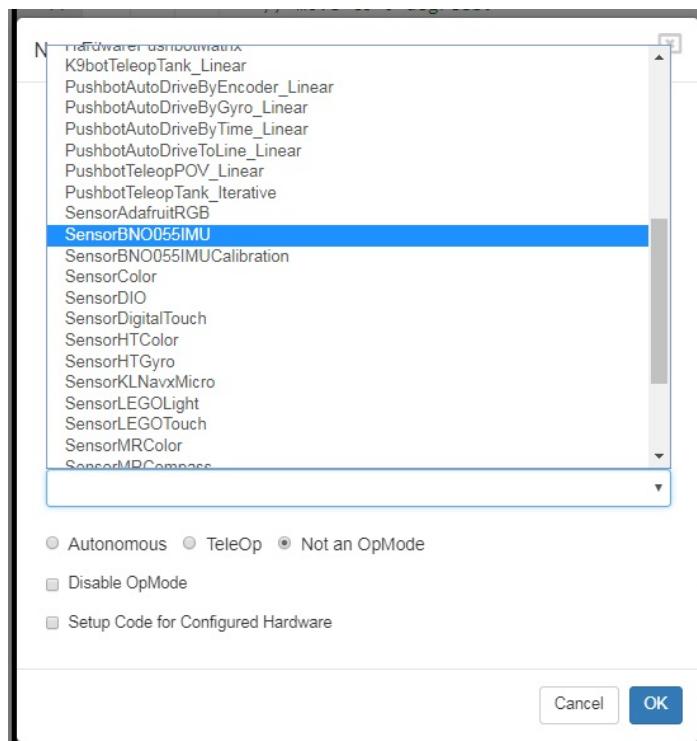
#### Javadoc Reference Pages

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:

- <http://ftc-community.firstinspires.org>

REV Robotics Expansion Hub Documentation

REV Robotics Expansion Hub Getting Started Guide

## 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

### 1.4.1 Introduction AS

## 1.4.2 Configuring your Hardware AS

### 1.4.3 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

FTC Docs

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

FTC Programming Resources 240

## Java Development Kit

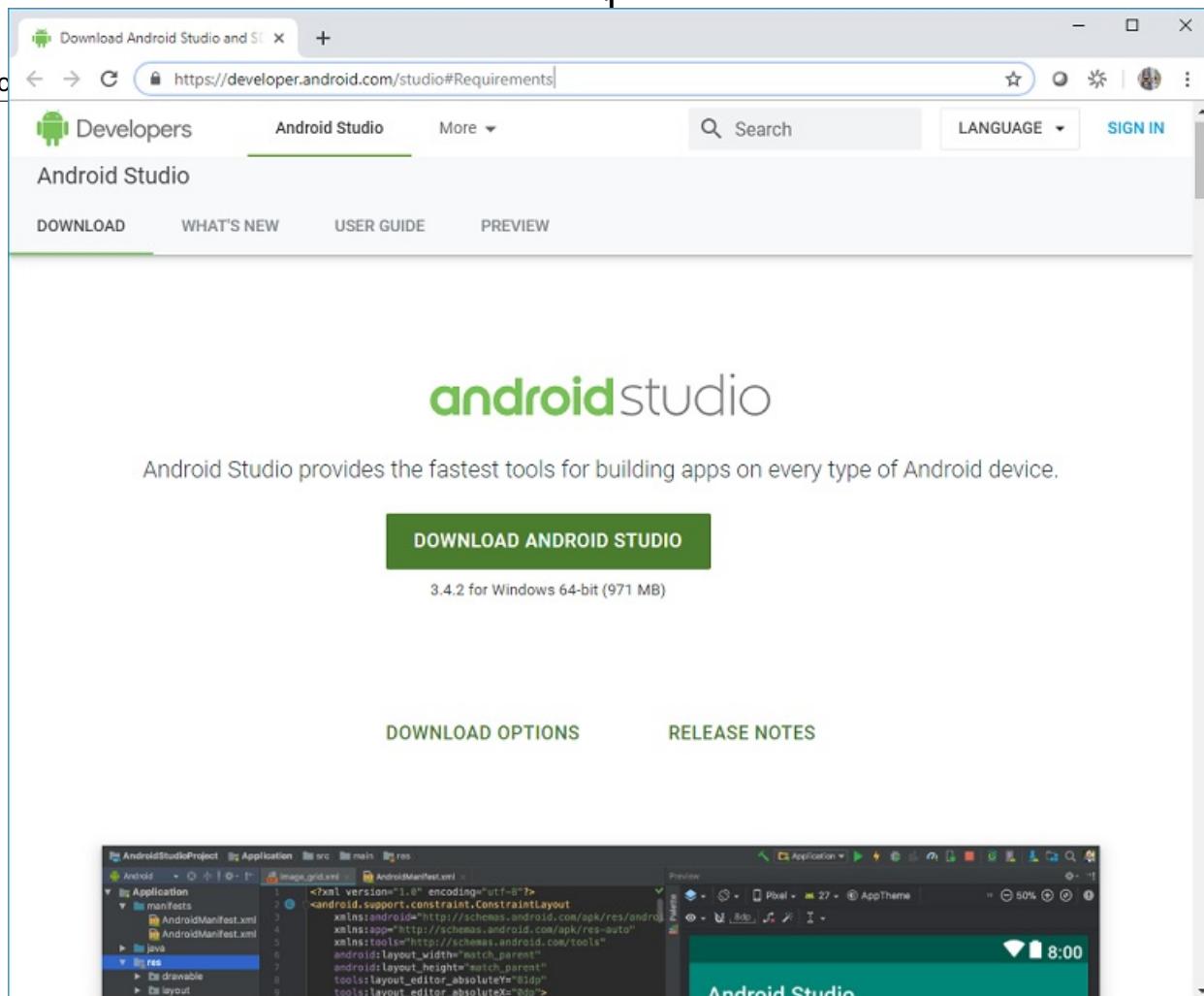
Earlier versions of Android Studio required that the user install the Java Development Kit software separately. Current versions of Android Studio incorporate the Java development software as part of the entire install package. It is no longer necessary (or recommended) to install the Java Development Kit separately. Instead, it is recommended that you use the Java Development Kit that is included with Android Studio.

## 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.

### 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.

<https://github.com/FIRST-Tech-Challenge/FtcRobotController>

**FIRST-Tech-Challenge / FtcRobotController**

Code Issues Pull requests Discussions Actions Projects Wiki ...

master ▾ Go to file Code ▾

CalKestis Merge pull request #7 from FIRST-Tech-Challenge/missing-g... on Sep 25 9

.github FtcRobotController v6.0 Click on the "Releases" tab to jump to the Releases page of the repository 3 months ago

FtcRobotController FtcRobotController v6.0 3 months ago

TeamCode FtcRobotController v6.0 3 months ago

doc FtcRobotController v6.0 3 months ago

gradle/wrapper FtcRobotController v6.0 3 months ago

libs FtcRobotController v6.0 3 months ago

.gitignore Update .gitignore 3 months ago

README.md FtcRobotController v6.0 3 months ago

build.common.gradle FtcRobotController v6.0 3 months ago

build.gradle FtcRobotController v6.0 3 months ago

gradle.properties FtcRobotController v6.0 3 months ago

gradlew Missing gradle wrappers 3 months ago

gradlew.bat Missing gradle wrappers 3 months ago

settings.gradle FtcRobotController v6.0 3 months ago

README.md

**NOTICE**

This repository contains the public FTC SDK for the Ultimate Goal (2020-2021)

About

No description, website, or topics provided.

Readme

Releases 1

v6.0 Latest on Sep 24

Packages

No packages published

Contributors 2

cmacfarl Craig MacFarlane

CalKestis Cal Kestis

Languages

Java 100.0%

From the main repository web page, click on the “releases” link to jump to the Releases page for the repository. The Releases

**Gracious Professionalism® - “Doing your best work while treating others with respect and kindness - It’s what makes FIRST, first.”**

page should list the available software releases for the repository. The latest release should be displayed near the top of the page.

The screenshot shows the GitHub releases page for the repository `FIRST-Tech-Challenge/FtcRobotController`. The page displays the `v6.0` release, which was released by `CalKestis` on Sep 24. The release notes mention that it includes 2 commits to the `master` branch since the previous release. The release is marked as `Verified`. The page also lists four assets: `FtcDriverStation-release.apk`, `FtcRobotController-release.apk`, `Source code (zip)`, and `Source code (tar.gz)`.

**Latest release**

`v6.0`  
-o- 7ab3861  
`Verified`

[Compare ▾](#)

## v6.0

CalKestis released this on Sep 24 · 2 commits to master since this release

This is the official release for the 2020/2021 season.

For a list of what's new please see the README.

For details on how to use the FTC Android control system, please visit the online wiki:

<https://github.com/FIRST-Tech-Challenge/FtcRobotController/wiki>

The Javadoc reference info is online at the following URL:

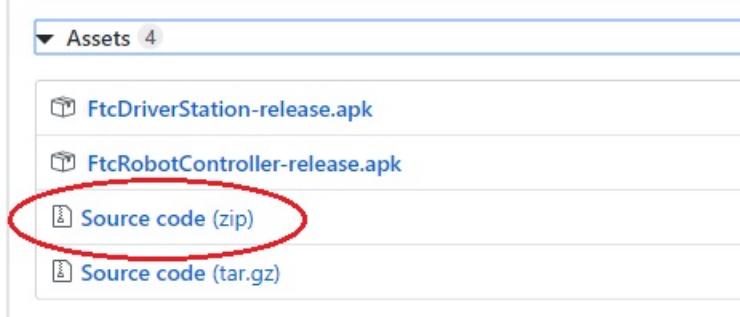
<https://first-tech-challenge.github.io/FtcRobotController/>

**Assets 4**

<a href="#">FtcDriverStation-release.apk</a>	34 MB
<a href="#">FtcRobotController-release.apk</a>	39.2 MB
<a href="#">Source code (zip)</a>	
<a href="#">Source code (tar.gz)</a>	

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.  
1  
FTC Docs

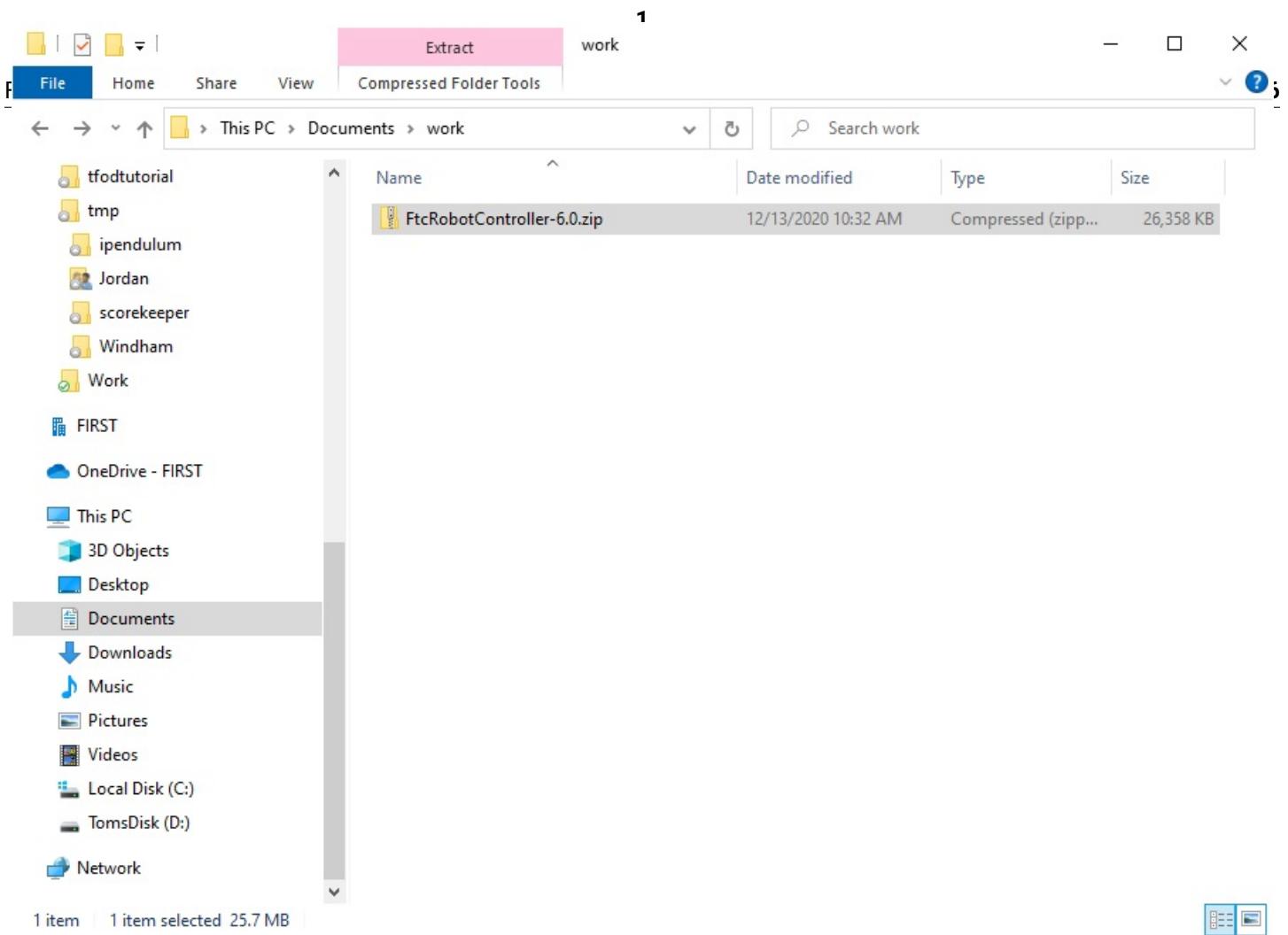
FTC Programming Resources, 245



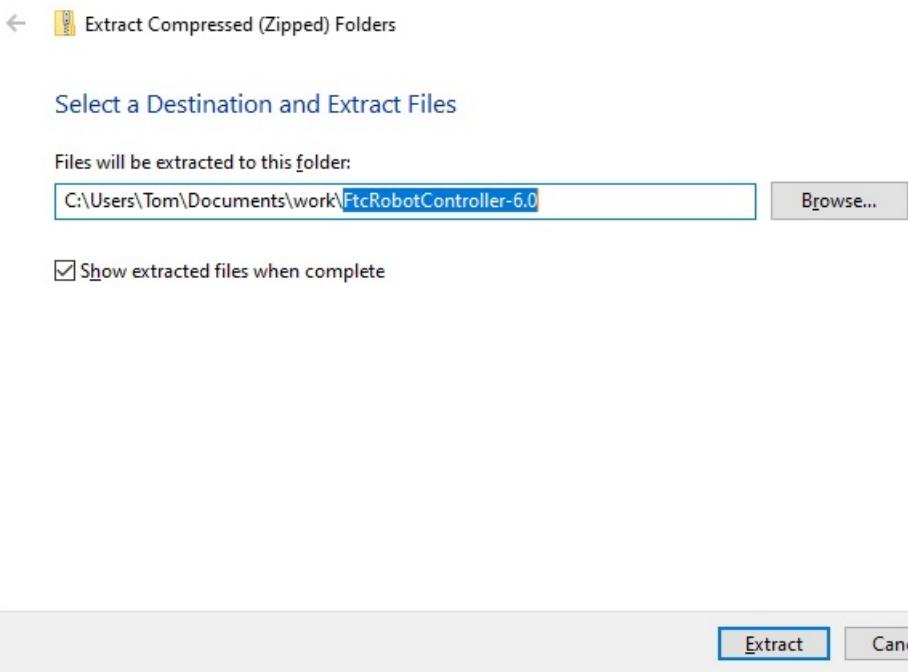
Click on the Source code (zip) link to download the compressed Android Studio project folder.

#### 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".

Files will be extracted to this folder:

C:\Users\Tom\Documents\work\FtcRobotController-6.0



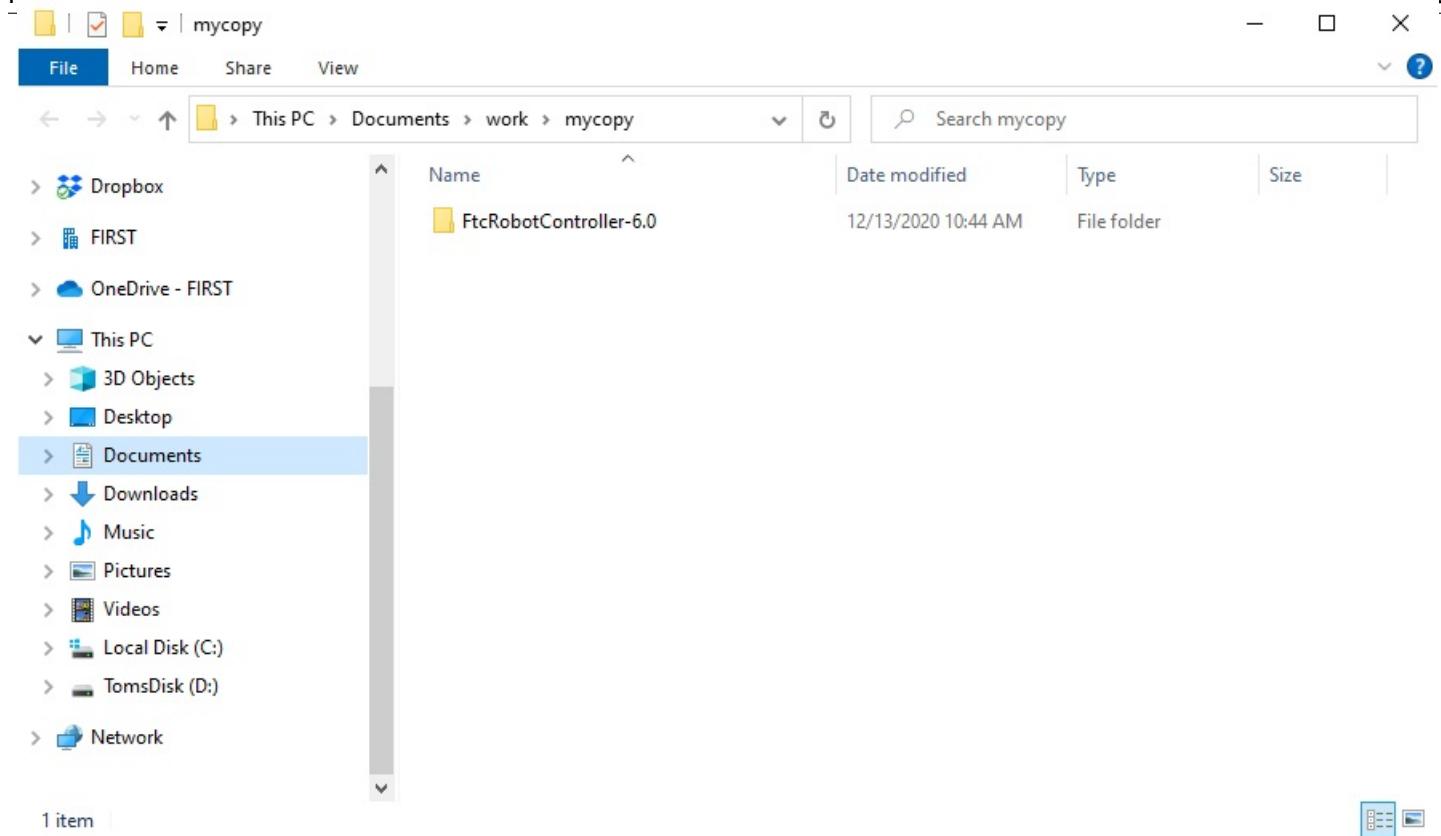
Rename destination folder...



Files will be extracted to this folder:

C:\Users\Tom\Documents\work\mycopy

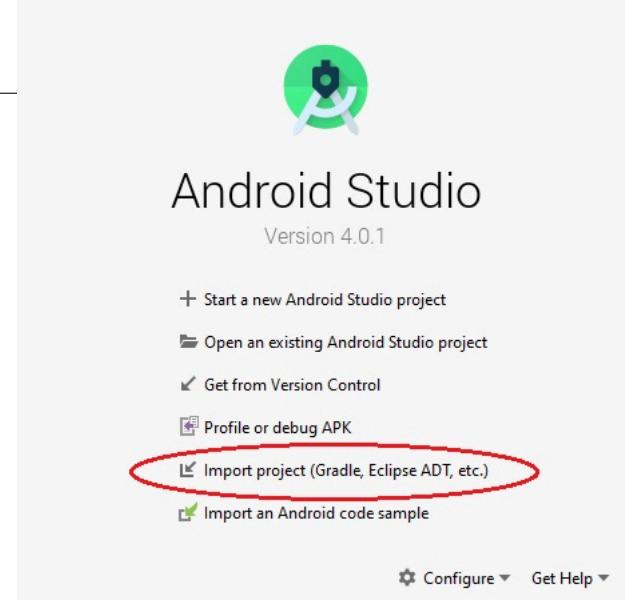
After you have renamed the destination folder, extract the contents of the archive to the folder. After the extraction process is complete, verify that the project folder was successfully extracted to its target destination.



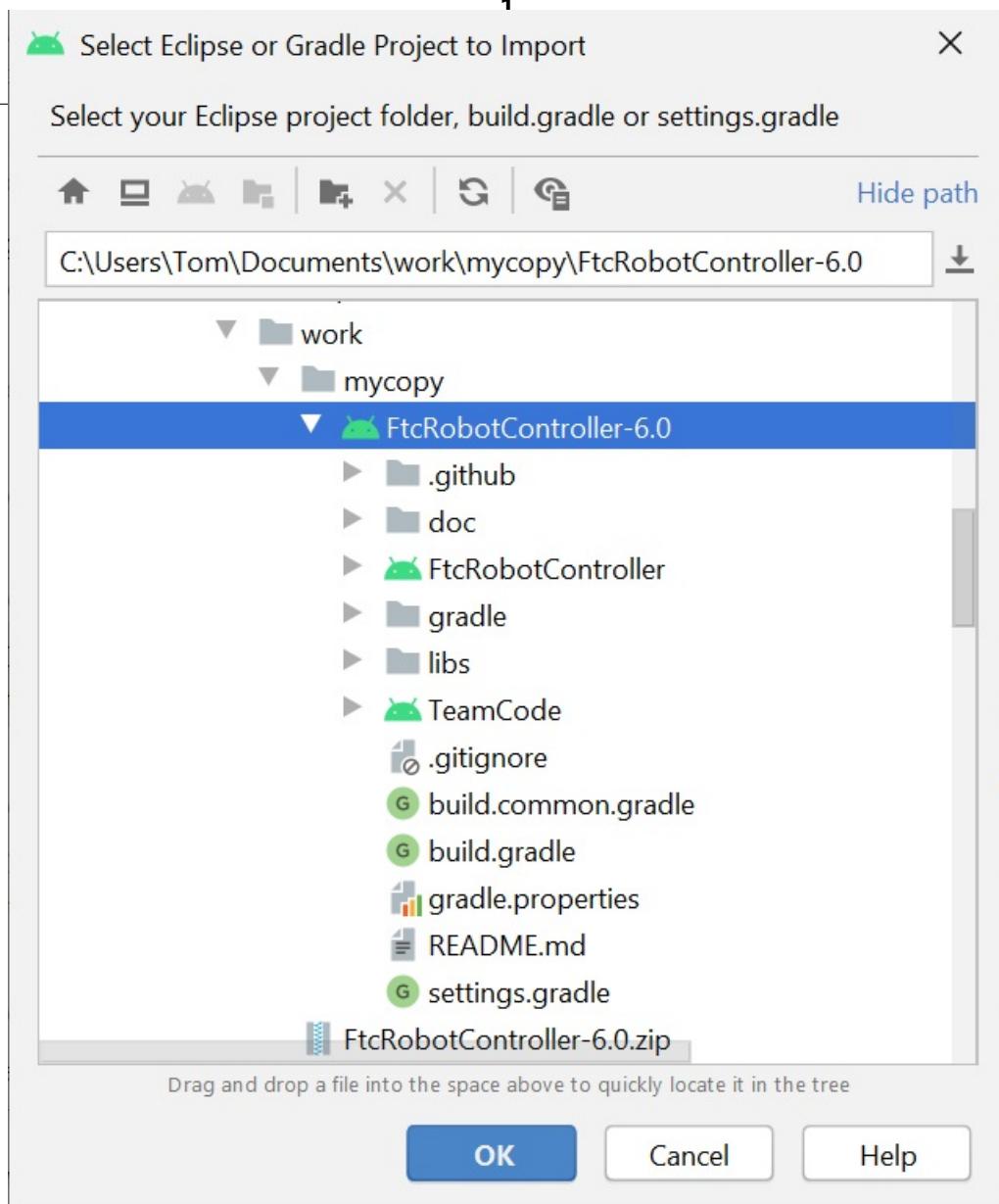
Once you have successfully extracted the contents of the archived file, you are ready to import the project into Android Studio.

### Importing the Project into Android Studio

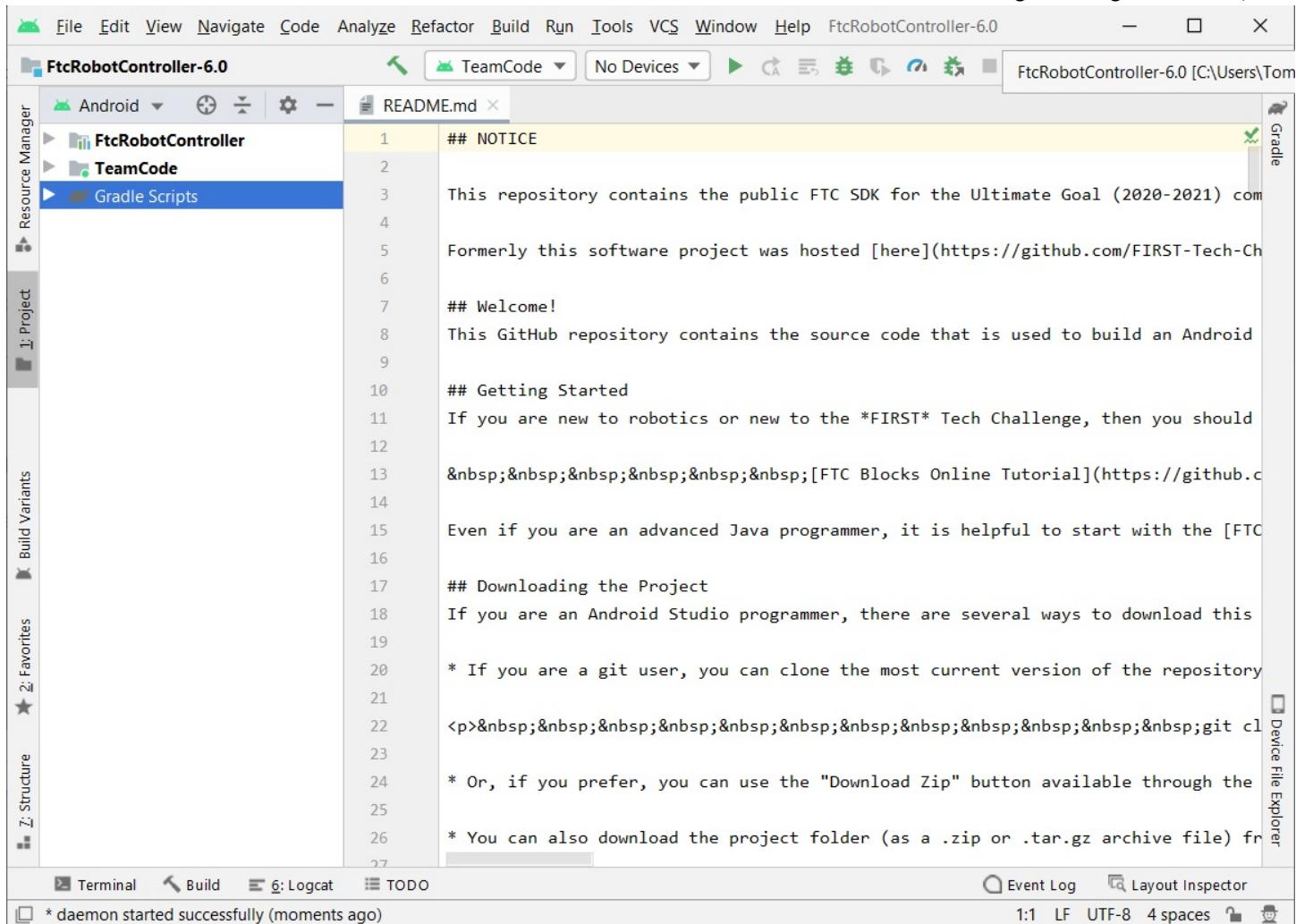
In order to import the Project, you will need to launch the Android Studio software on your computer. On the main Android Studio Welcome screen, select the option to “Import project (Gradle, Eclipse, ADT, etc.)” to begin the import process.



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.

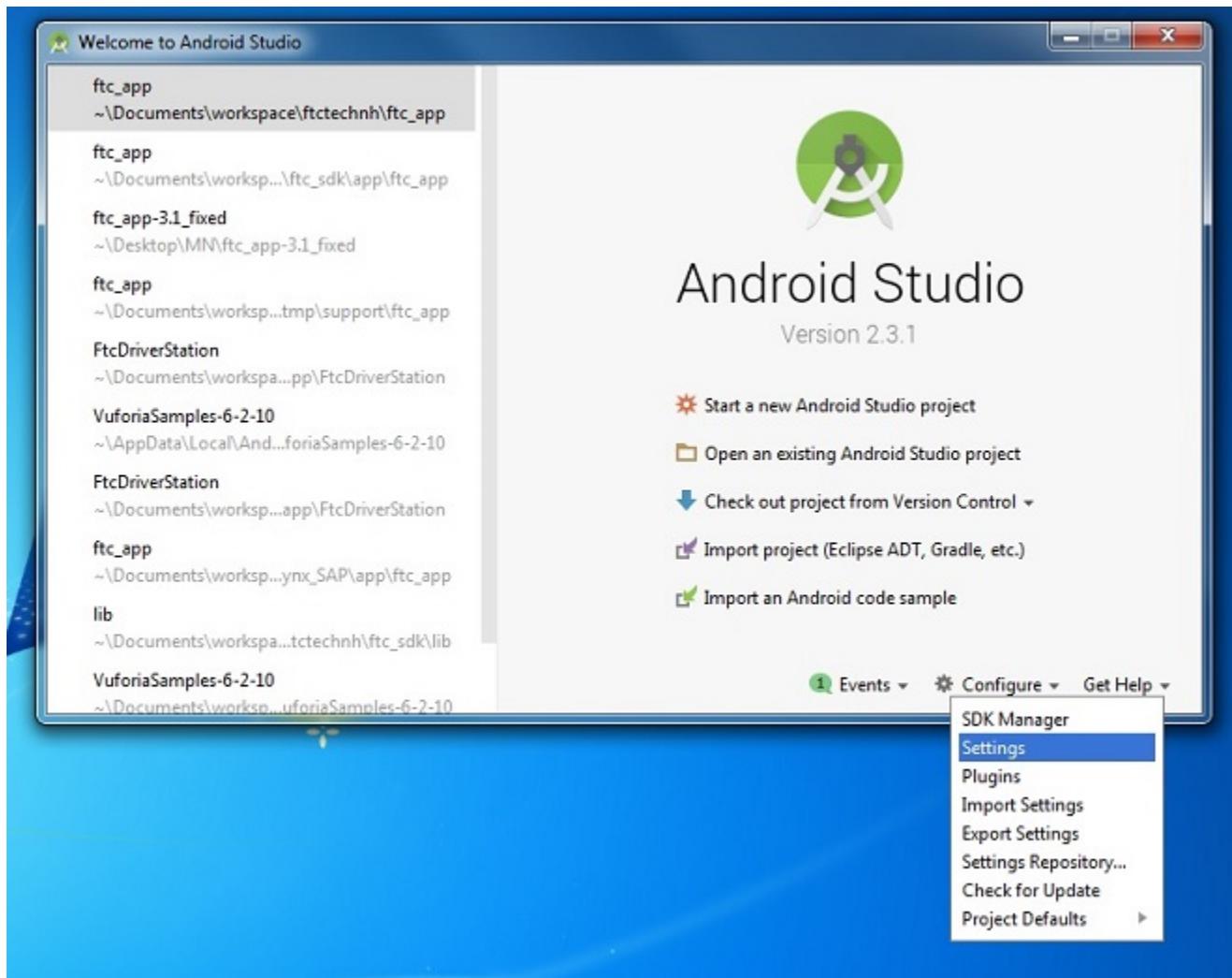
## Introduction

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.

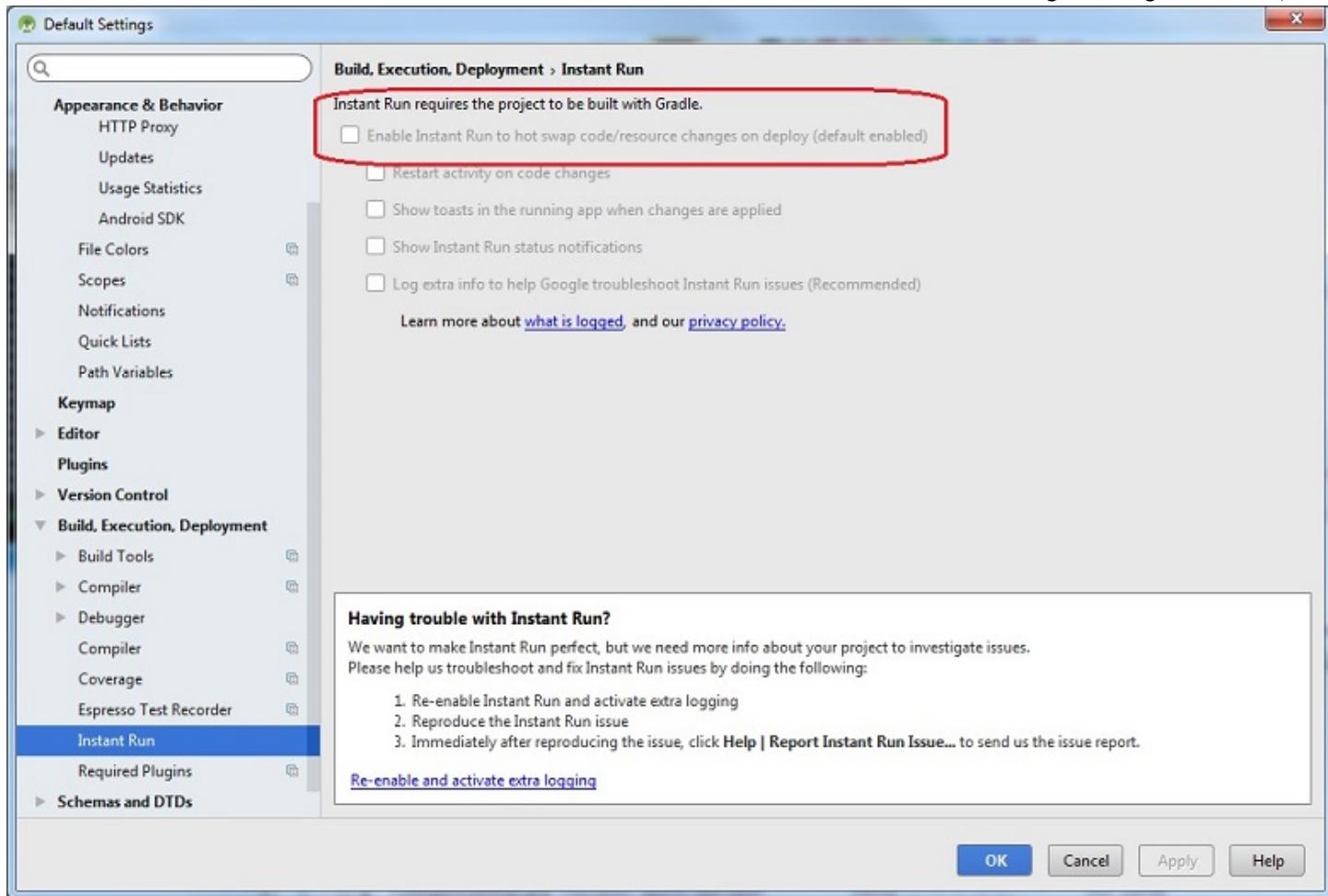
Teams who use Android Studio **must** disable Instant Run.

## 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>

### 1.4.4 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/FtcRobotController repository. Forks of the FIRST-Tech-Challenge/FtcRobotController 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.

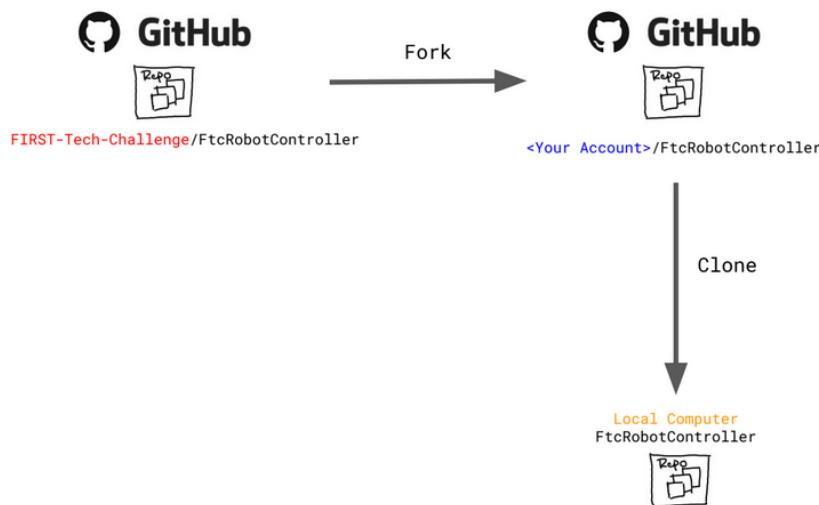


Fig. 18: The relationship between forks and clones. The clone exists on your local laptop while the fork exists on GitHub servers.

## 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'.

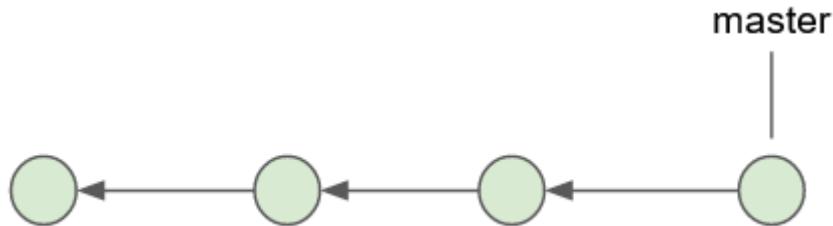


Fig. 19: A single branch with the default name of master

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.

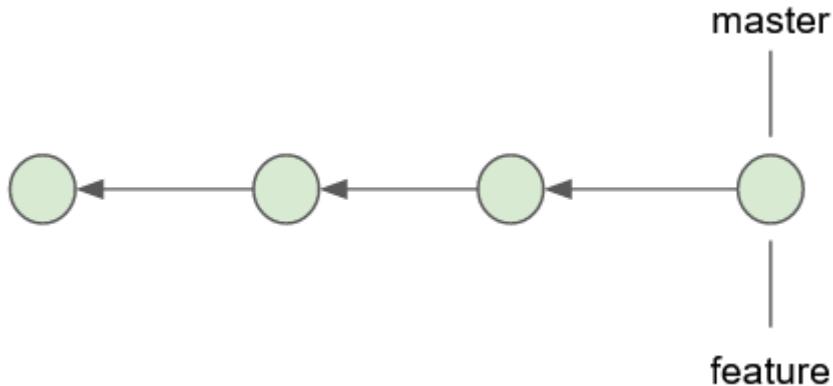


Fig. 20: Two branches that point to the same commit.

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.

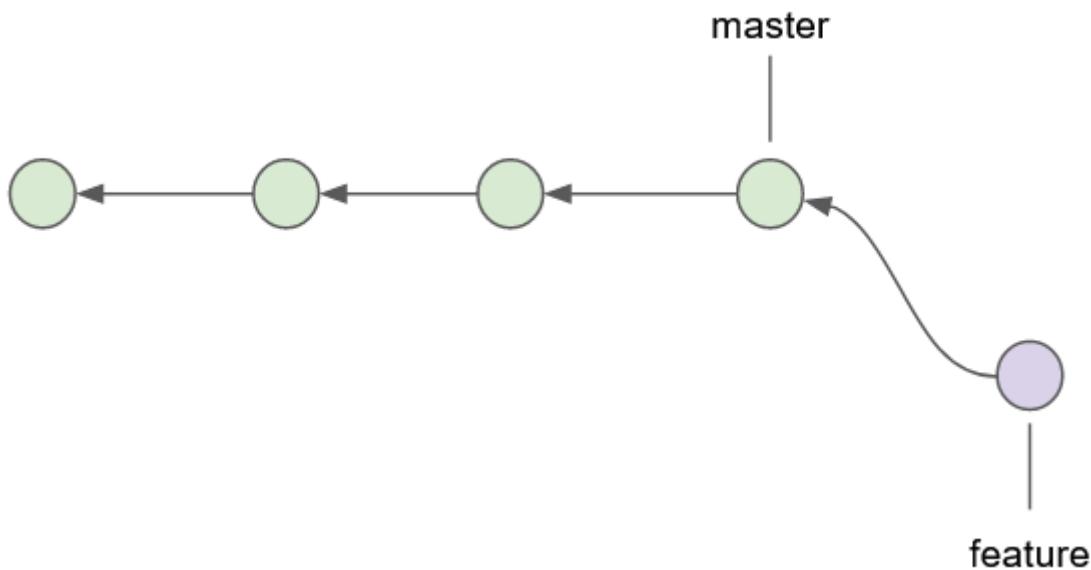


Fig. 21: New commit on the feature branch.

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.

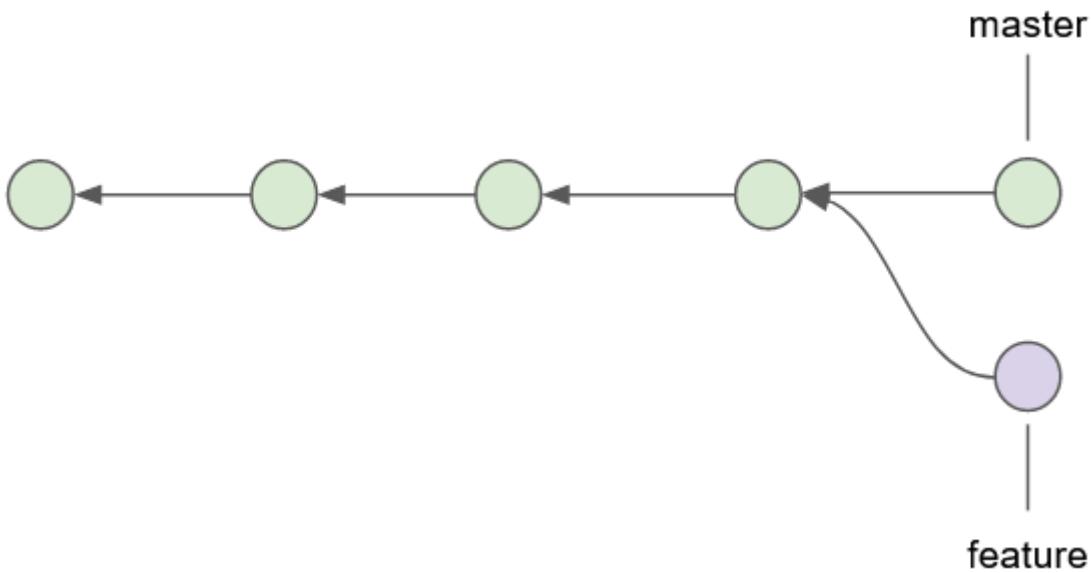


Fig. 22: Two independent lines of development.

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

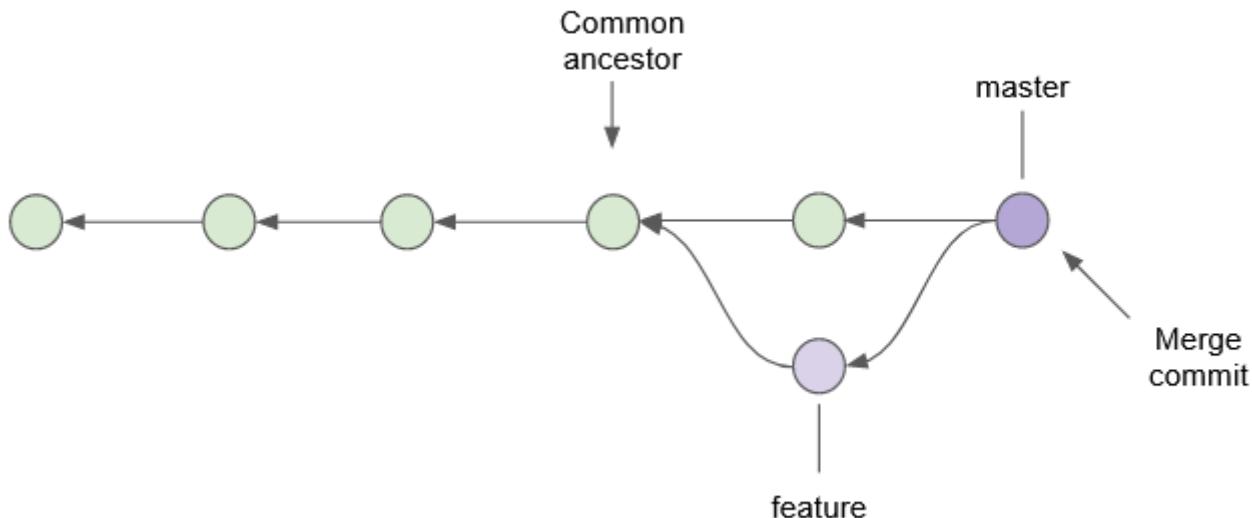


Fig. 23: Merging the feature branch back into the master branch.

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](#)

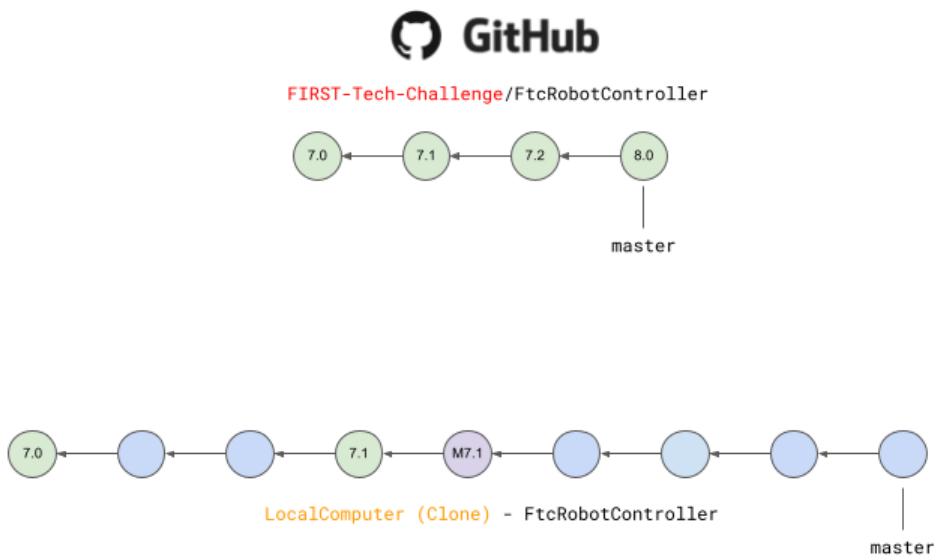


Fig. 24: FIRST-Tech-Challenge/FtcRobotController master vs. typical team repository master.

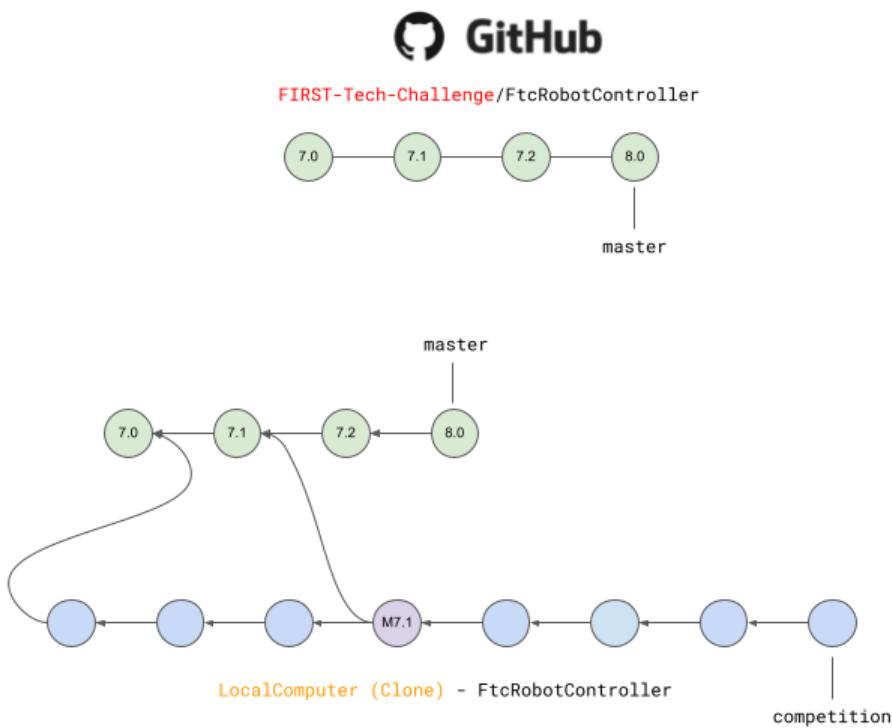


Fig. 25: Team repository's master always matches FIRST-Tech-Challenge/FtcRobotController's master branch.

## 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.

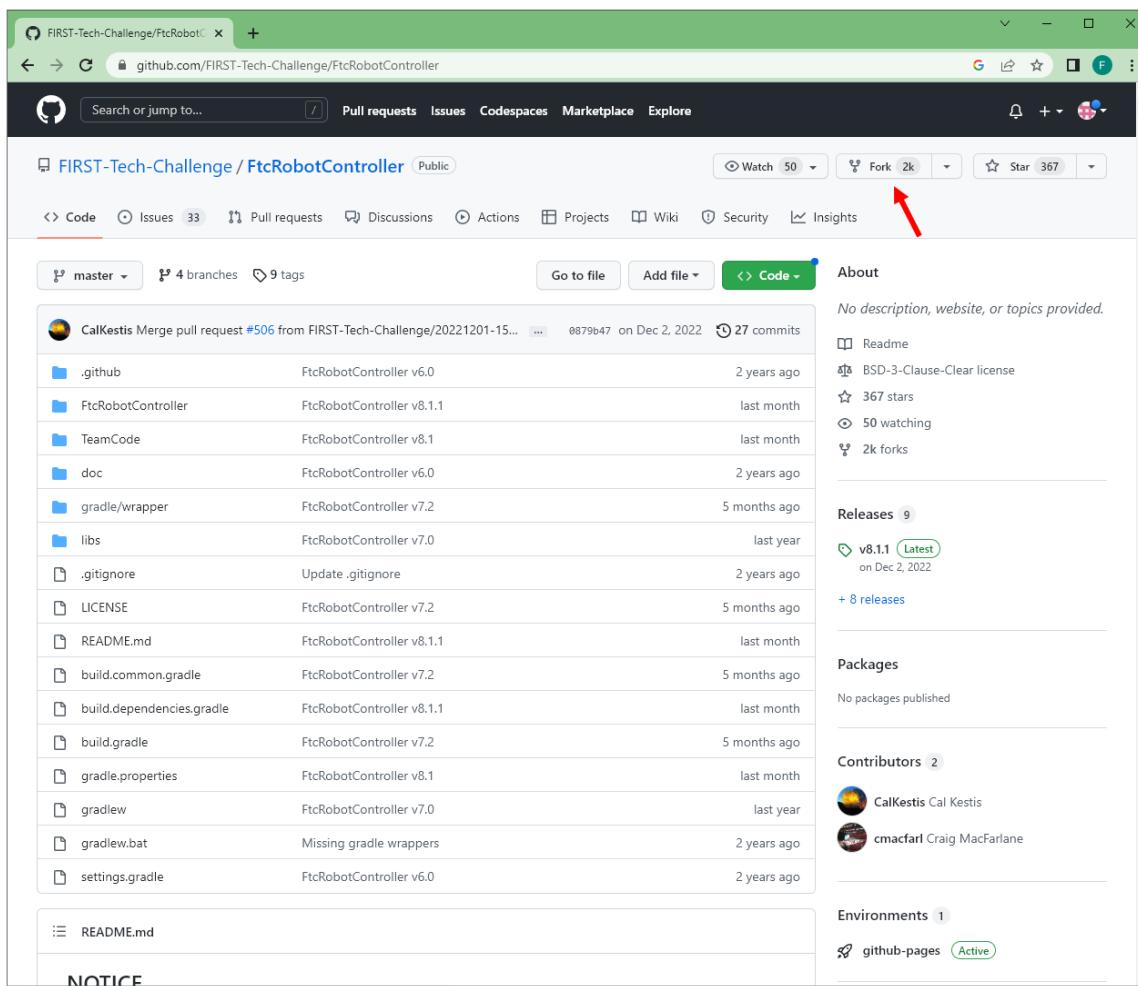


Fig. 26: Forking a GitHub repository.

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](https://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.

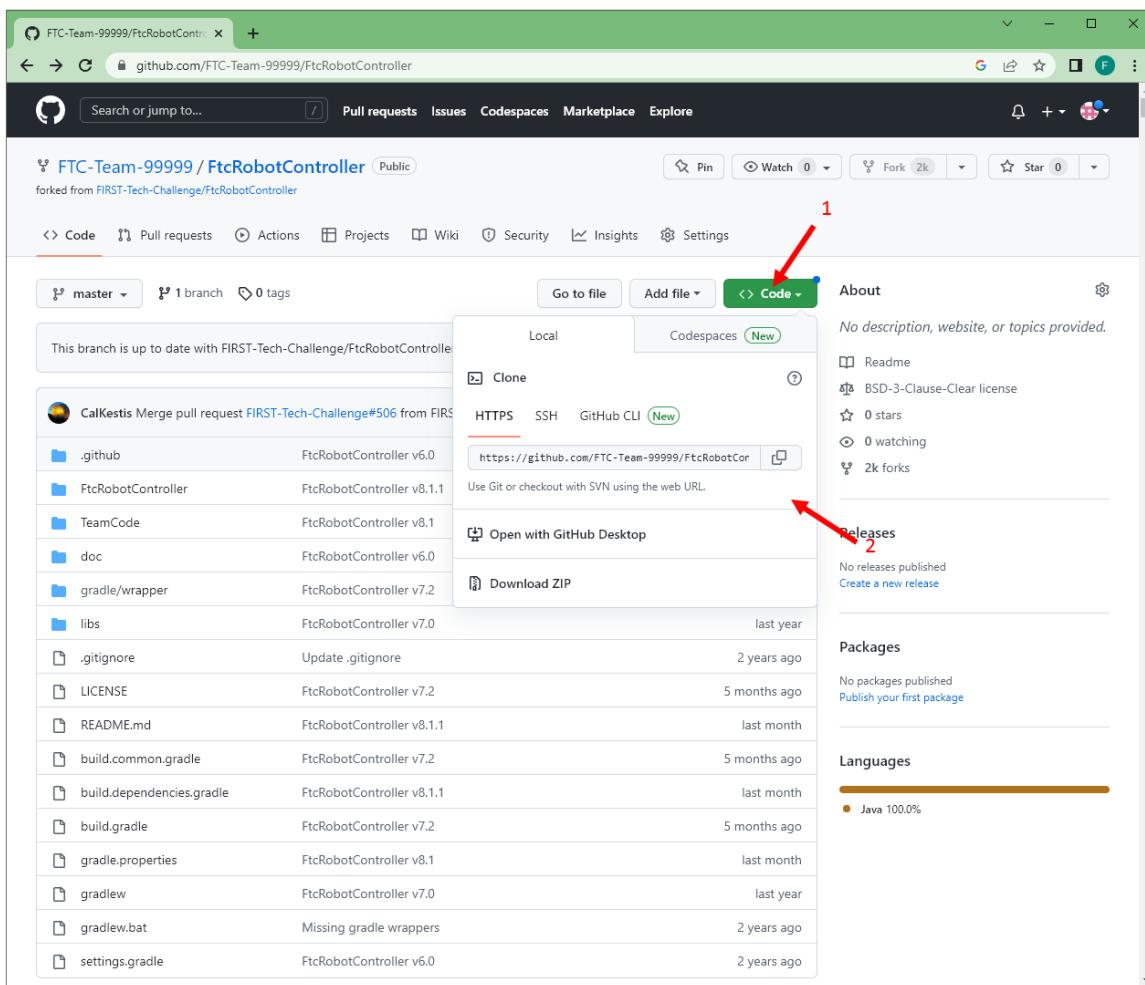


Fig. 27: Cloning a forked repository.

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

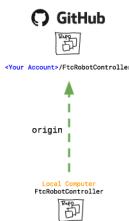


Fig. 28: Illustration of FtcRobotController as remote named *origin*.

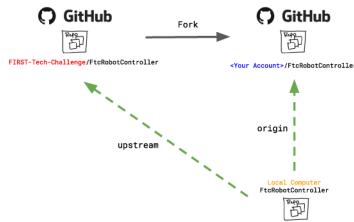


Fig. 29: A local repository with two remotes.

```
$ 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. 30: 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.

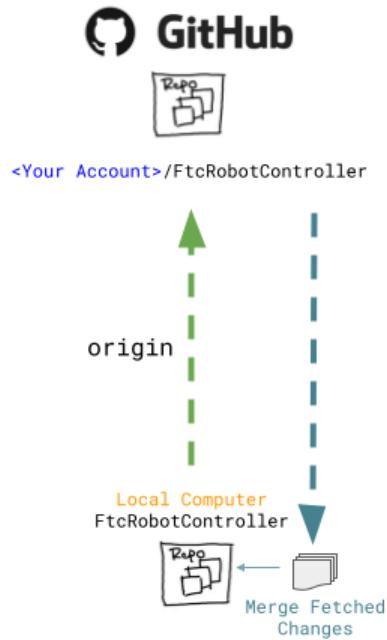


Fig. 31: Merging fetched changes from the origin repository.

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, 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.

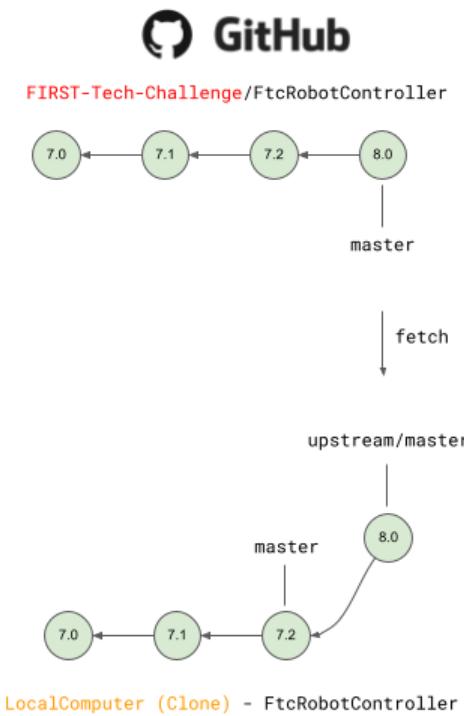
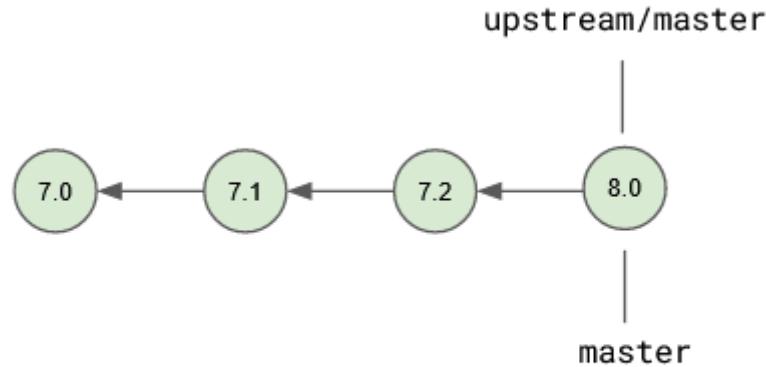


Fig. 32: Fetching changes from the upstream repository.

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



### LocalComputer (Clone) - FtcRobotController

Fig. 33: 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>
$ git 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 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.

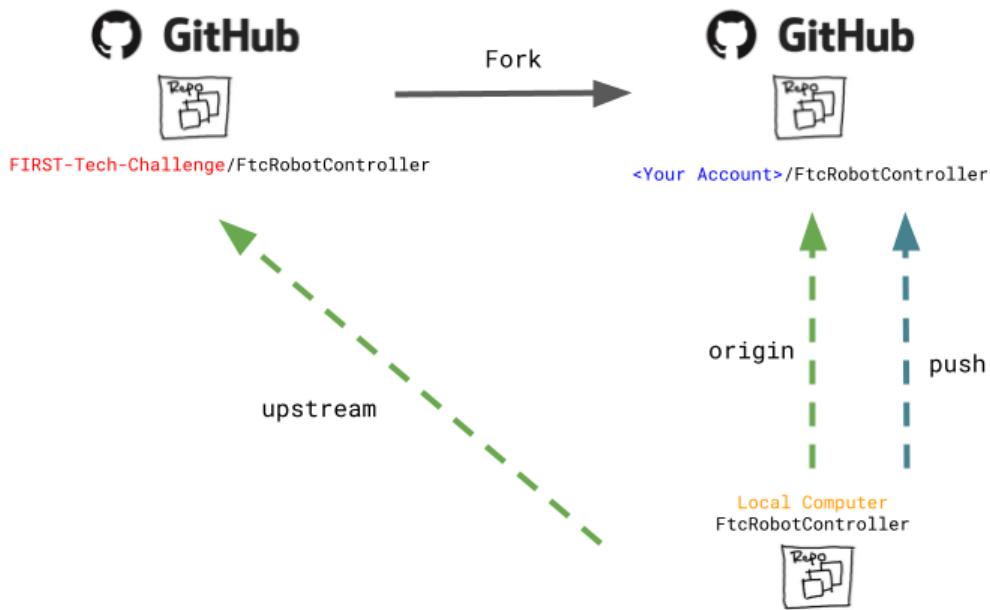


Fig. 34: Pushing fetched and merged changes back to your team fork.

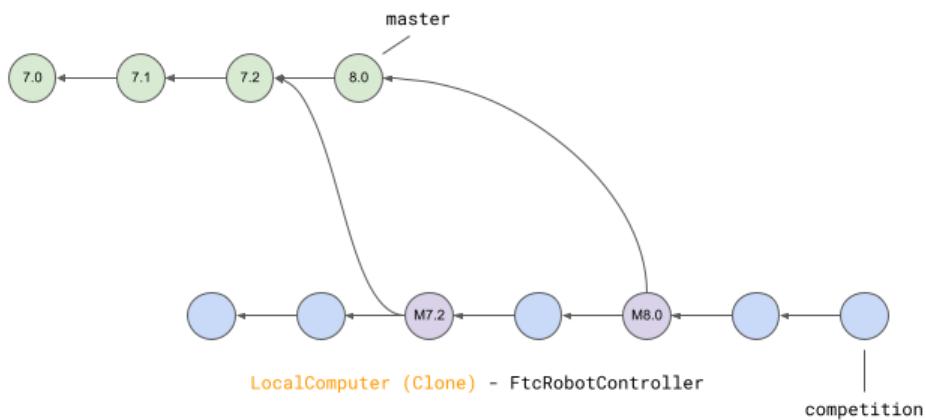


Fig. 35: A repository with both team commits and SDK update commits.

## 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.

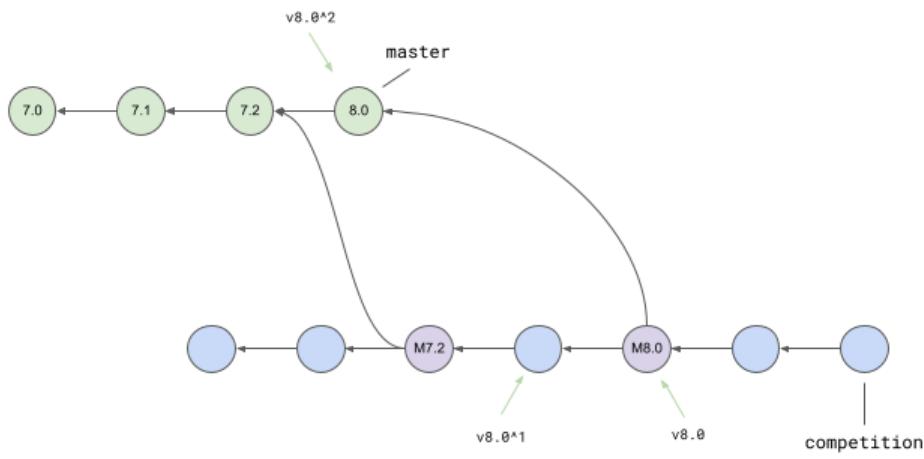
```
$ git fetch --all --tags
```

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.



LocalComputer (Clone) - FtcRobotController

Fig. 36: 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.

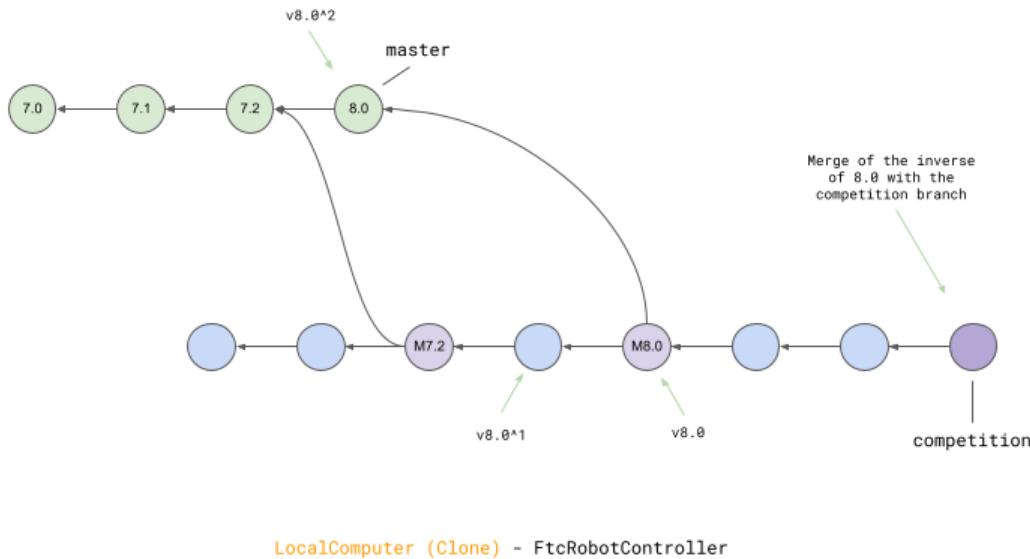


Fig. 37: Result of revert - a new merge commit representing the revert from v8.0 to v7.2.

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

**FTC Docs** 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.

**FTC Programming Resources 269**

### 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
```

## 1.4.5 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.

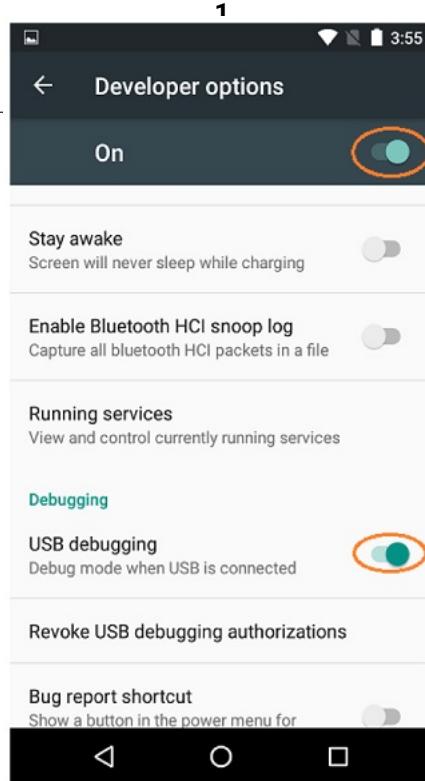
---

**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.

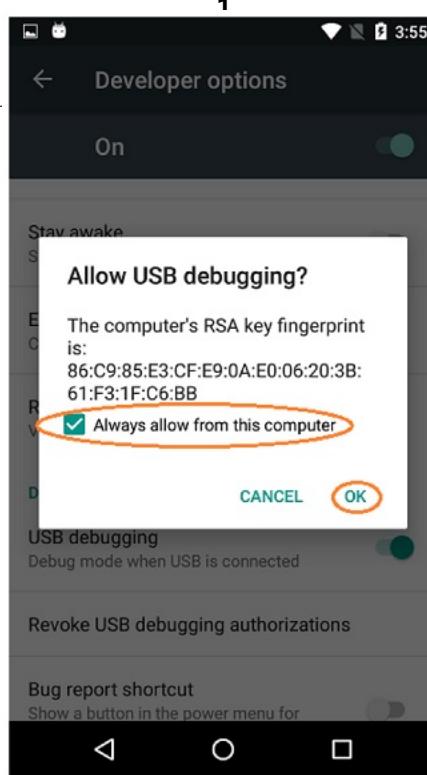
The Android Developer website contains information on how to enable Developer Options onto your phone. If you visit the following link and read the section entitled "Enabling On-device Developer Options" you will see that you can enable Developer Options on your Android phone by going to Settings->About 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.



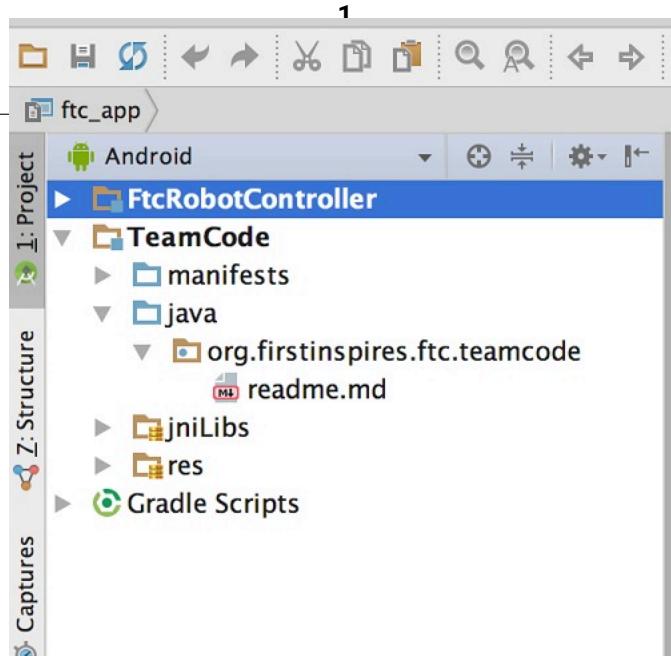
When you first connect a phone to your computer with Android Studio running, the phone might prompt you if it is OK to allow the computer to have USB debugging access to the phone. If this happens, make sure that you check the “Always allow from this computer” option and hit the OK button to allow USB debugging.



## Creating and Running an Op Mode 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 op modes that you will write to control your competition robot.



When you create your classes and op modes, you will want 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

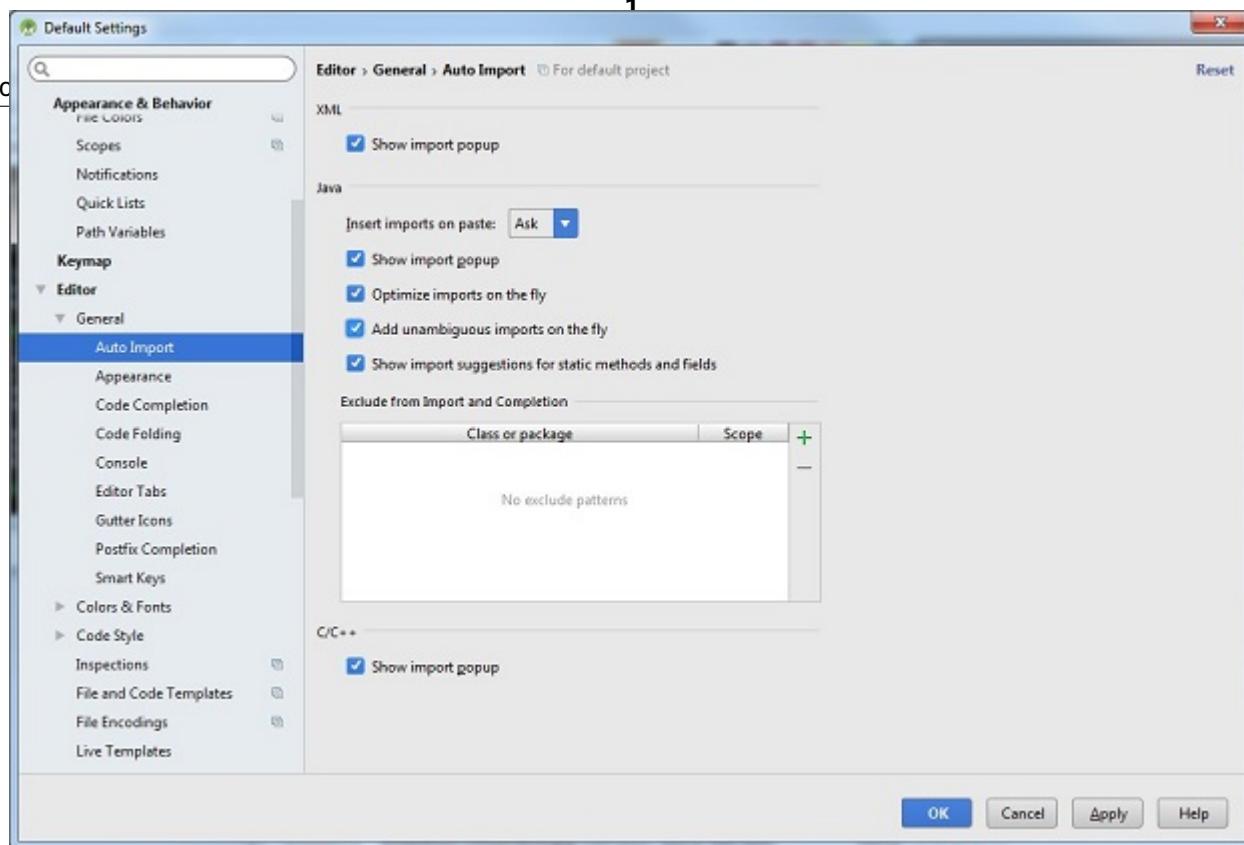
The Javadoc reference documentation for the SDK is available online. Visit the following URL to view the SDK documentation:

- <https://javadoc.io/doc/org.firstinspires.ftc>

### Enabling Auto Import

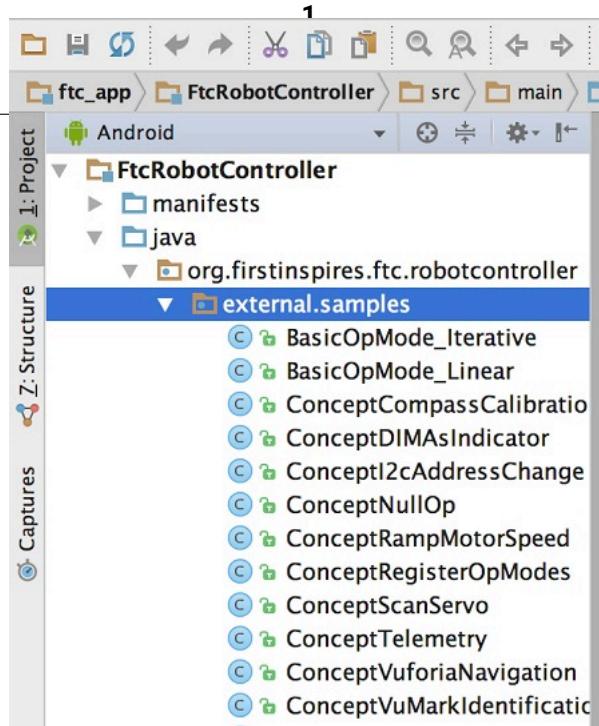
The auto import feature of Android Studio is a convenient function that helps save time as you write your op mode. If you would like to enable this feature, select the Editor->General->Auto Import item from the Android Studio Settings screen. This will display the editor's auto import settings.

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 op mode.



## Sample Op Modes

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 op mode, 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 op mode, look for the following annotation,

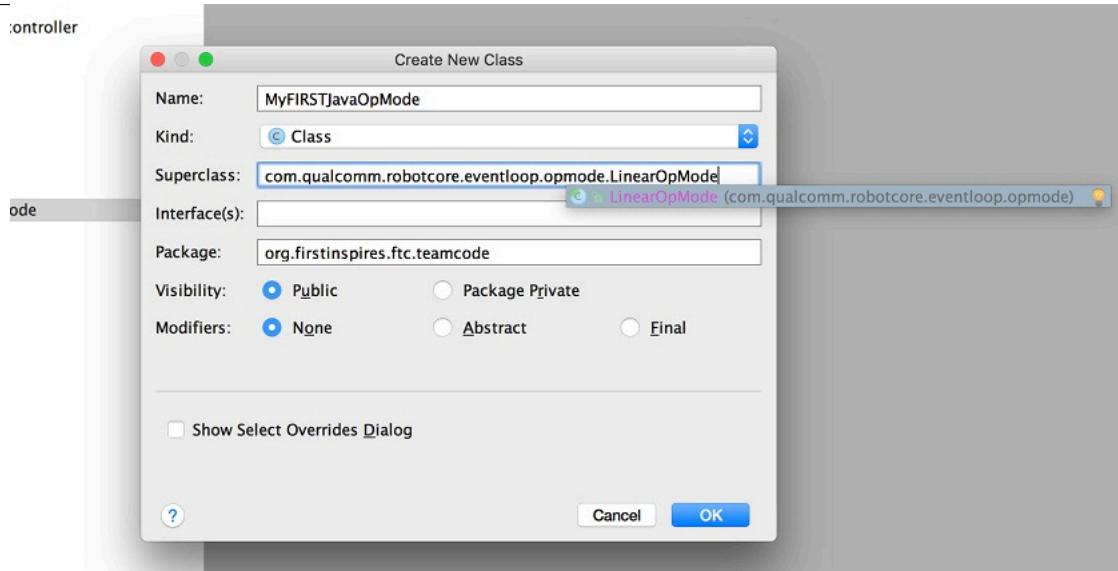
`@Disabled`

and comment out this line to enable the op mode and allow it to be run on the Robot Controller:

`//@Disabled`

### Creating Your FIRST Op Mode

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.

```

MyFIRSTJavaOpMode.java

MyFIRSTJavaOpMode
package org.firstinspires.ftc.teamcode;
import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
/**
 * Created by tom on 9/19/17.
 */
public class MyFIRSTJavaOpMode extends LinearOpMode {
}

```

Modify the main portion of your op mode 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)

```

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 op mode. Note that Android Studio automatically saves your source code as you are editing it.

Congratulations! You've written an op mode. It does not do much, but we will modify it to make it more useful.

### 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):

```
@TeleOp  
  
public class MyFIRSTJavaOpMode extends LinearOpMode {  
    private Gyroscope imu;  
    private DcMotor motorTest;
```

(continues on next 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 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:

@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.

```
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.

```
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.

```
@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();
}
```

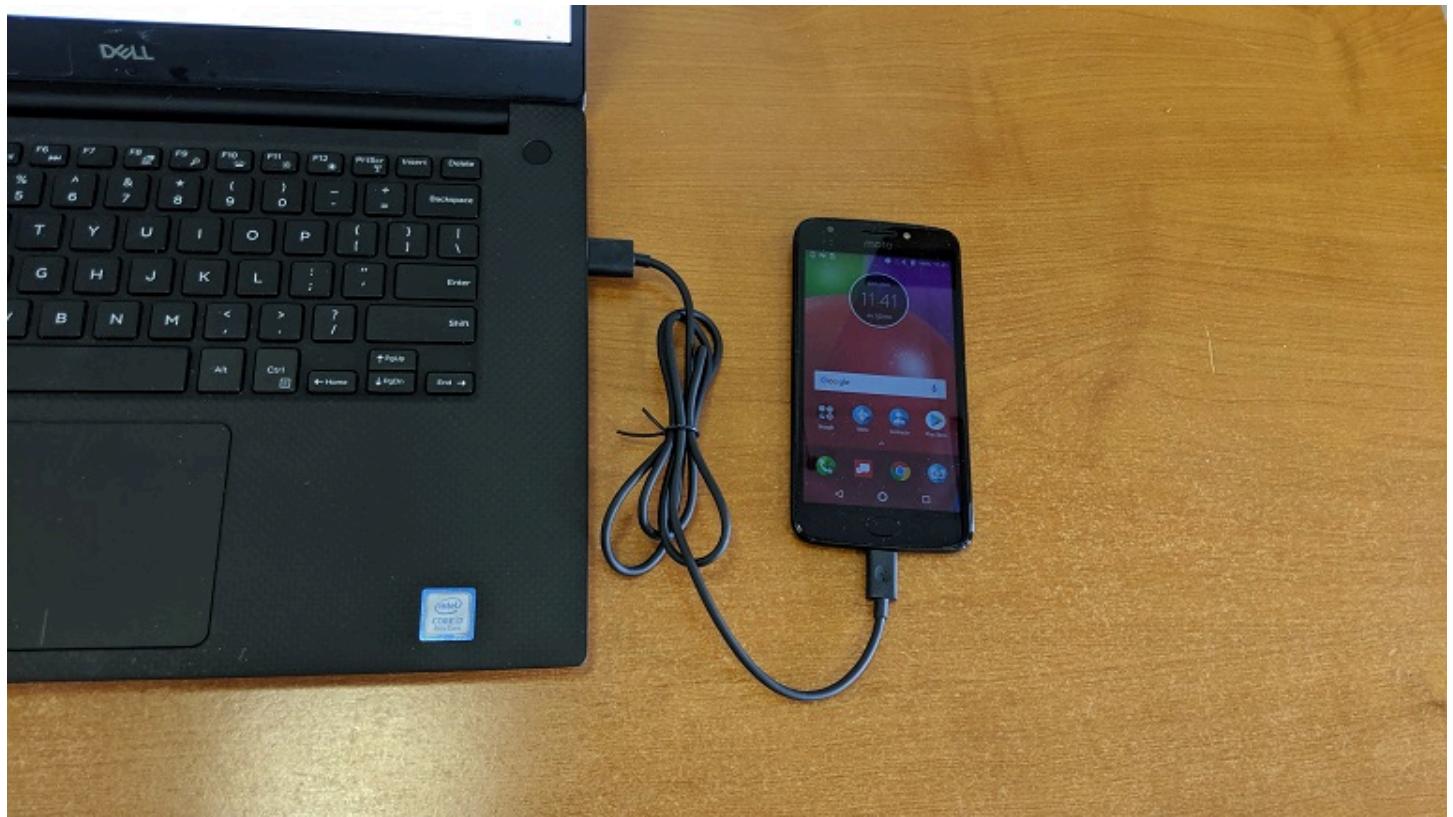
As the op mode iterates in the while loop, it will continue to send telemetry messages with the index of "Status" and the message of "Running" to be displayed on the Driver Station.

## Building and Installing Your Op Mode

FTC Docs

Verify that the Robot Controller phone is connected to your laptop and that the laptop has USB debugging permission for the phone.

FTC Programming Resources 280



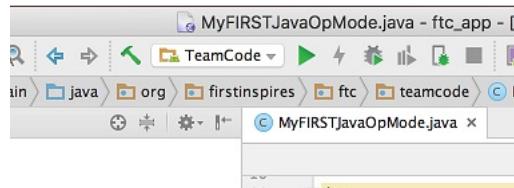
Or, if you are using a Control Hub, verify that the Control Hub is powered by a freshly charged 12V battery, and that it is connected to your laptop through its USB Type C port. Note that the Control Hub should automatically have USB debugging permission enabled.



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.

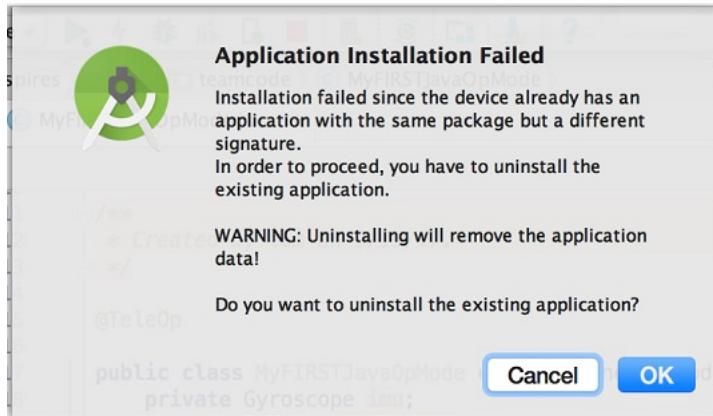


Android Studio should prompt you to select a target device to install the Robot Controller app. Your screen might look something like the image shown below.

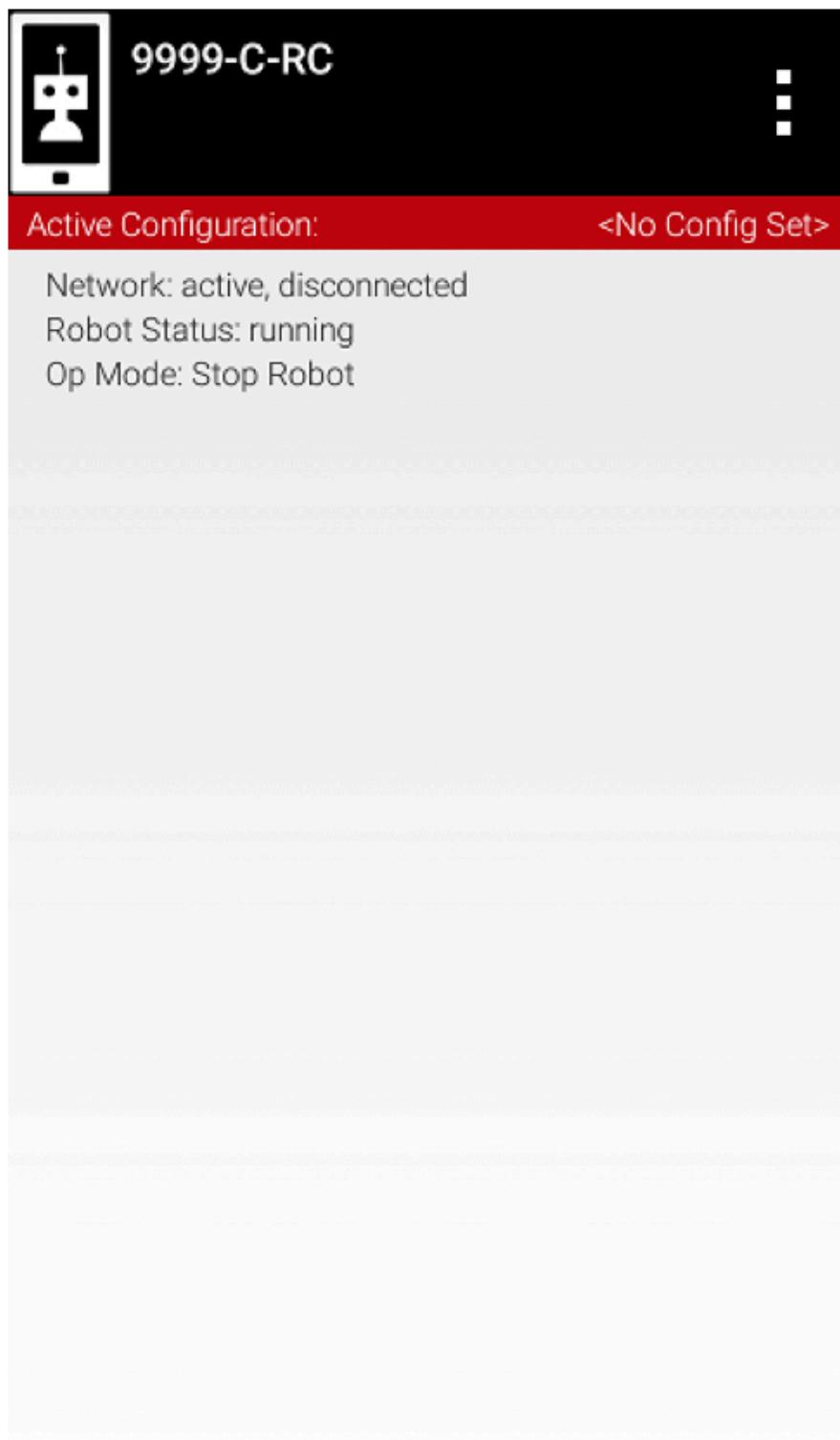


Make sure that you select the correct target device. In the figure above the Motorola phone is selected as the target device. Hit OK to build the APK file and install it on the target device.

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

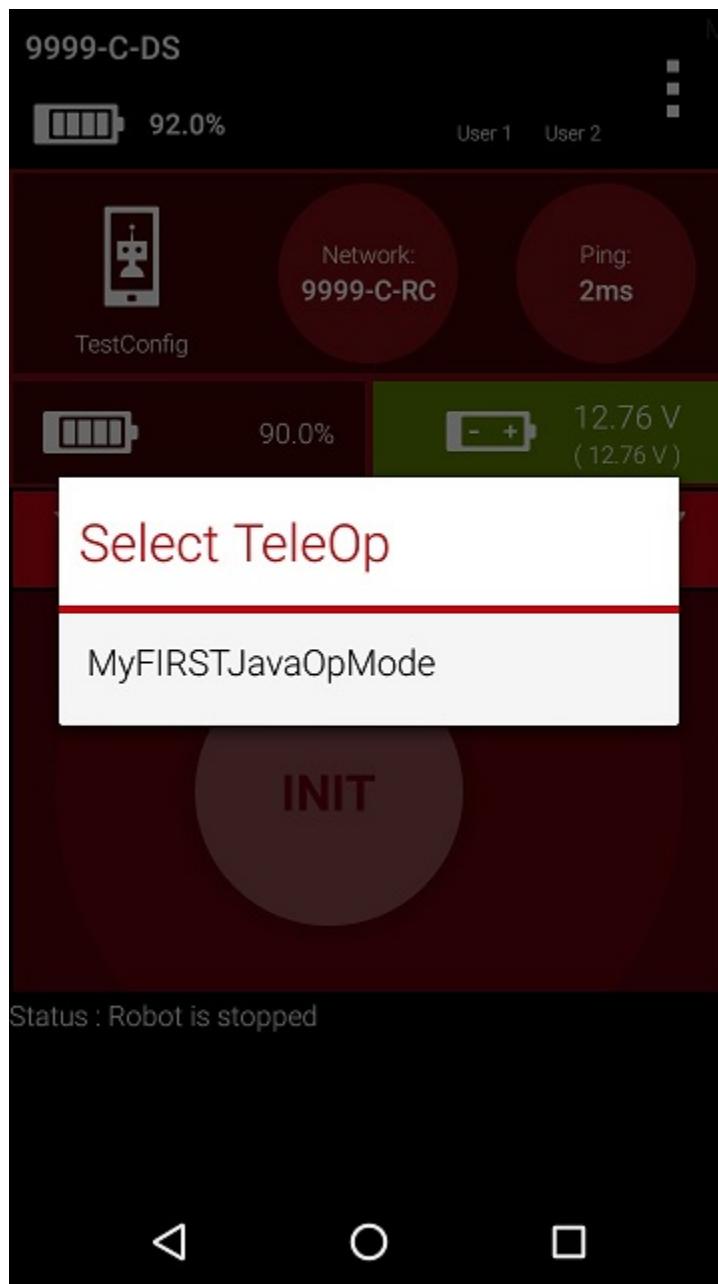
an Android phone as your Robot Controller, you should see the main Robot Controller app screen displayed on the phone.

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 Op Mode

If you successfully built and installed your updated Android app with your new op mode, then 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.

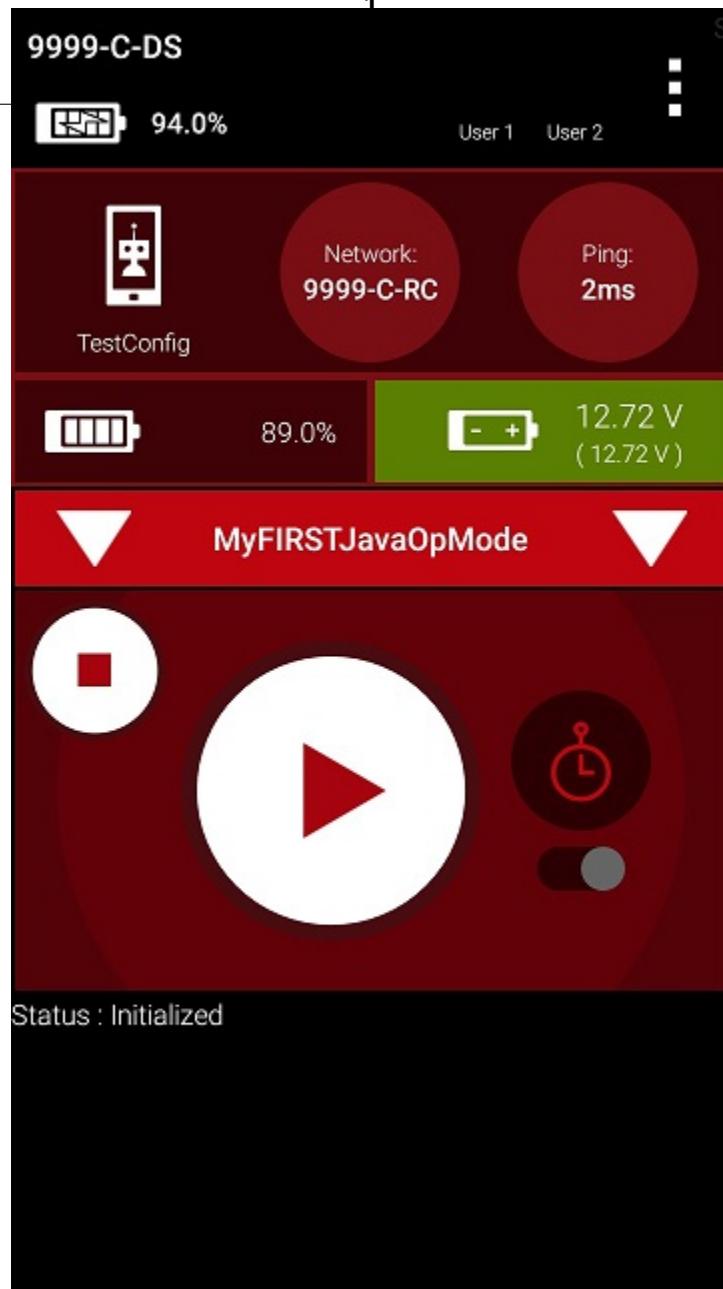
On the Driver Station, use the TeleOp dropdown list control to display the list of available op modes. Select your op mode ("MyFIRSTJavaOpMode") from the list.



Press the "INIT" button to initialize the op mode.



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.



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.



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 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 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

### FTC Docs

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.

### FTC Programming Resources 290

Before you connect your gamepad to the phone, verify that the switch on the bottom of the gamepad is set to the "X" (i.e., the "Xbox" mode) position.



Connect the gamepad to the Driver Station using the Micro USB OTG adapter cable.



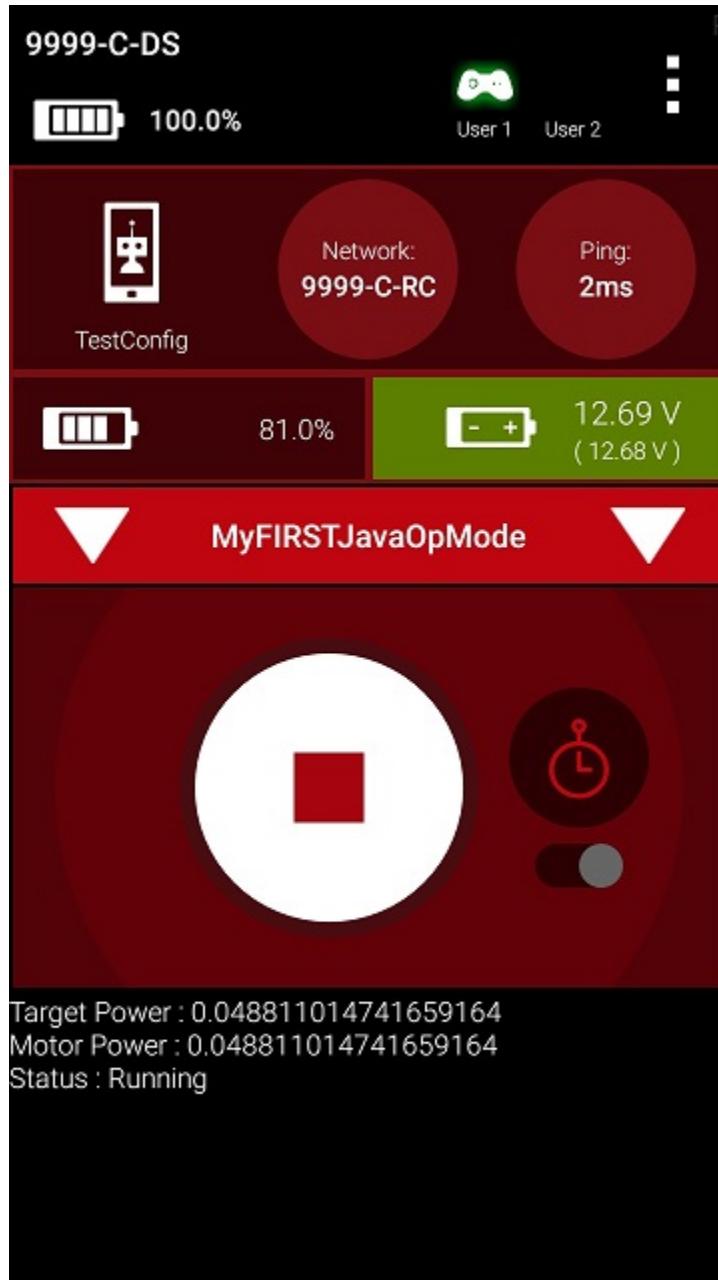
Your example op mode is looking for input from the gamepad designated as the user or driver #1. Press the Start button and the A button simultaneously on the Logitech F310 controller to designate your gamepad as user #1. Note that pushing the Start button and the B button simultaneously would designate the gamepad as user #2.



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.

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.



## Controlling a Servo AS

[FTC Docs](#)

In this section, you will modify your op mode to control a servo motor with the buttons of the gamepad.

[FTC Programming Resources, 293](#)

### 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)

```

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 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 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 an 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 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”).

## Chapter 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)*

## 2.1 Phone Pairing

### 2.1.1 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.

## 2.1.2 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.

## 2.1.3 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.

## 2.1.4 Pairing

FTC Docs [On RC phone: open the current season's RC app. Check Self Inspect for any RC issues.](#) [FTC Programming Resources, 299](#)

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.

## 2.1.5 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](mailto:westsiderobotics@verizon.net)

## 2.2 Managing a Control Hub

### 2.2.1 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.

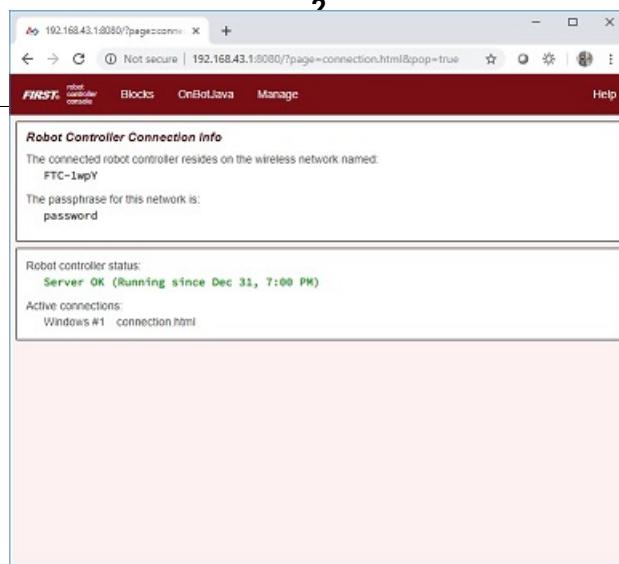
---

**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

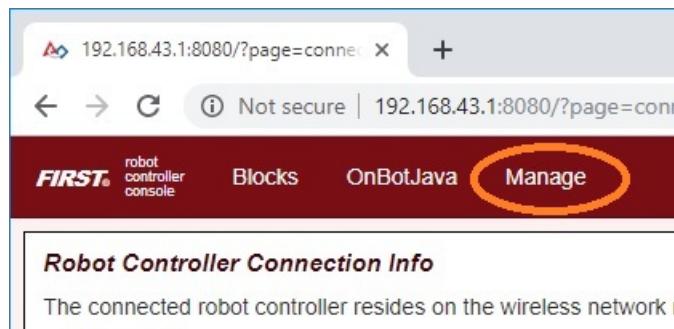
1. Verify that your laptop or Chromebook is connected to the Program & Manage wireless network of the Control Hub. If you are connected to the network, you should be able to see the *Robot Controller Connection Info* page when you navigate to address "192.168.43.1:8080":



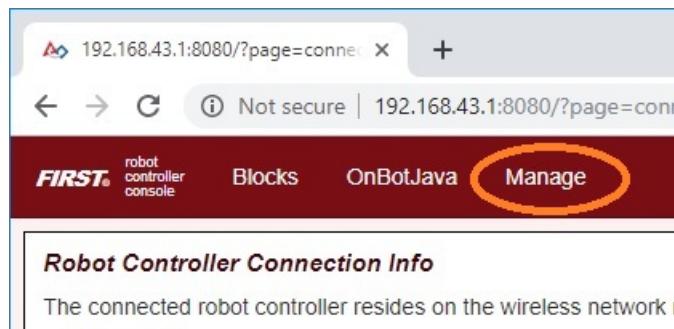
If your laptop or Chromebook is not connected and you are unable to access the *Robot Controller Connection Info* page, then read the instructions in the following tutorial to learn how to connect to the Program & Manage network.

#### *Connecting a laptop to the Program & Manage Network*

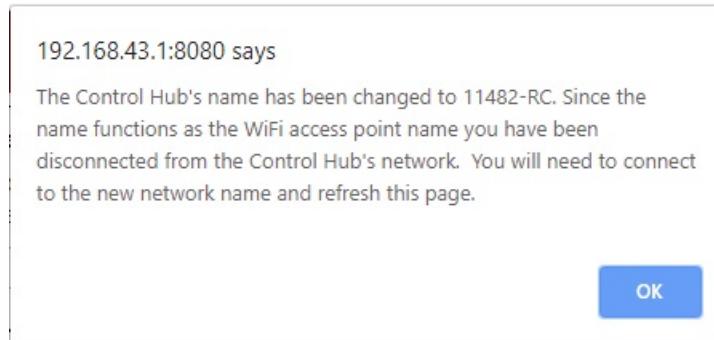
2. Click on the *Manage* link towards the top of the *Robot Controller Connection Info* page to navigate to the *Manage* page.



3. Change the name in the *Robot Controller Name* field and click on the *Change Name* button to change the Control Hub's name.



- 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.



## 2.2.2 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

- 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.

#### **Access Point Password**

Change the password for the wireless access point. Changing the password disconnects all clients from the Control Hub. You will need to reconnect using the new password.

New Password

Confirm Password

Show 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.

192.168.43.1:8080 says

The password has been changed, you have been disconnected from the Control Hub's network. You will need to login again and refresh this page.

OK

### 2.2.3 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).

## Press and Hold Button



- 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.

### 2.2.4 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.

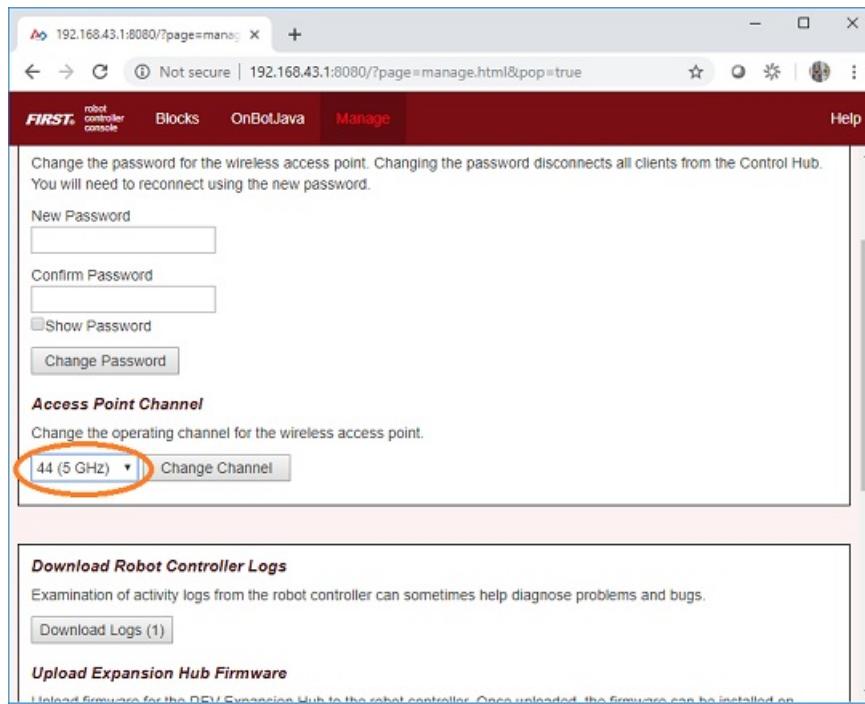
For example, at a large competition an FTA might ask that you switch to a designated channel to avoid wireless interference that is present in the venue. Similarly, an FTA might ask you to switch to a specific channel because the FTA is monitoring that designated channel for interference or other wireless disruptions.

You can select the operating channel for the Control Hub from the *Manage* page.

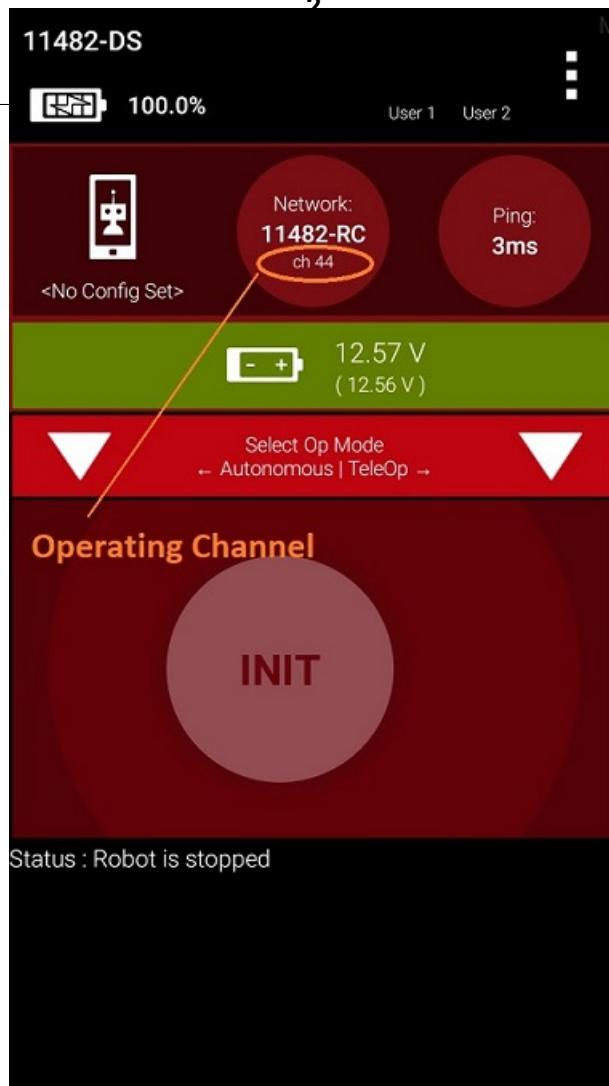
## Changing the WiFi Channel Instructions

**FTC Docs** 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.

**FTC Programming Resources** 304



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.

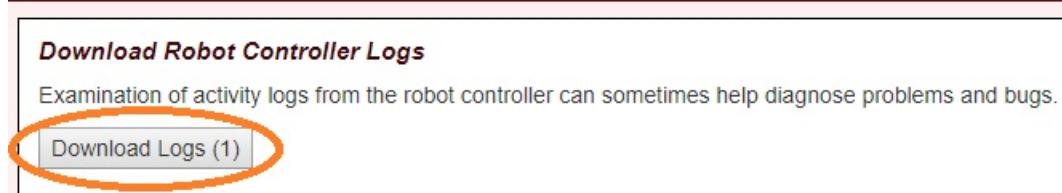


## 2.2.5 Downloading the Log File

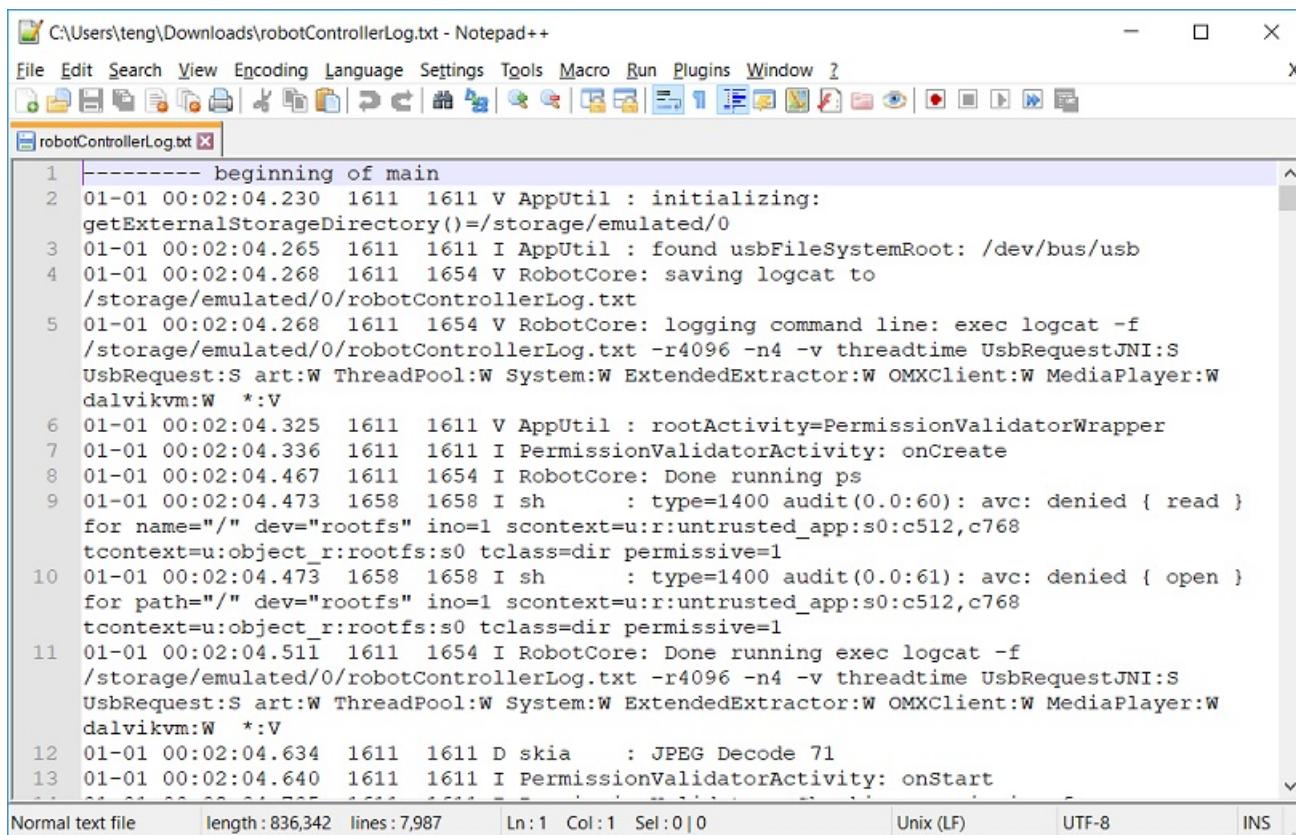
It's often helpful when troubleshooting problems with the Control System to download the log file from the Control Hub. This can be done from the *Manage* page. Note that the log file name is *robotControllerLog.txt* by default.

### 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.



```

1 ----- beginning of main
2 01-01 00:02:04.230 1611 1611 V AppUtil : initializing:
getExternalStorageDirectory()=/storage/emulated/0
3 01-01 00:02:04.265 1611 1611 I AppUtil : found usbFileSystemRoot: /dev/bus/usb
4 01-01 00:02:04.268 1611 1654 V RobotCore: saving logcat to
/storage/emulated/0/robotControllerLog.txt
5 01-01 00:02:04.268 1611 1654 V RobotCore: logging command line: exec logcat -f
/storage/emulated/0/robotControllerLog.txt -r4096 -n4 -v threadtime UsbRequestJNI:S
UsbRequest:S art:W ThreadPool:W System:W ExtendedExtractor:W OMXClient:W MediaPlayer:W
dalvikvm:W *:V
6 01-01 00:02:04.325 1611 1611 V AppUtil : rootActivity=PermissionValidatorWrapper
7 01-01 00:02:04.336 1611 1611 I PermissionValidatorActivity: onCreate
8 01-01 00:02:04.467 1611 1654 I RobotCore: Done running ps
9 01-01 00:02:04.473 1658 1658 I sh      : type=1400 audit(0.0:60): avc: denied { read }
for name="/" dev="rootfs" ino=1 scontext=u:r:untrusted_app:s0:c512,c768
tcontext=u:object_r:rootfs:s0 tclass=dir permissive=1
10 01-01 00:02:04.473 1658 1658 I sh      : type=1400 audit(0.0:61): avc: denied { open }
for path="/" dev="rootfs" ino=1 scontext=u:r:untrusted_app:s0:c512,c768
tcontext=u:object_r:rootfs:s0 tclass=dir permissive=1
11 01-01 00:02:04.511 1611 1654 I RobotCore: Done running exec logcat -f
/storage/emulated/0/robotControllerLog.txt -r4096 -n4 -v threadtime UsbRequestJNI:S
UsbRequest:S art:W ThreadPool:W System:W ExtendedExtractor:W OMXClient:W MediaPlayer:W
dalvikvm:W *:V
12 01-01 00:02:04.634 1611 1611 D skia    : JPEG Decode 71
13 01-01 00:02:04.640 1611 1611 I PermissionValidatorActivity: onStart

```

Normal text file    length: 836,342    lines: 7,987    Ln:1 Col:1 Sel:0|0    Unix (LF)    UTF-8    INS

## 2.2.6 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.

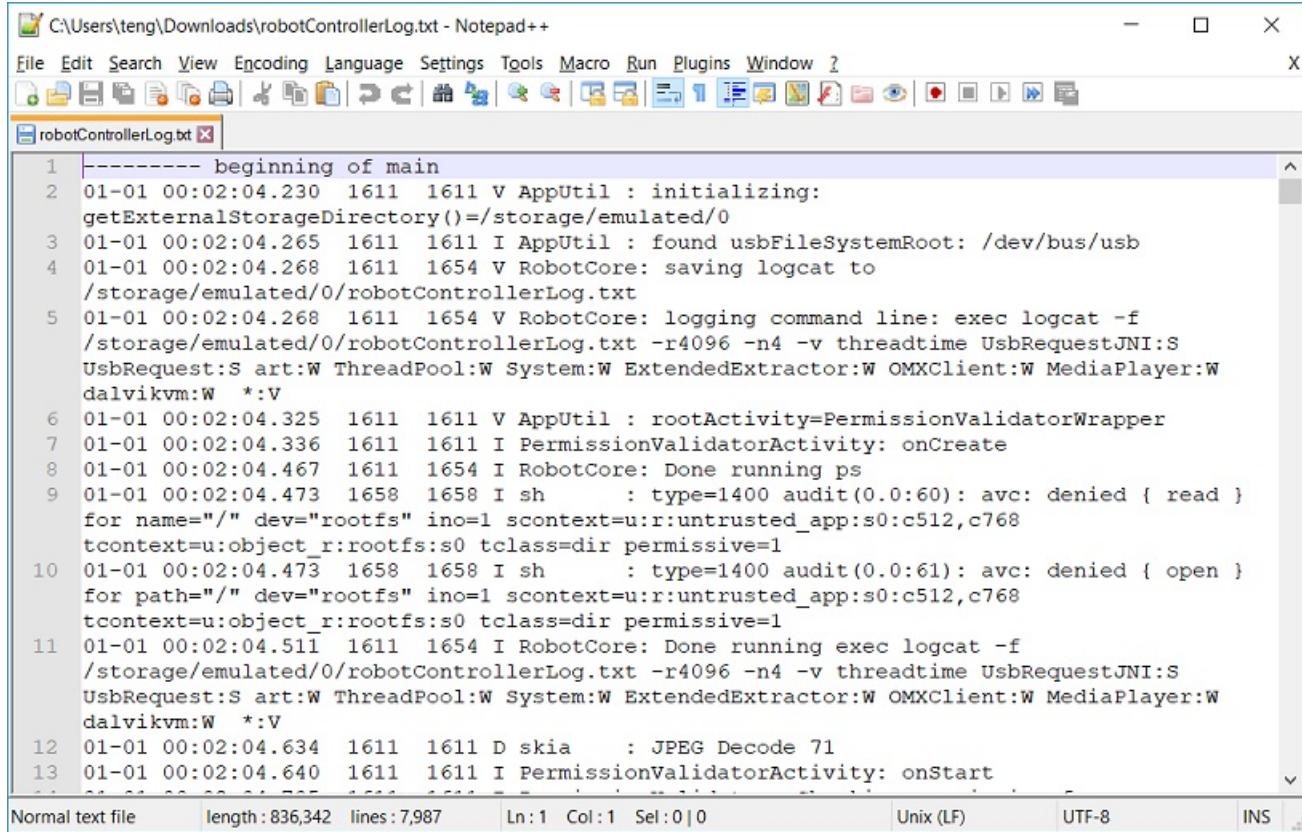
As an alternate, you can use the *Manage* interface from a connected laptop or Driver Station (DS) app to upload a Control Hub's firmware, or to update it using the included or uploaded version. New firmware images can be obtained from the [REV Robotics website](#).

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

**FTC Docs** On the Manage page of the Control Hub user interface, press the *Select Firmware* button to select the firmware file that you would like to upload.



```

1 ----- beginning of main
2 01-01 00:02:04.230 1611 1611 V AppUtil : initializing:
getExternalStorageDirectory()=/storage/emulated/0
3 01-01 00:02:04.265 1611 1611 I AppUtil : found usbFileSystemRoot: /dev/bus/usb
4 01-01 00:02:04.268 1611 1654 V RobotCore: saving logcat to
/storage/emulated/0/robotControllerLog.txt
5 01-01 00:02:04.268 1611 1654 V RobotCore: logging command line: exec logcat -f
/storage/emulated/0/robotControllerLog.txt -r4096 -n4 -v threadtime UsbRequestJNI:S
UsbRequest:S art:W ThreadPool:W System:W ExtendedExtractor:W OMXClient:W MediaPlayer:W
dalvikvm:W *:V
6 01-01 00:02:04.325 1611 1611 V AppUtil : rootActivity=PermissionValidatorWrapper
7 01-01 00:02:04.336 1611 1611 I PermissionValidatorActivity: onCreate
8 01-01 00:02:04.467 1611 1654 I RobotCore: Done running ps
9 01-01 00:02:04.473 1658 1658 I sh : type=1400 audit(0.0:60): avc: denied { read }
for name="/" dev="rootfs" ino=1 scontext=u:r:untrusted_app:s0:c512,c768
tcontext=u:object_r:rootfs:s0 tclass=dir permissive=1
10 01-01 00:02:04.473 1658 1658 I sh : type=1400 audit(0.0:61): avc: denied { open }
for path="/" dev="rootfs" ino=1 scontext=u:r:untrusted_app:s0:c512,c768
tcontext=u:object_r:rootfs:s0 tclass=dir permissive=1
11 01-01 00:02:04.511 1611 1654 I RobotCore: Done running exec logcat -f
/storage/emulated/0/robotControllerLog.txt -r4096 -n4 -v threadtime UsbRequestJNI:S
UsbRequest:S art:W ThreadPool:W System:W ExtendedExtractor:W OMXClient:W MediaPlayer:W
dalvikvm:W *:V
12 01-01 00:02:04.634 1611 1611 D skia : JPEG Decode 71
13 01-01 00:02:04.640 1611 1611 I PermissionValidatorActivity: onStart

```

Normal text file length: 836,342 lines: 7,987 Ln:1 Col:1 Sel:0|0 Unix (LF) UTF-8 INS

An *Upload* button should appear after you successfully selected a file.

- Press the *Upload* button to upload the firmware file from your computer to the Control Hub.

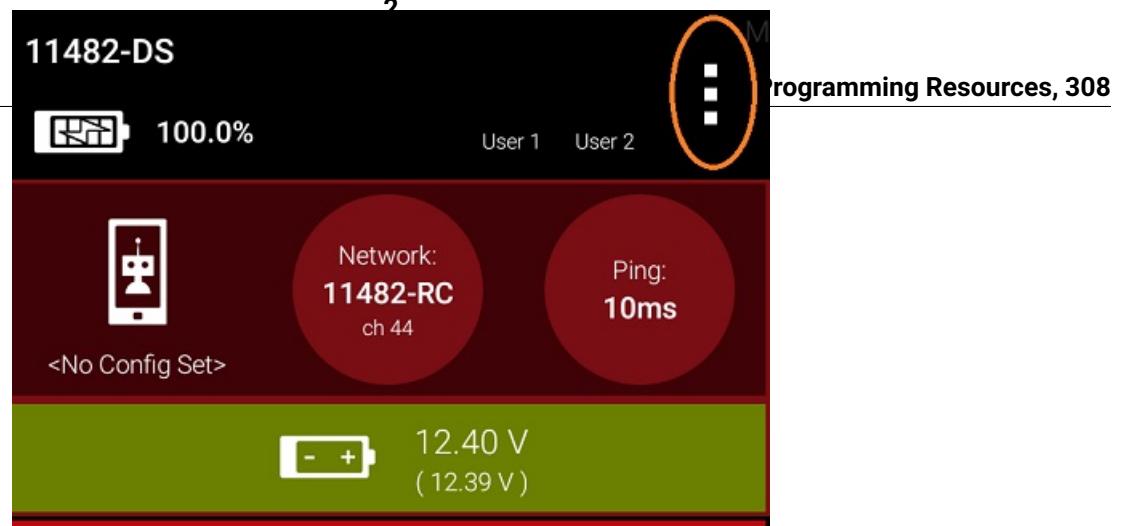
### Upload Expansion Hub Firmware

Upload firmware for the REV Expansion Hub to the robot controller. Once uploaded, the firmware can be installed on Expansion Hubs using the Advanced Settings menu on the robot controller or driver station.

Firmware upload complete

The words “Firmware upload complete” should appear once the file has been uploaded successfully.

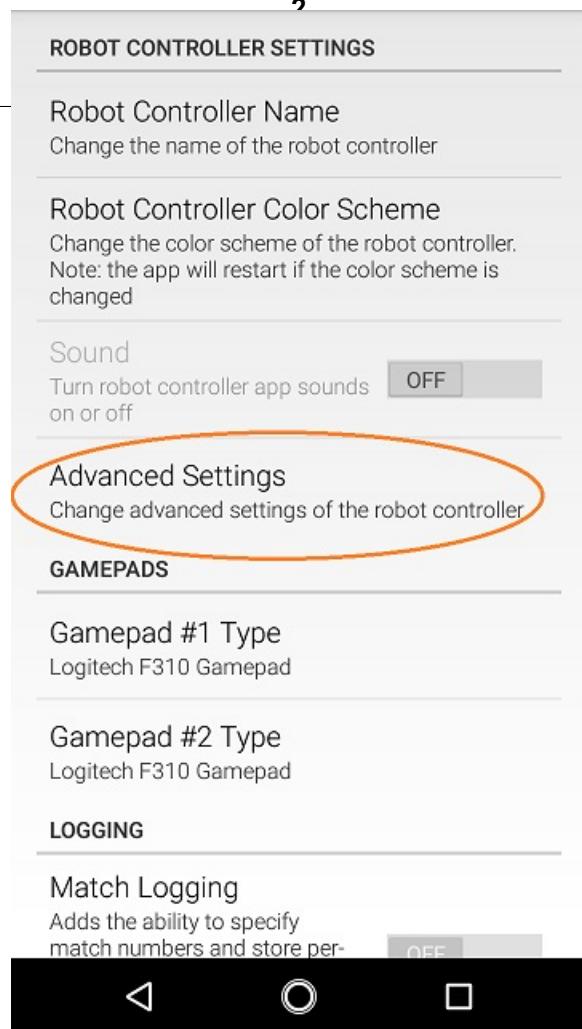
- On the Driver Station, touch the three dots in the upper right hand corner to display a pop-up menu.



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.

2

### 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

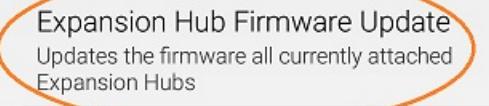
---

**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

---



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.

Expansion Hub firmware update file "REVHubFirmware\_1\_08\_02.bin" was found, as were the following Expansion Hubs:

Serial number: (embedded)  
Module address: 1  
Current Firmware: HW: 20, Maj: 1, Min: 7, Eng: 2

Click on the button below to update the firmware on each of these Expansion Hubs.

**WARNING:** While the firmware update is in progress, do not unplug hubs or restart the robot, or a hub may become permanently unusable.

**Update Expansion Hub Firmware**



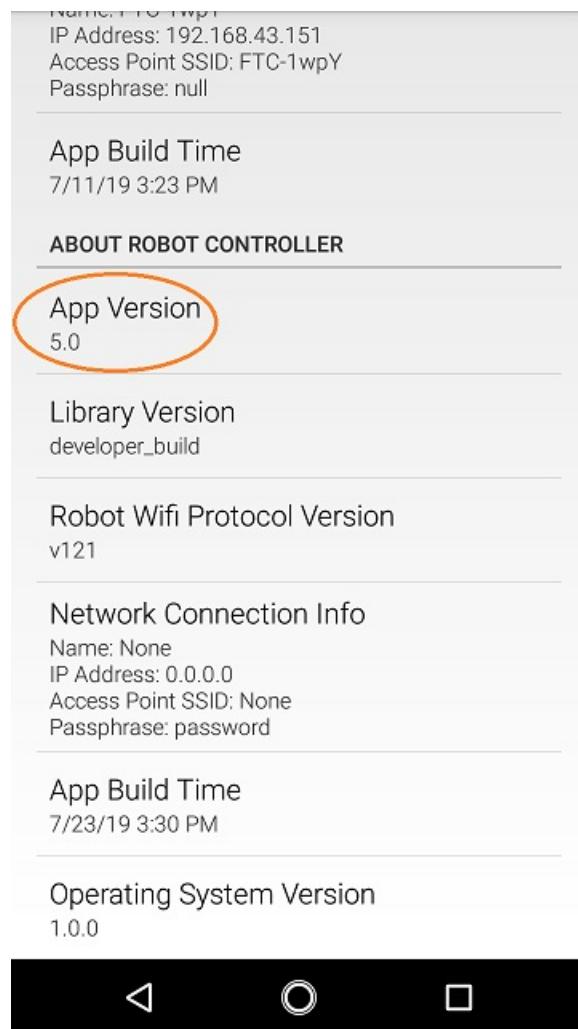
8. A progress bar will display while the firmware is being updated. Do not power off the Control Hub/Expansion Hub during this process. The Driver Station will display a message when the update process is complete.

Firmware update of Expansion Hub (embedded) succeeded.

## 2.2.7 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.

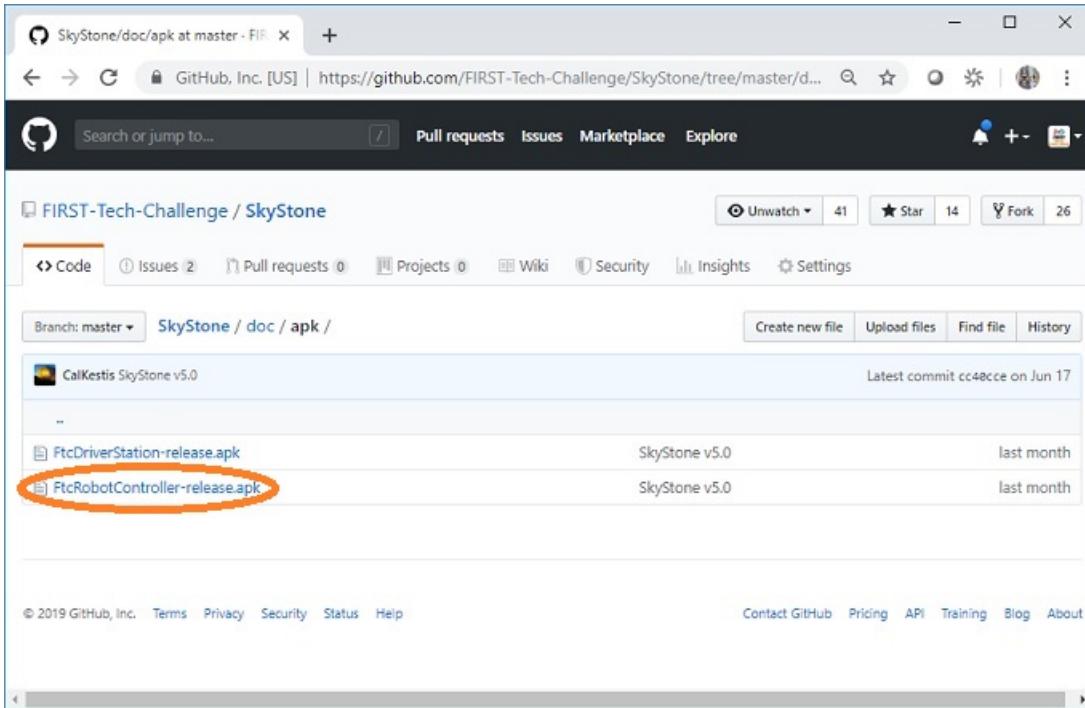
---

## Updating the Robot Controller App Instructions

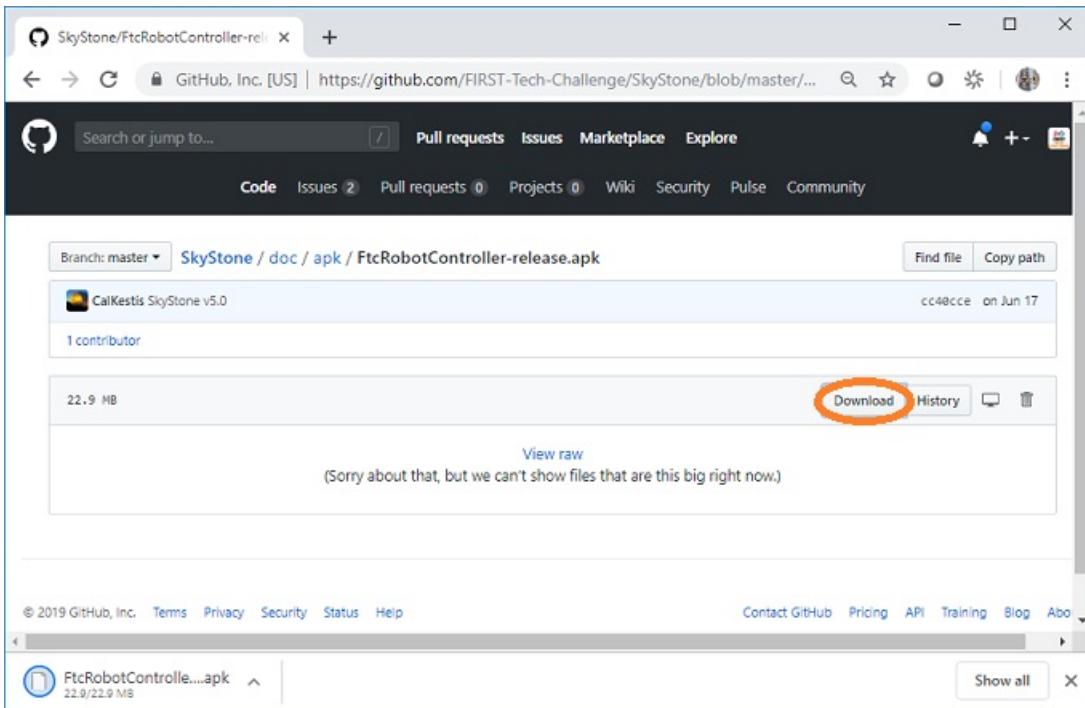
FTC Docs [Go to the GitHub repository.](#)

**FTC Programming Resources, 313**

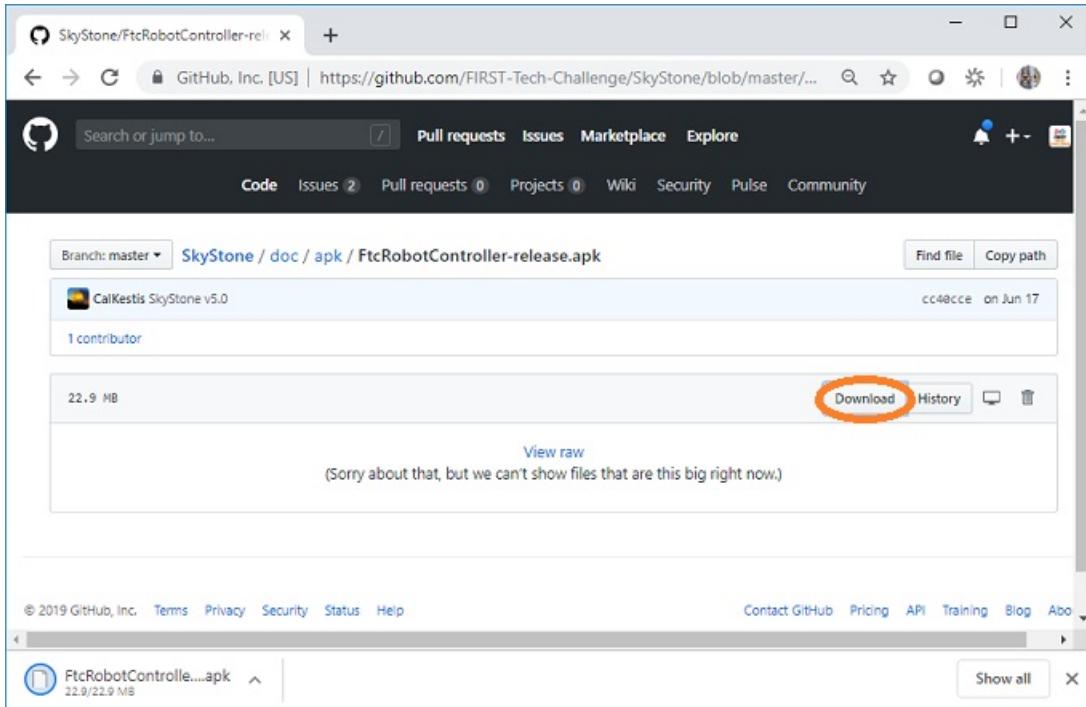
2. Locate the *FtcRobotController-release.apk* file.



3. Click on the filename (or *Download* button) to download the Robot Controller app as an APK file to your computer.

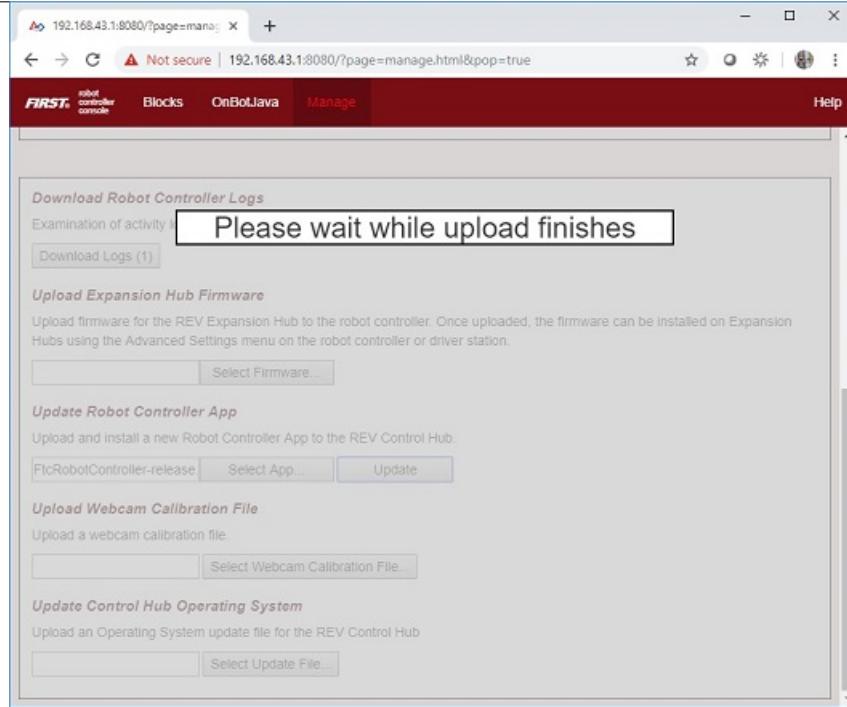


4. On the Manage page, click on the Select App button to select the Robot Controller app that you would like to upload to the Control Hub.



An *Update* button should appear if an APK file was successfully selected.

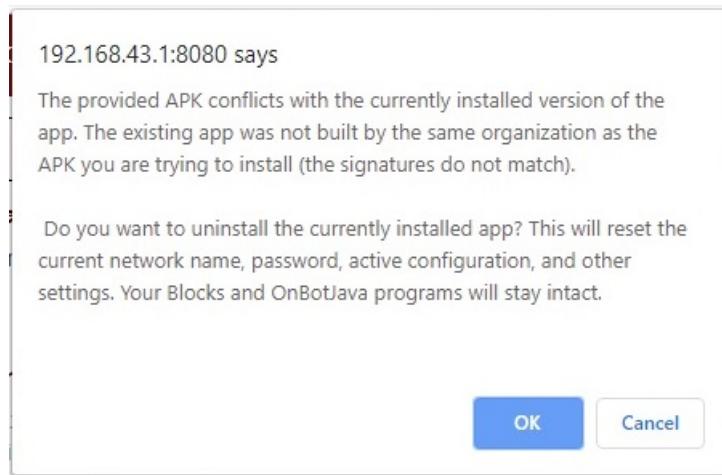
5. Click on the *Update* button to begin the update process.



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.

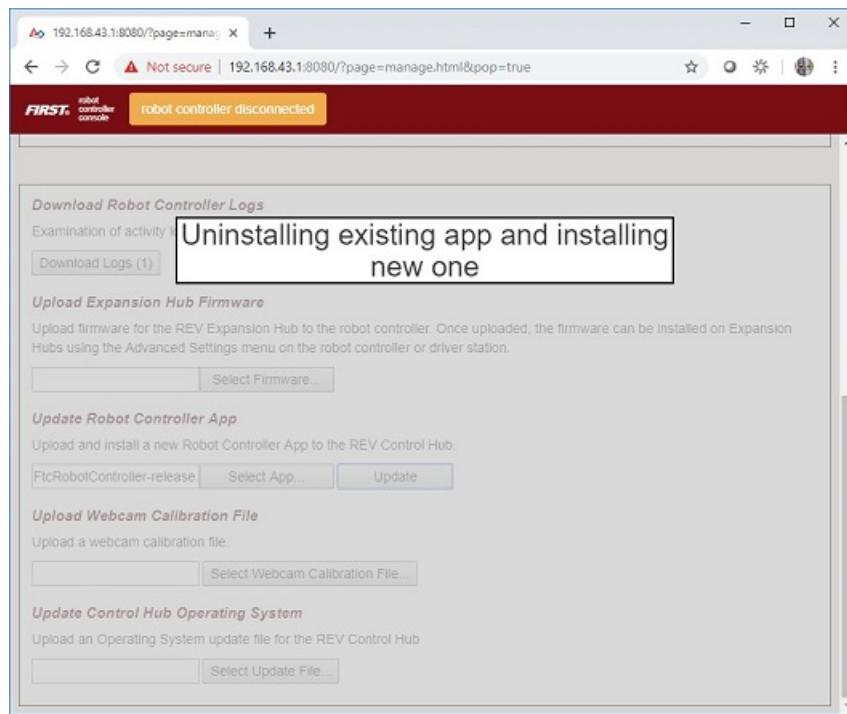
This difference in digital signatures can occur, for example, if the previous version of the app was built and installed using Android Studio, but the newer app was downloaded from the GitHub repository.

Press *OK* to uninstall the old app and continue with the update process.

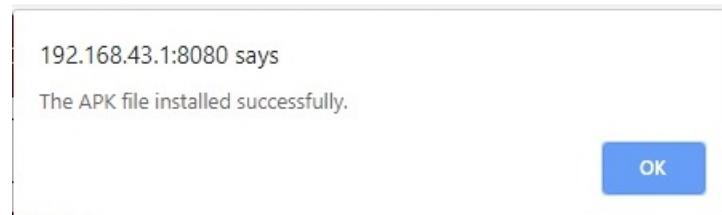


7. If the update process had to uninstall the previous version of the Robot Controller app, the network name and password for the Control Hub will be reset back to their factory values. If this happens, then you will need to reconnect your

computer to the Control Hub using the factory default values.



- 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.



## 2.2.8 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

[FTC Docs](#) [FTC Programming Resources, 317](#)

### Upload Webcam Calibration File

Upload a webcam calibration file.

mywebcamcalibrations.xml

Select Webcam Calibration File...

Upload

An *Upload* button should appear if a file was successfully selected.

- 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.

### Upload Webcam Calibration File

Upload a webcam calibration file.

mywebcamcalibrations.xml

Select Webcam Calibration File...

Upload

Webcam Calibration upload complete

## 2.2.9 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.

Name: FTC-1wpY  
 IP Address: 192.168.43.151  
 Access Point SSID: FTC-1wpY  
 Passphrase: null

App Build Time  
 7/11/19 3:23 PM

#### ABOUT ROBOT CONTROLLER

App Version  
 5.0

Library Version  
 developer\_build

Robot Wifi Protocol Version  
 v121

#### Network Connection Info

Name: None  
 IP Address: 0.0.0.0  
 Access Point SSID: None  
 Passphrase: password

App Build Time  
 7/23/19 3:30 PM

Operating System Version  
 1.0.0



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

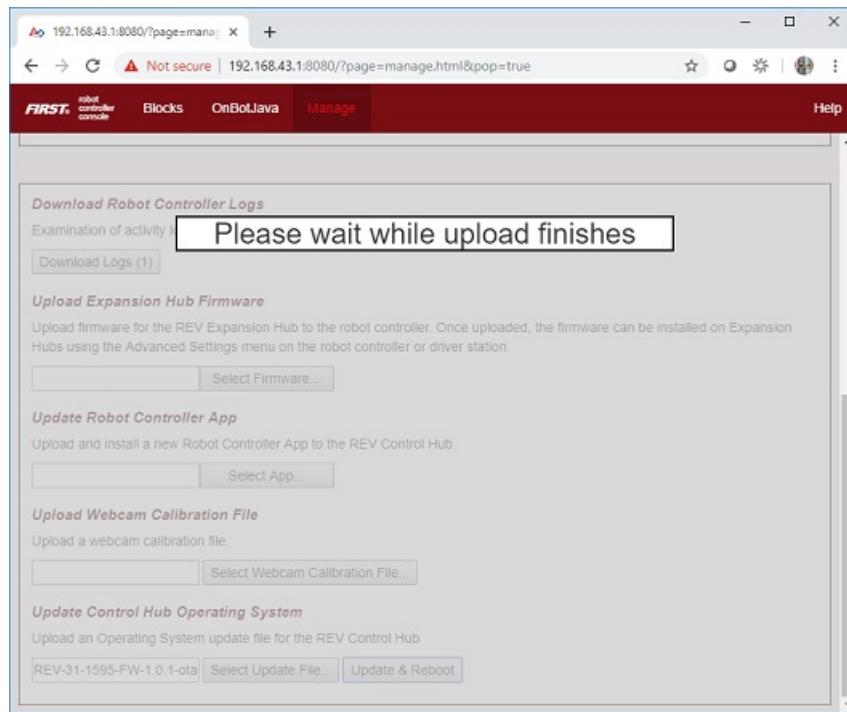
1. Download the new Control Hub OS update file from the [REV Robotics website](#).
2. On the *Manage* page, click on the *Select Update File* button to select the OS update file that you would like to upload.

##### ***Update Control Hub Operating System***

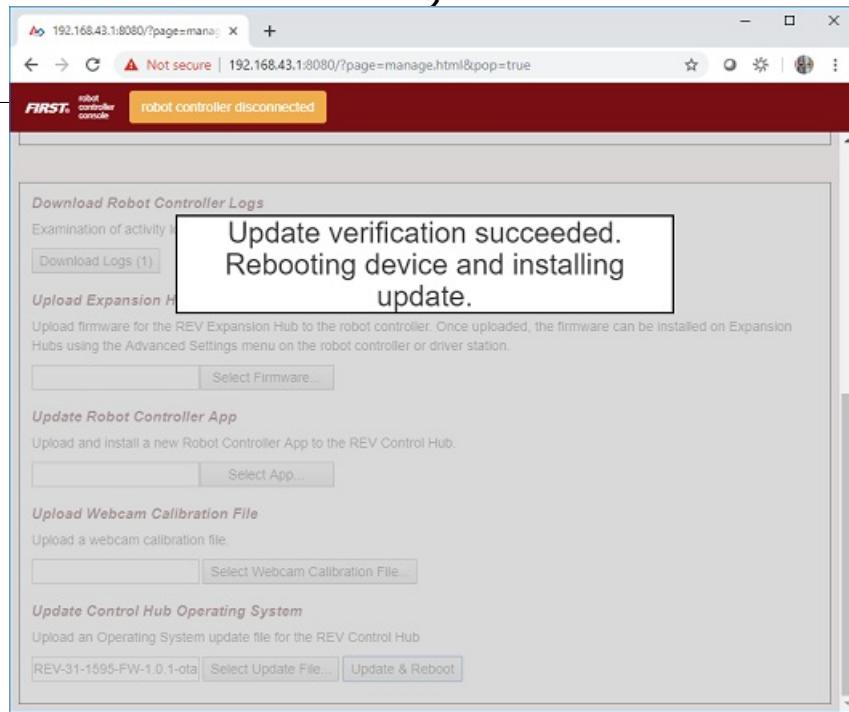
Upload an Operating System update file for the REV Control Hub

An *Update & Reboot* button should appear if an update file was successfully selected.

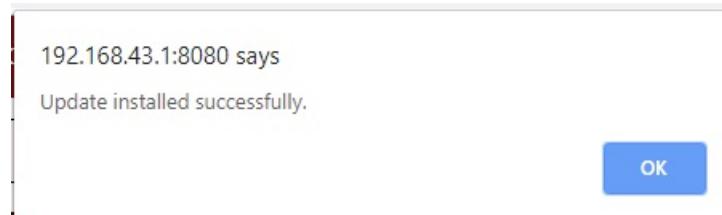
3. Click on the *Update & Reboot* button to start the update process. Please wait while the OS file gets uploaded to the Control Hub. Note that since the file is relatively large, it might take several minutes before the upload is complete. Do not turn off the Control Hub while the process is underway.



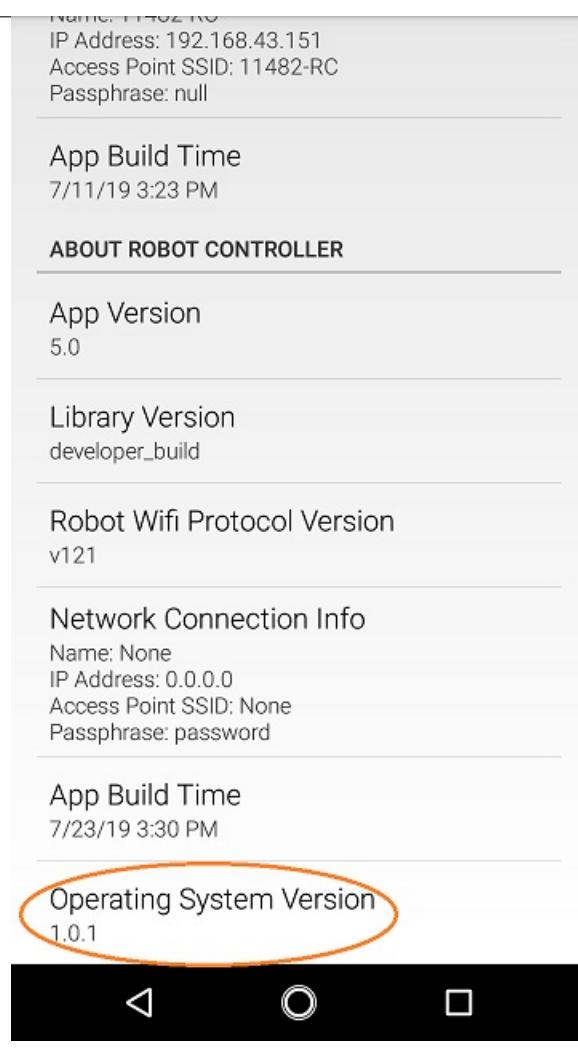
4. If the upload was successful, the *Manage* page will display a message indicating that the device is being rebooted and the update is being installed.



- 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.



### 2.2.10 Connecting to the Control Hub Using Wireless ADB

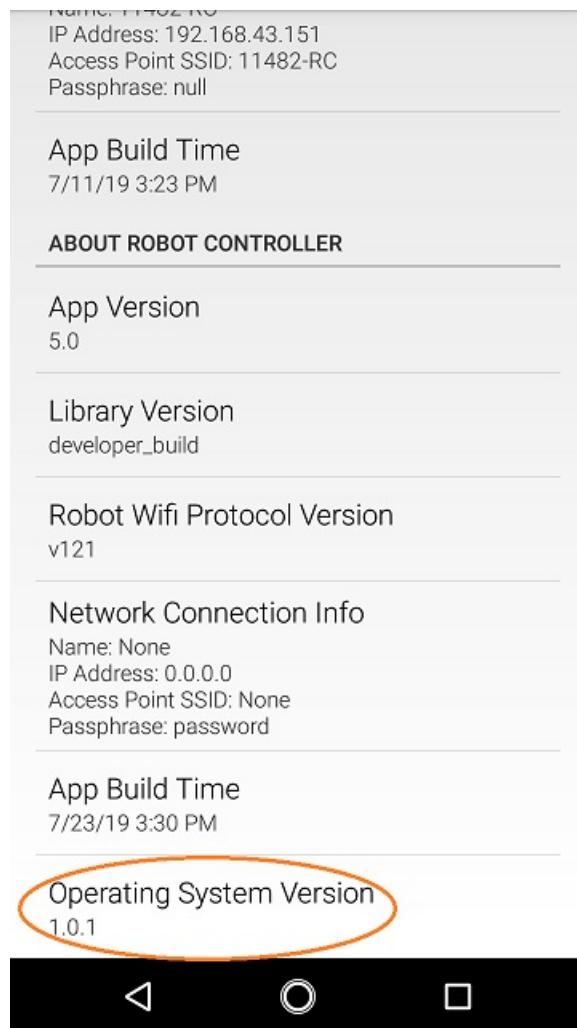
Advanced users who use Android Studio to build and install the Robot Controller app onto their Control Hub should be familiar with the Android Debug Bridge (adb) utility. adb is included with the Android development platform tools. It can be used to communicate with an Android device such as the Control 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

- 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](#)

- 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.
- 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.

## 2.3 Managing a Smartphone Driver Station

### 2.3.1 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.

### 2.3.2 Changing the Name

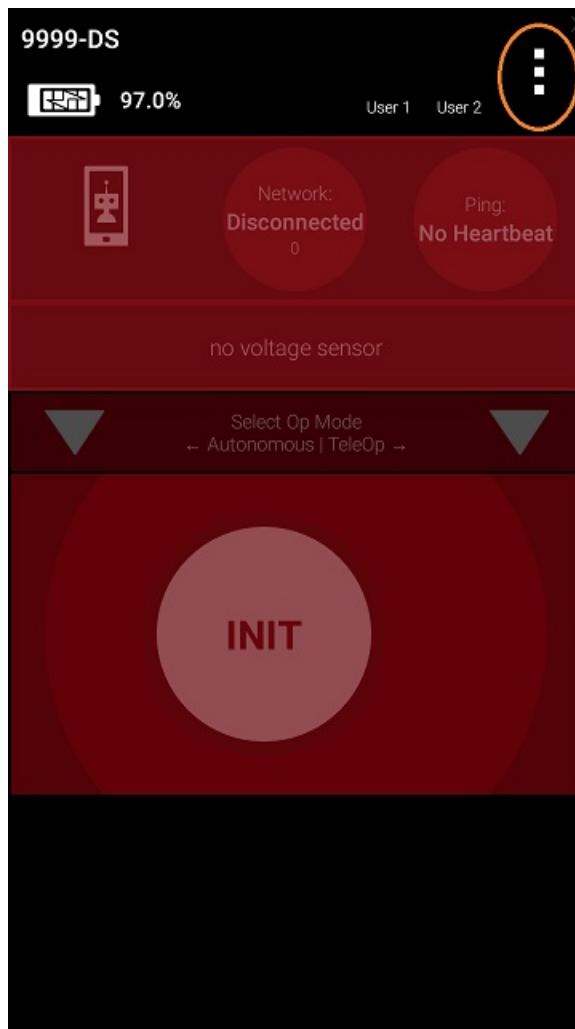
In order to comply with game manual rule <RS01>, the name of the Driver Station (DS) smartphone should be changed.

This can be done in the DS app, as described below.

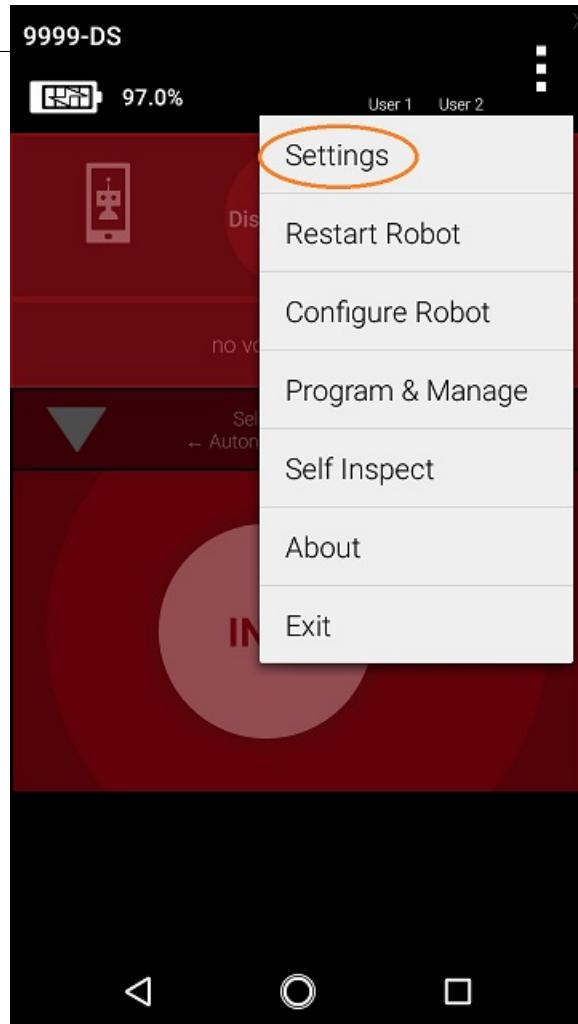
As an alternate, [found here](#) show how to rename a smartphone using the Android Settings activity of the phone.

#### Changing the Name of a Driver Station Instructions

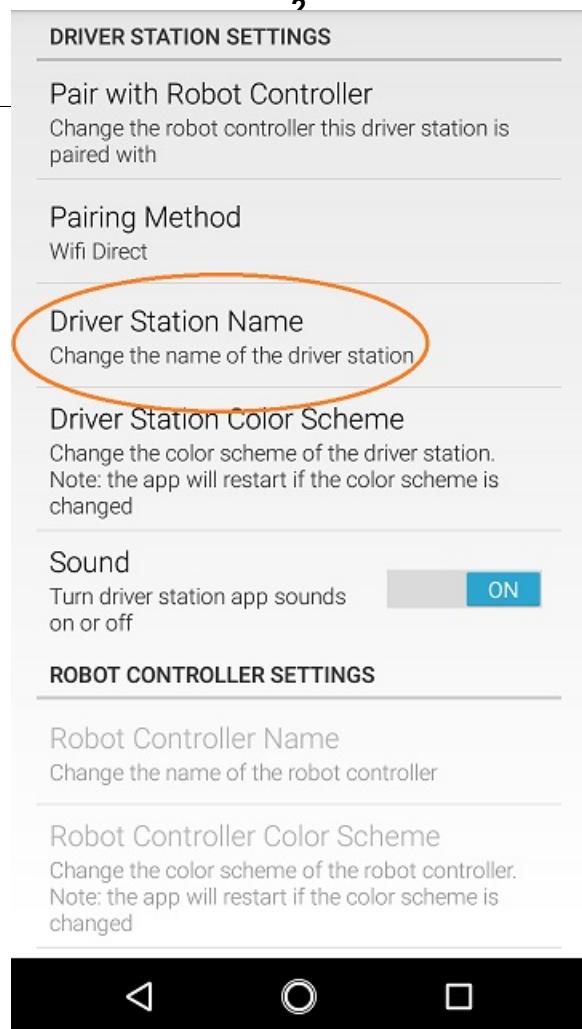
1. On the Driver Station phone, touch the three dots in the upper right hand corner to display a pop-up menu.



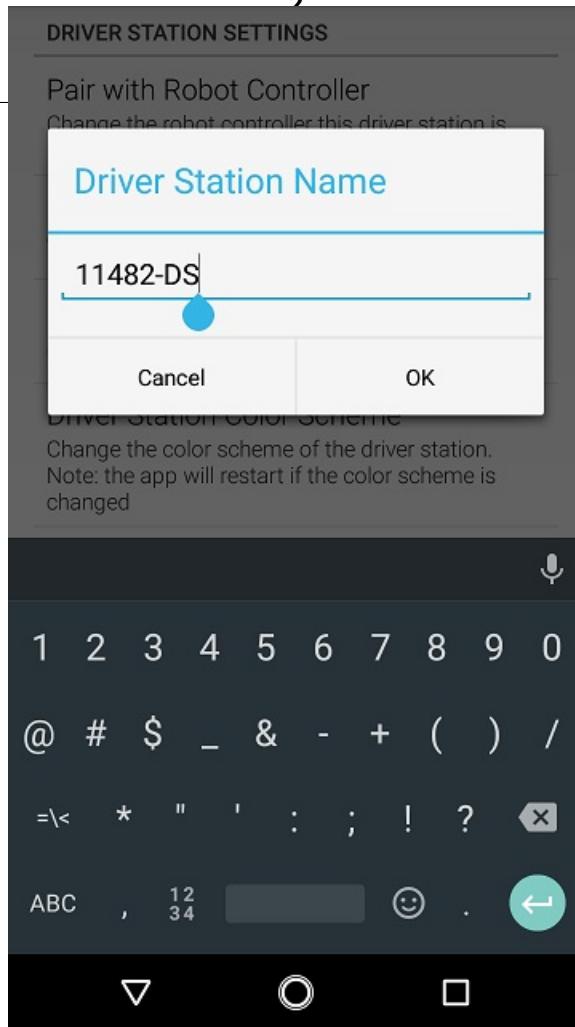
2. Select the *Settings* menu item from the pop-up menu.



3. Click on *Driver Station Name* on the *DRIVER STATION SETTINGS* page.



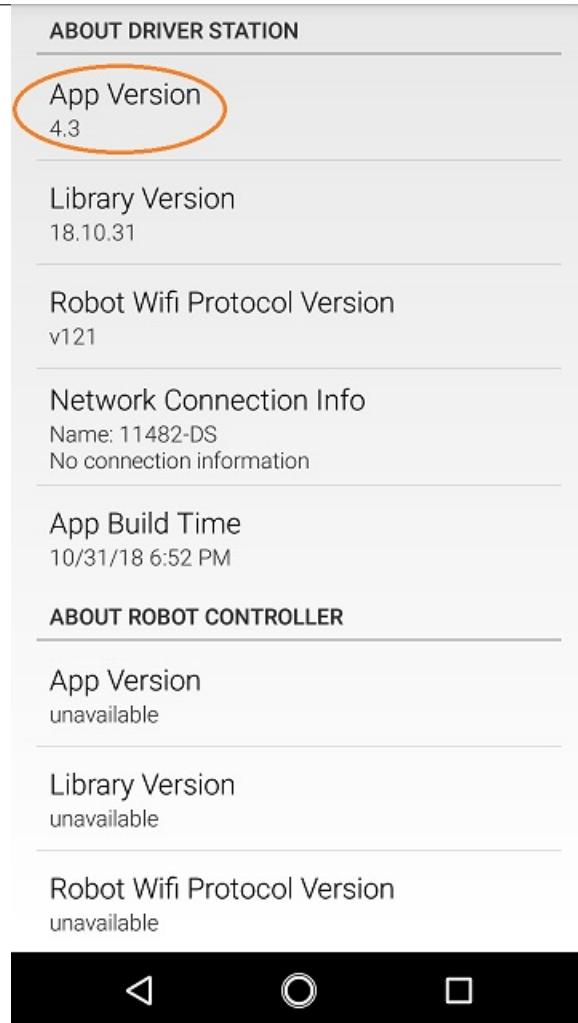
4. Specify the new Driver Station Name and press *OK* to accept the changes.



### 2.3.3 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 [FTC Robot Controller 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".

**FTC Docs** **FTC Programming Resources** **328**  
**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](#)

---

## 2.4 Managing a Smartphone Robot Controller

### 2.4.1 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.

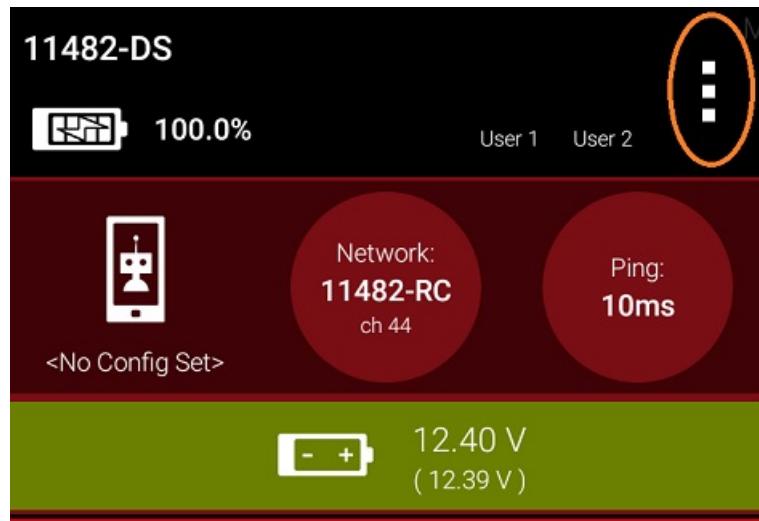
---

**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.

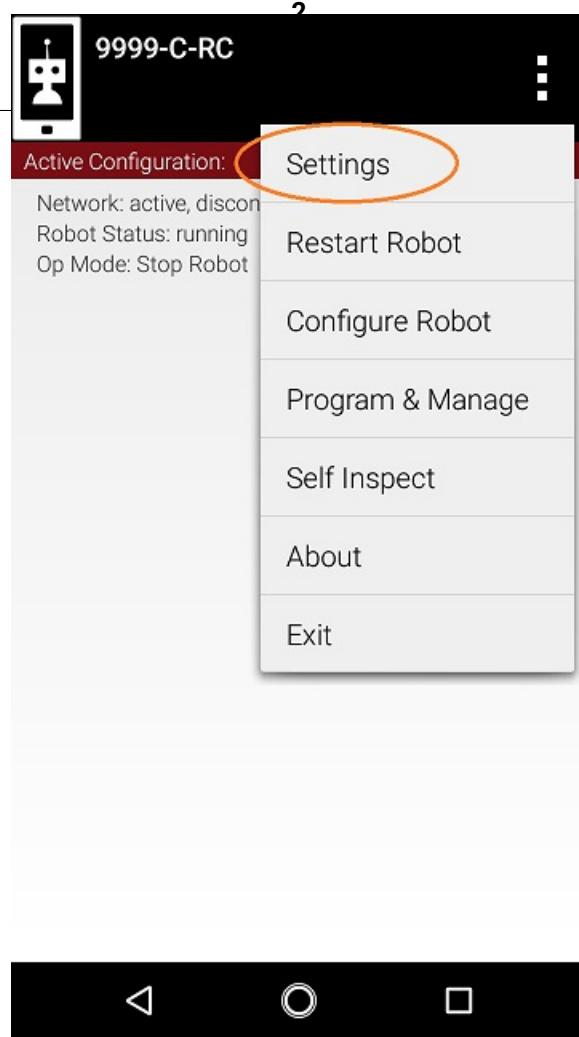
---

#### Changing the Name of a Robot Controller

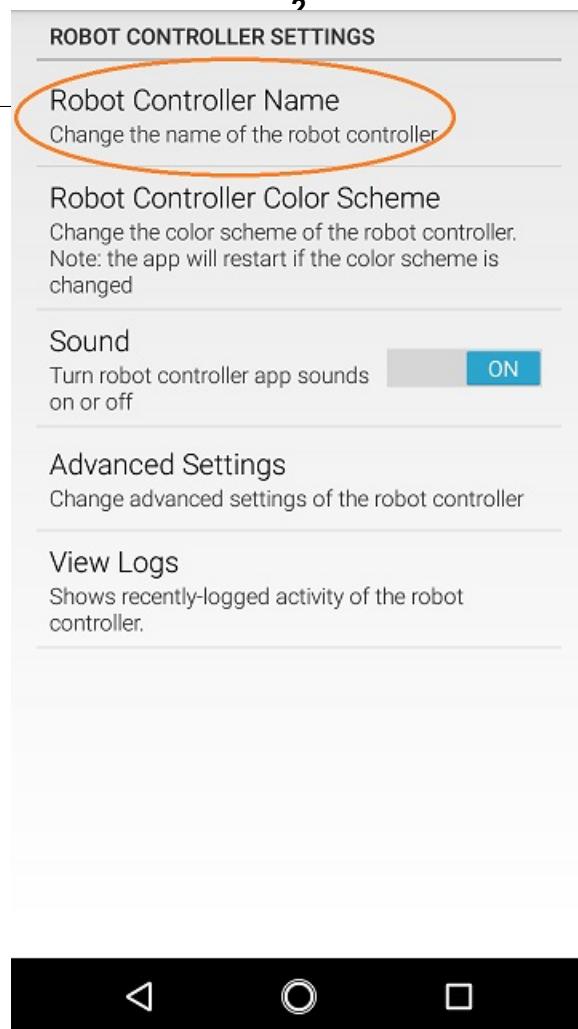
1. On the Robot Controller phone or paired Driver Station phone, touch the three dots in the upper right hand corner to display a pop-up menu.



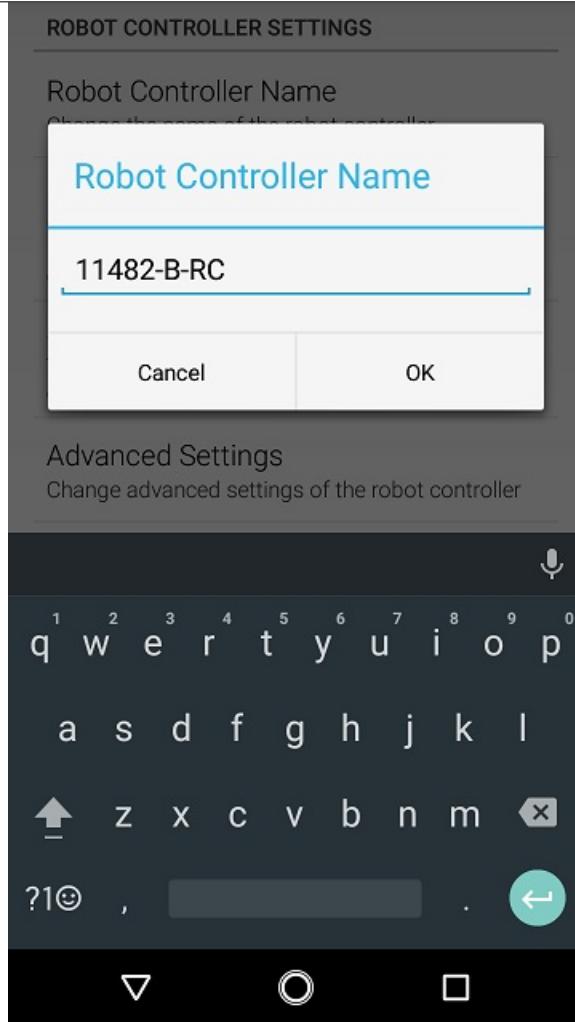
2. Select the *Settings* menu item from the pop-up menu.



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.



#### 2.4.2 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.

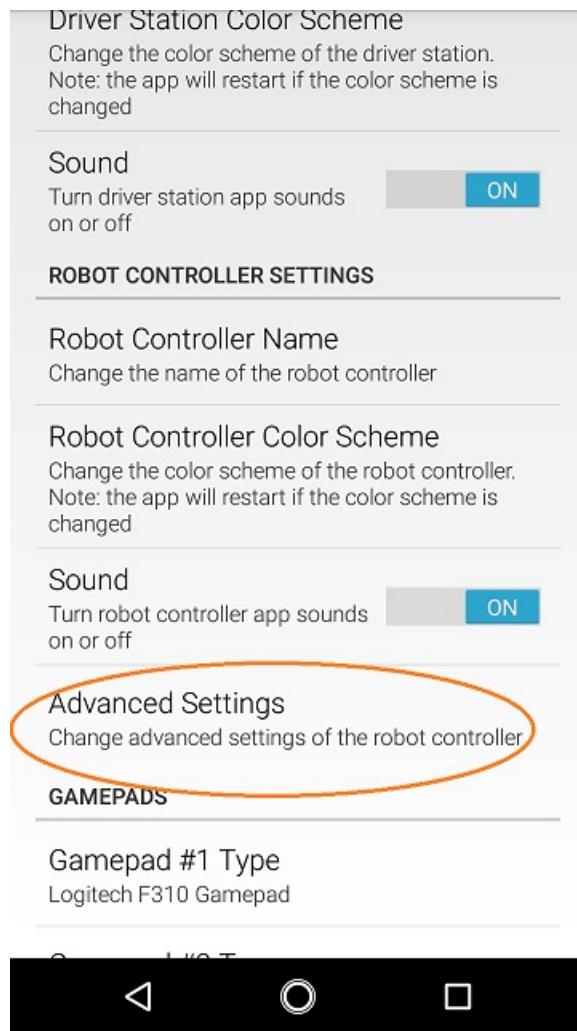
## Changing the WiFi Channel Instructions

**FTC Docs** Verify that the Driver Station is connected to your Robot Controller.

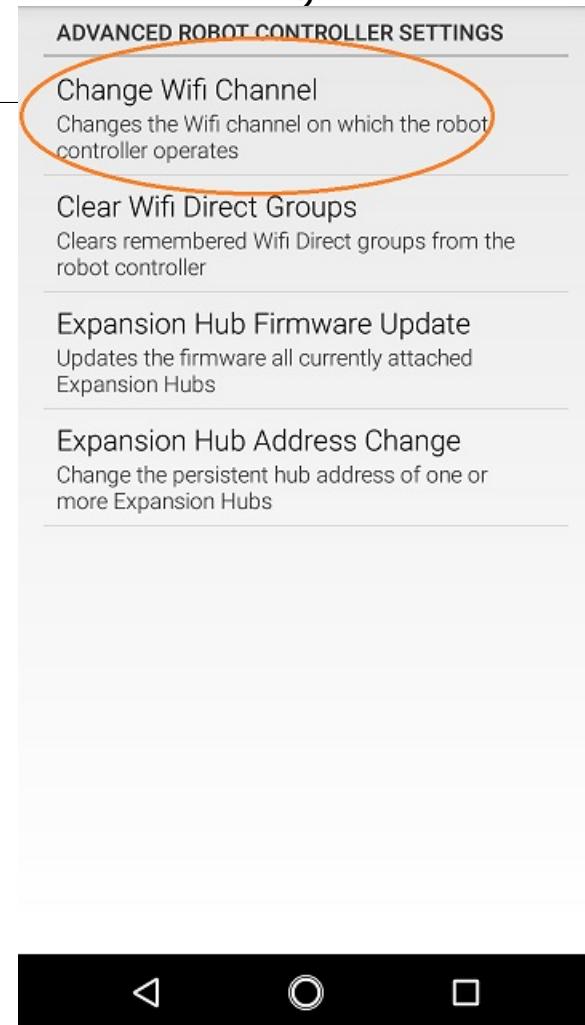
**FTC Programming Resources, 332**

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.

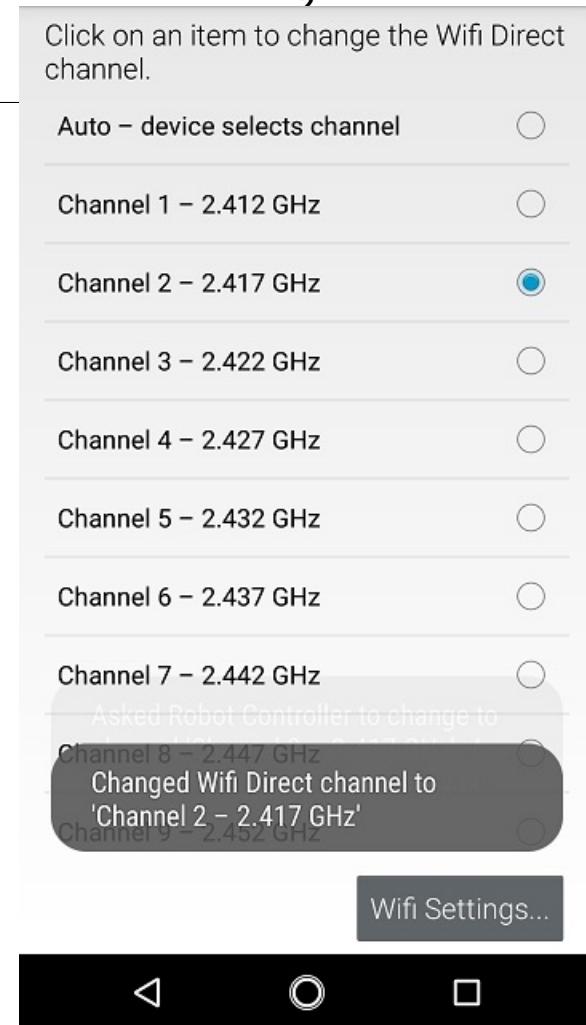
3. Scroll down to the *ROBOT CONTROLLER SETTINGS* section of the *Settings* screen and click on the words *Advanced Settings* to display the *ADVANCED ROBOT CONTROLLER SETTINGS* activity.



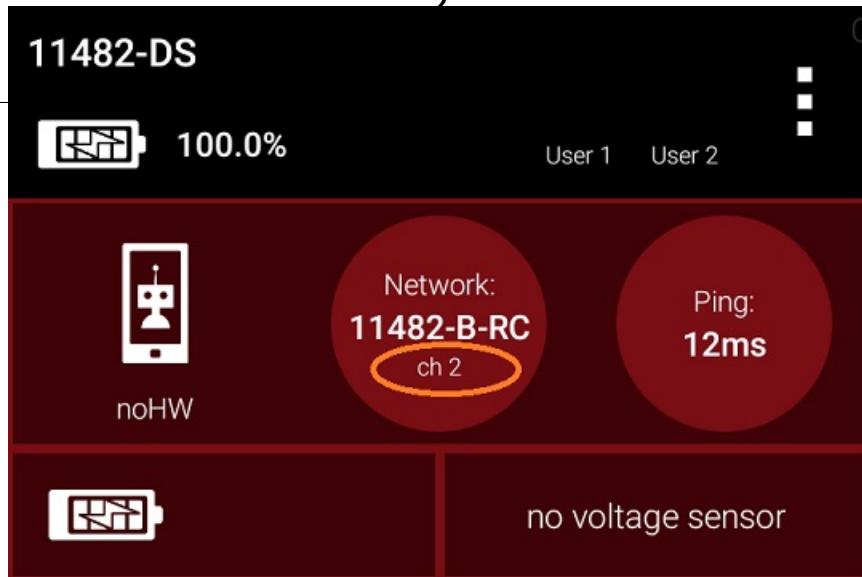
4. Click on the *Change Wifi Channel* link to display a list of available channels.



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



### 2.4.3 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":

The screenshot shows the 'Manage' section of the FIRST robot controller console. It includes fields for changing the robot controller name (11482-B-RC), download logs (1), upload expansion hub firmware, and upload a webcam calibration file.

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*

- Click on the *Manage* link towards the top of the *Robot Controller Connection Info* page to navigate to the Manage page.

The screenshot shows the 'Manage' section of the FIRST robot controller console. The 'Manage' link in the header is highlighted with a red circle.

- Click the *Download Logs* button to download the Robot Controller log file.

### Download Robot Controller Logs

Examination of activity logs from the robot controller can sometimes help diagnose problems and bugs.

**Download Logs (1)**

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.

```

1 ----- beginning of main
2 01-01 00:02:04.230 1611 1611 V AppUtil : initializing:
getExternalStorageDirectory()=/storage/emulated/0
3 01-01 00:02:04.265 1611 1611 I AppUtil : found usbFileSystemRoot: /dev/bus/usb
4 01-01 00:02:04.268 1611 1654 V RobotCore: saving logcat to
/storage/emulated/0/robotControllerLog.txt
5 01-01 00:02:04.268 1611 1654 V RobotCore: logging command line: exec logcat -f
/storage/emulated/0/robotControllerLog.txt -r4096 -n4 -v threadtime UsbRequestJNI:S
UsbRequest:S art:W ThreadPool:W System:W ExtendedExtractor:W OMXClient:W MediaPlayer:W
dalvikvm:W *:V
6 01-01 00:02:04.325 1611 1611 V AppUtil : rootActivity=PermissionValidatorWrapper
7 01-01 00:02:04.336 1611 1611 I PermissionValidatorActivity: onCreate
8 01-01 00:02:04.467 1611 1654 I RobotCore: Done running ps
9 01-01 00:02:04.473 1658 1658 I sh      : type=1400 audit(0.0:60): avc: denied { read }
for name="/" dev="rootfs" ino=1 scontext=u:r:untrusted_app:s0:c512,c768
tcontext=u:object_r:rootfs:s0 tclass=dir permissive=1
10 01-01 00:02:04.473 1658 1658 I sh      : type=1400 audit(0.0:61): avc: denied { open }
for path="/" dev="rootfs" ino=1 scontext=u:r:untrusted_app:s0:c512,c768
tcontext=u:object_r:rootfs:s0 tclass=dir permissive=1
11 01-01 00:02:04.511 1611 1654 I RobotCore: Done running exec logcat -f
/storage/emulated/0/robotControllerLog.txt -r4096 -n4 -v threadtime UsbRequestJNI:S
UsbRequest:S art:W ThreadPool:W System:W ExtendedExtractor:W OMXClient:W MediaPlayer:W
dalvikvm:W *:V
12 01-01 00:02:04.634 1611 1611 D skia     : JPEG Decode 71
13 01-01 00:02:04.640 1611 1611 I PermissionValidatorActivity: onStart

```

Normal text file length: 836,342 lines: 7,987 Ln:1 Col:1 Sel:0|0 Unix (LF) UTF-8 INS

## 2.4.4 Updating the Expansion Hub Firmware

**ETC Docs** 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 select the firmware file that you would like to upload.

#### ***Upload Expansion Hub Firmware***

Upload firmware for the REV Expansion Hub to the robot controller. Once uploaded, the firmware can be installed on Expansion Hubs using the Advanced Settings menu on the robot controller or driver station.

REVHubFirmware_1_08_0:	<a href="#">Select Firmware...</a>	<a href="#">Upload</a>
------------------------	------------------------------------	------------------------

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.

#### ***Upload Expansion Hub Firmware***

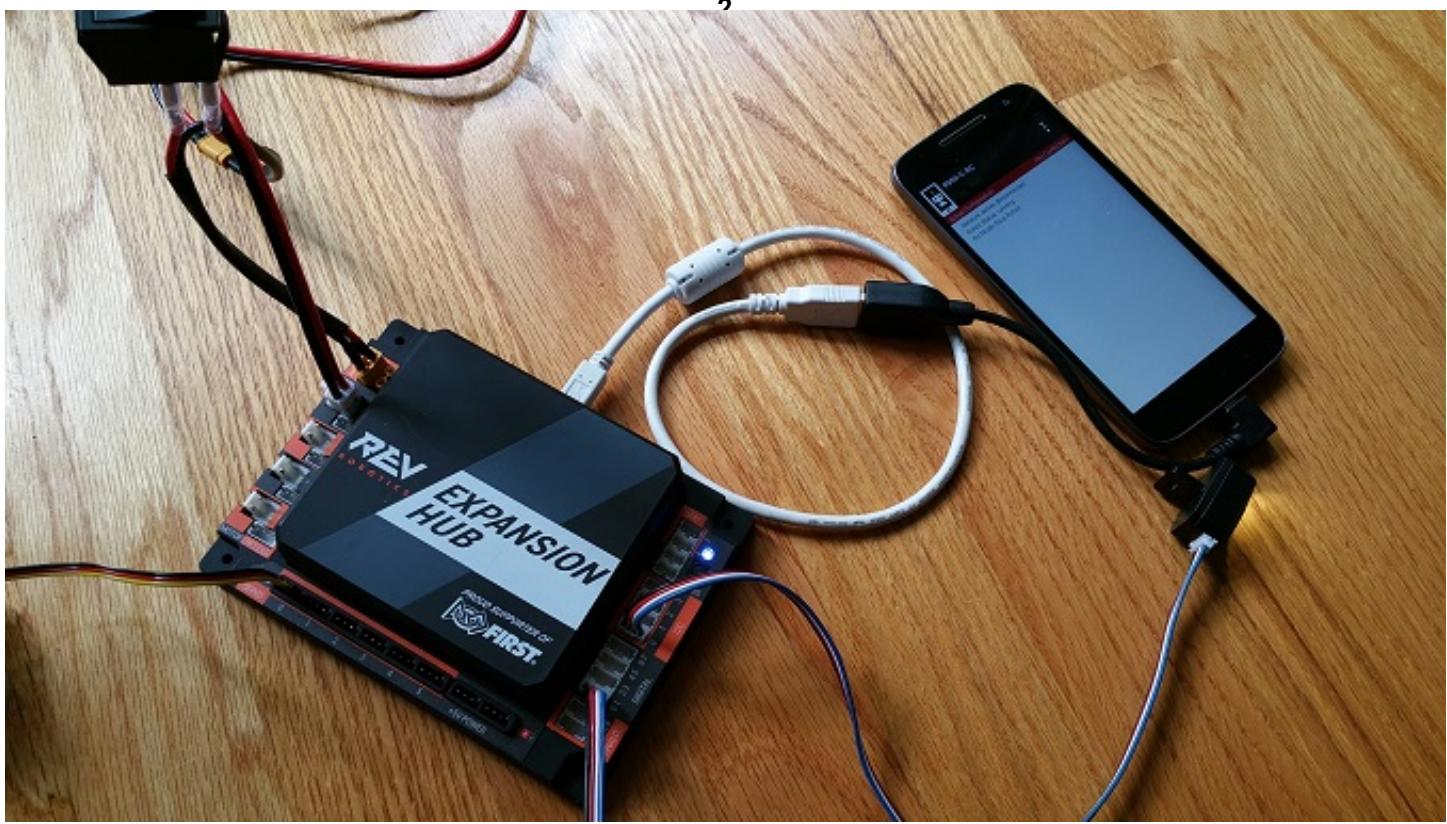
Upload firmware for the REV Expansion Hub to the robot controller. Once uploaded, the firmware can be installed on Expansion Hubs using the Advanced Settings menu on the robot controller or driver station.

REVHubFirmware_1_08_0:	<a href="#">Select Firmware...</a>	<a href="#">Upload</a>
------------------------	------------------------------------	------------------------

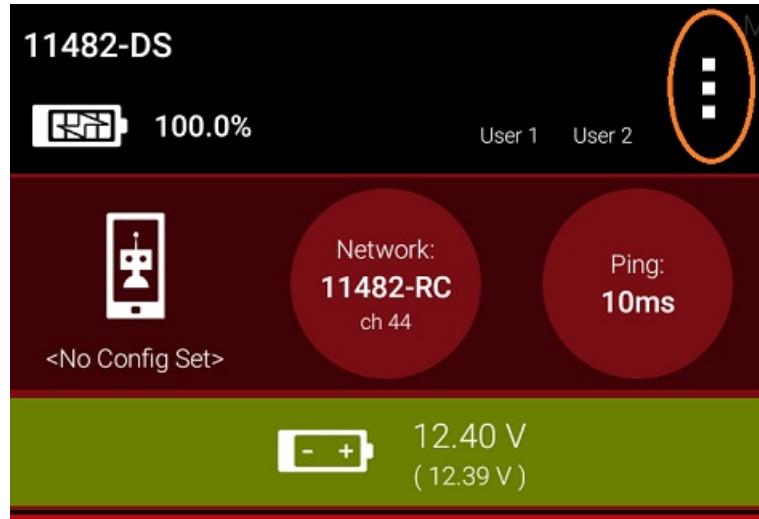
**Firmware upload complete**

The words "Firmware upload complete" should appear once the file has been uploaded successfully.

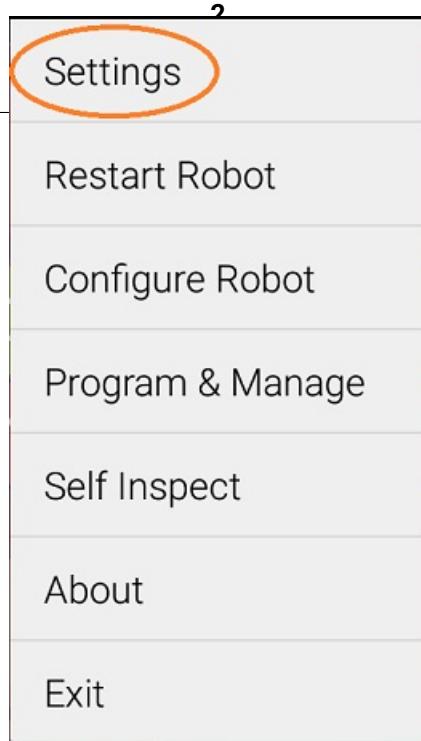
3. Make sure that your Expansion Hub is turned on and powered by a freshly charged 12V battery and that the Robot Controller phone is connected to the Expansion Hub through a USB connection. Note that the Robot Controller does **not** need to have the Expansion Hub included in an active configuration file in order for the update to work.



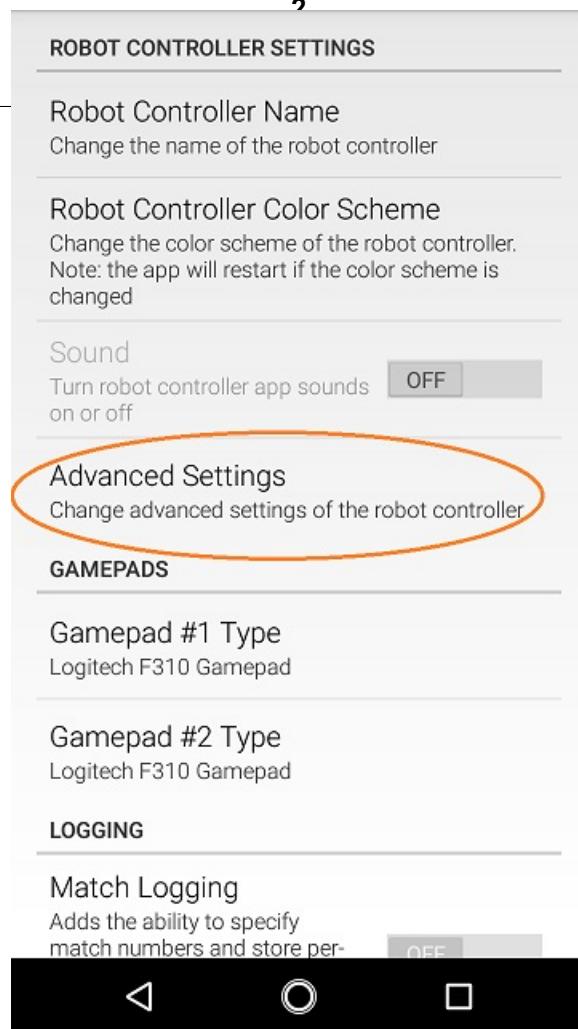
4. On the Driver Station, touch the three dots in the upper right hand corner to display a pop-up menu.



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.

2

### 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

---



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.

Expansion Hub firmware update file "REVHubFirmware\_1\_08\_02.bin" was found, as were the following Expansion Hubs:

Serial number: (embedded)  
Module address: 1  
Current Firmware: HW: 20, Maj: 1, Min: 7, Eng: 2

Click on the button below to update the firmware on each of these Expansion Hubs.

**WARNING:** While the firmware update is in progress, do not unplug hubs or restart the robot, or a hub may become permanently unusable.

**Update Expansion Hub Firmware**



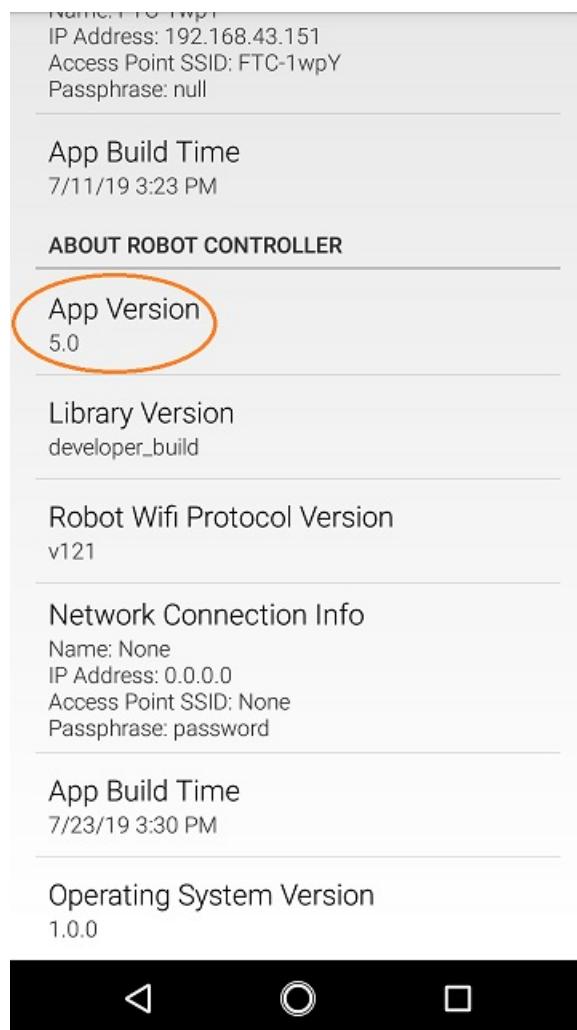
9. A progress bar will display while the firmware is being updated. Do not power off the Robot Controller/Expansion Hub during this process. The Driver Station will display a message when the update process is complete.

Firmware update of Expansion Hub (embedded) succeeded.

## 2.4.5 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 [FTC Robot Controller 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](#).

---

#### 2.4.6 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](#).

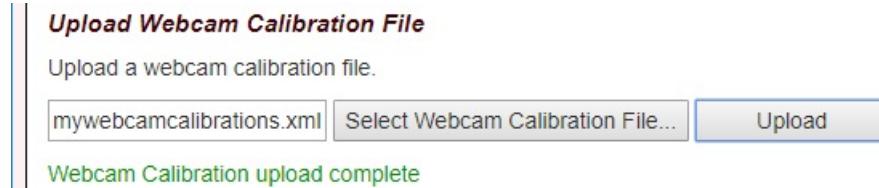
##### Uploading a Custom Webcam Calibration File Instructions

- 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.

- 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.



## Chapter 3

### AprilTag Programming

Topics for programming with AprilTags

#### 3.1 AprilTag Introduction

##### 3.1.1 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. 1: 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!

### 3.1.2 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**.



Fig. 2: AprilTags on Robots. Photo Credit: University of Michigan

AprilTag is a type of **visual fiducial**, or fiducial marker, containing information and designed for easy recognition.

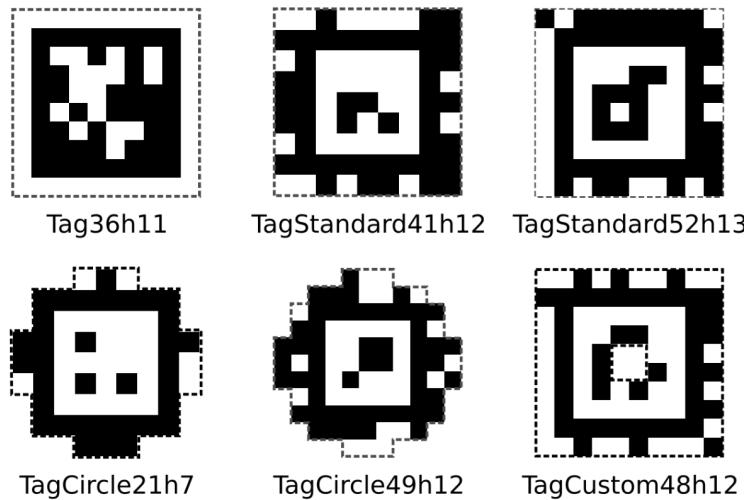


Fig. 3: A sample of different AprilTag families

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:

- 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**).

The above image shows a camera preview image, called LiveView, from a Robot Controller device (Control Hub or RC phone).

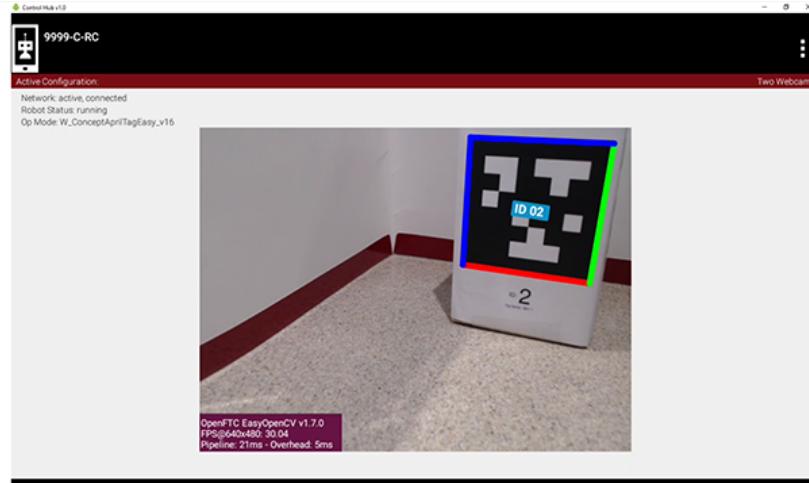


Fig. 4: Stream output showing the detected tag ID 02

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](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  $10 \times 10$  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.

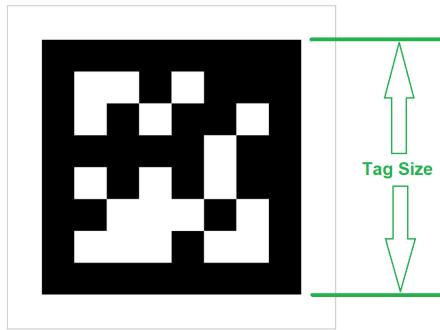


Fig. 5: Figure demonstrating the tag size measurement

The above image shows a complete AprilTag with outer white border. From the 36h11 family, its ID code is 42.

### 3.1.3 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).

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

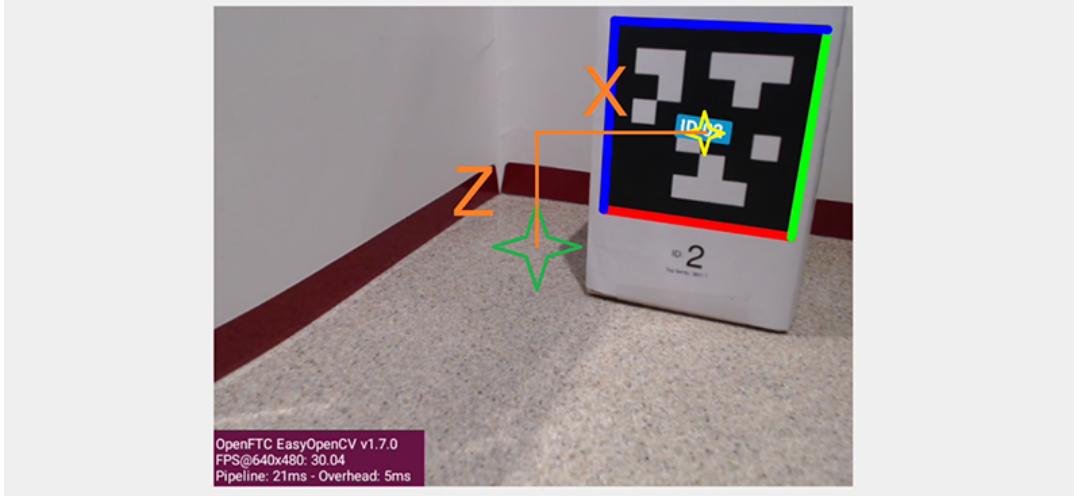


Fig. 6: LiveView Image with additional markings for explanation purposes

- 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.

### 3.1.4 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.

### 3.1.5 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:

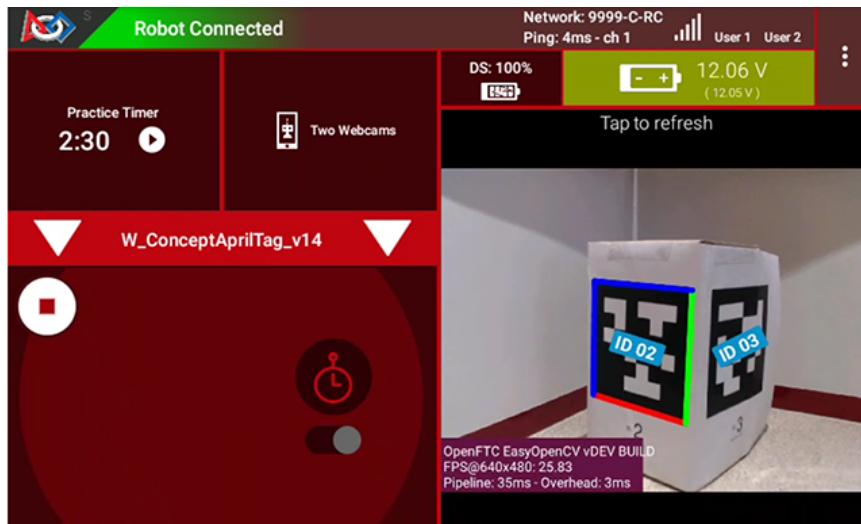


Fig. 7: Two AprilTags with different metadata being detected and annotations displayed

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:

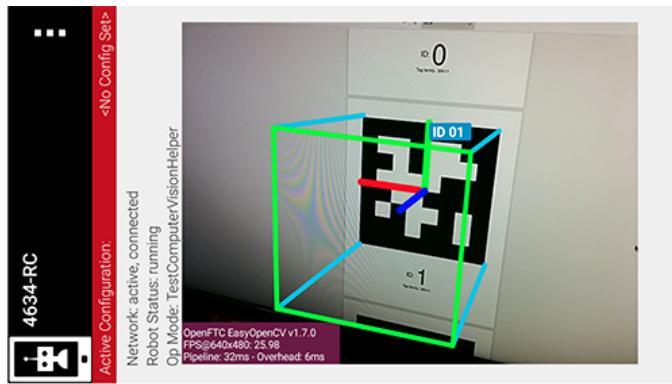


Fig. 8: LiveView with additional annotations enabled

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>

### 3.1.6 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:

- <https://ftc-docs.firstinspires.org/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.

### 3.1.7 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. [ConceptAprilTagOptimizeExposure.java](#).

The new SDK offers Java Sample OpMode

### 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();

    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:

- <https://ftc-docs.firstinspires.org/apriltag-detection-values>

### 3.1.8 Summary

[FTC Docs](#) [AprilTag](#) is a popular camera-based technology, using a scanned image similar to a QR Code. [FTC Programming Resources, 353](#)

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](mailto:westsiderobotics@verizon.net)

## 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!

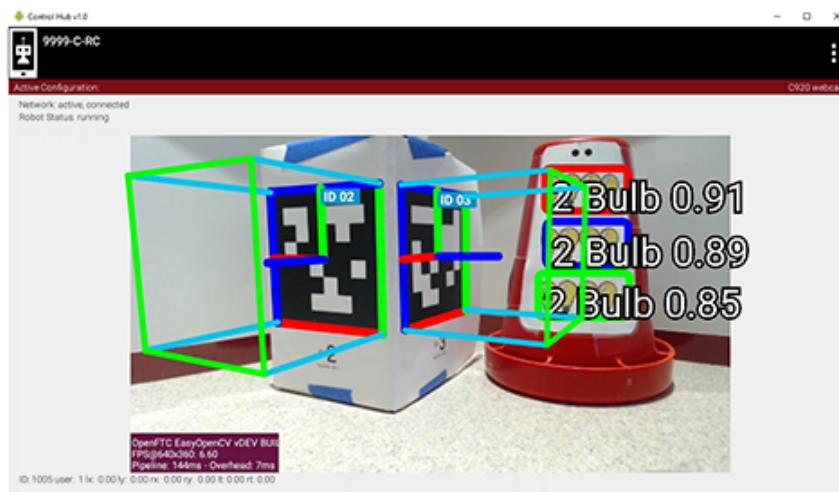


Fig. 9: Dual Preview with both AprilTags and TensorFlow

- **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.

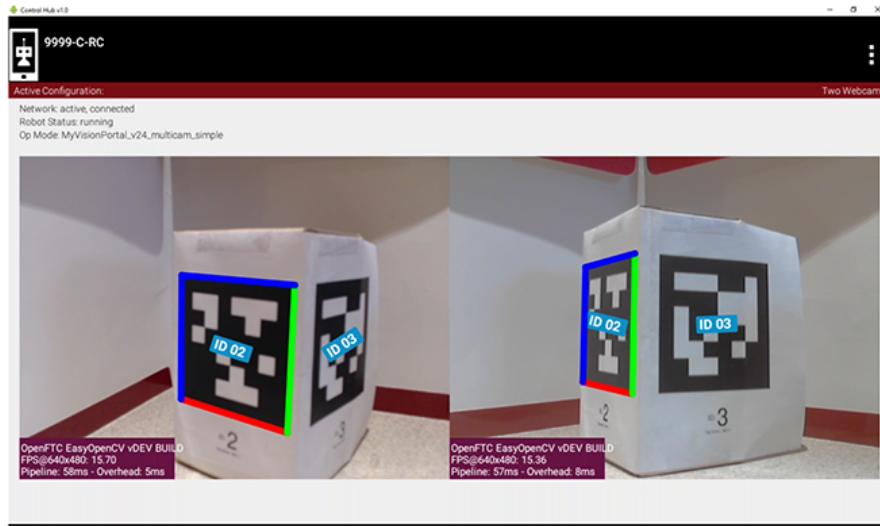


Fig. 10: Multiple Camera View

- **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. 11: 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.

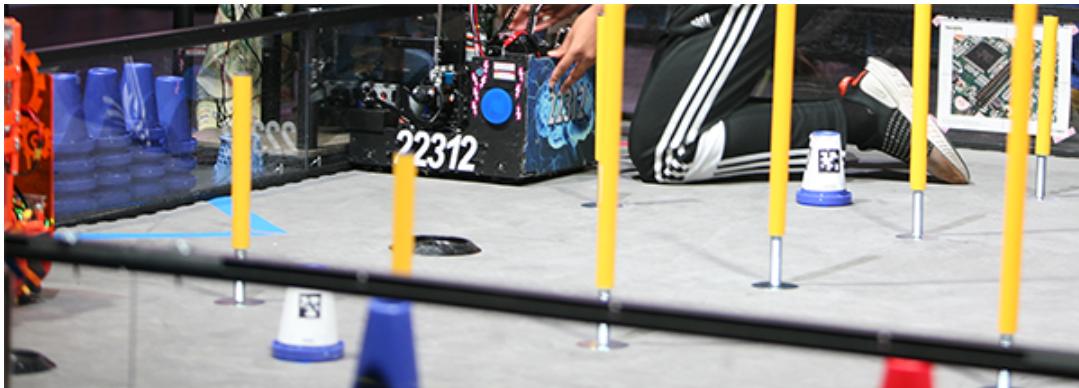


Fig. 12: Photo Credit: Mike Silversides

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.

### 3.2.1 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. 13: 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

### 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).
```

(continues on next page)

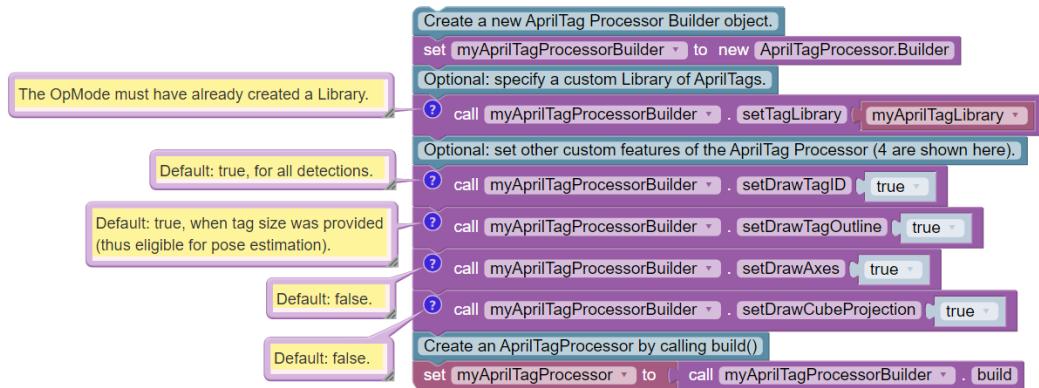


Fig. 14: AprilTag Processor Initialization with a Builder

(continued from previous page)

```

myAprilTagProcessorBuilder.setDrawAxes(true);           // Default: false.
myAprilTagProcessorBuilder.setDrawCubeProjection(true); // Default: false.

// 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

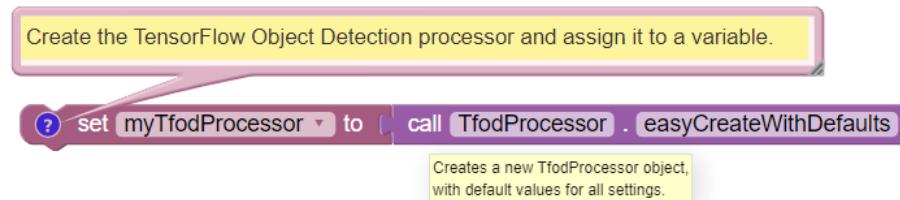


Fig. 15: Easy TensorFlow TFOD Processor Initialization without a Builder

### 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

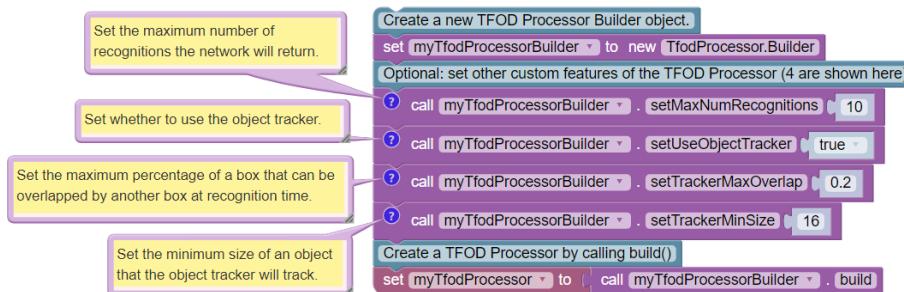


Fig. 16: TensorFlow TFOD Processor Initialization with a Builder

```

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 u
→return
myTfodProcessorBuilder.setUseObjectTracker(true); // Whether to use the object tracker
myTfodProcessorBuilder.setTrackerMaxOverlap((float) 0.2); // Max. % of box overlapped by another u
→box at recognition time
myTfodProcessorBuilder.setTrackerMinSize(16); // Min. size of object that the object tracker will u
→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:

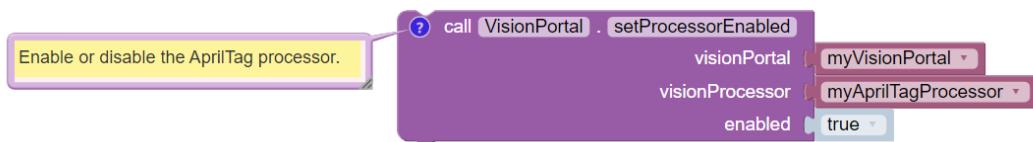


Fig. 17: Enable or Disable AprilTag Processor

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*

### 3.2.2 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.

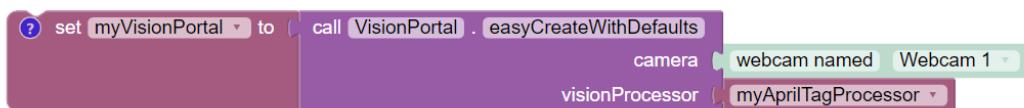
Steps 1, 2 and 3 are typically performed in the OpMode's INIT section, before the `waitForStart()` method or Block.

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  $3 \times 3 = 9$  total choices, all “Easy”.

```
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:

```
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

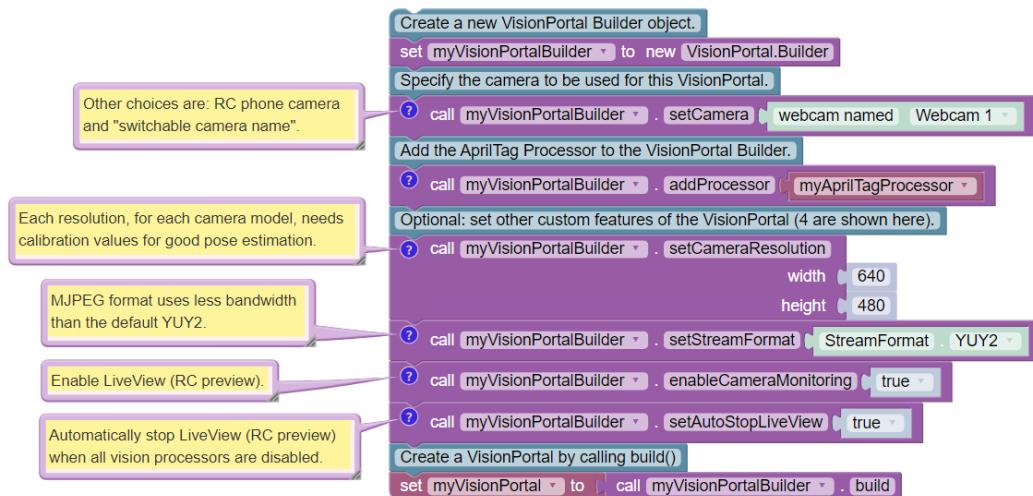


Fig. 18: VisionPortal Initialization with a Builder

### 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.
```

(continues on next page)

```

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:

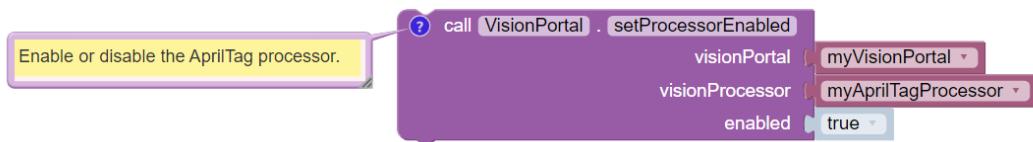


Fig. 19: VisionPortal Enabling/Disabling

**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](mailto:westsiderobotics@verizon.net)*

### 3.2.3 VisionPortal Previews

#### Introduction

Managing AprilTag and TFOD performance is greatly enhanced with visual feedback of the camera's view.

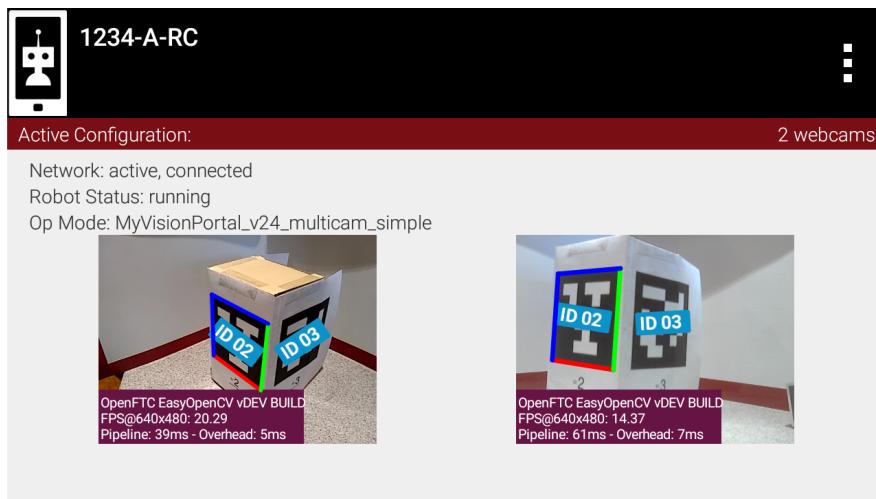


Fig. 20: LiveView demonstrating multiple camera support

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 **VisionPort 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:

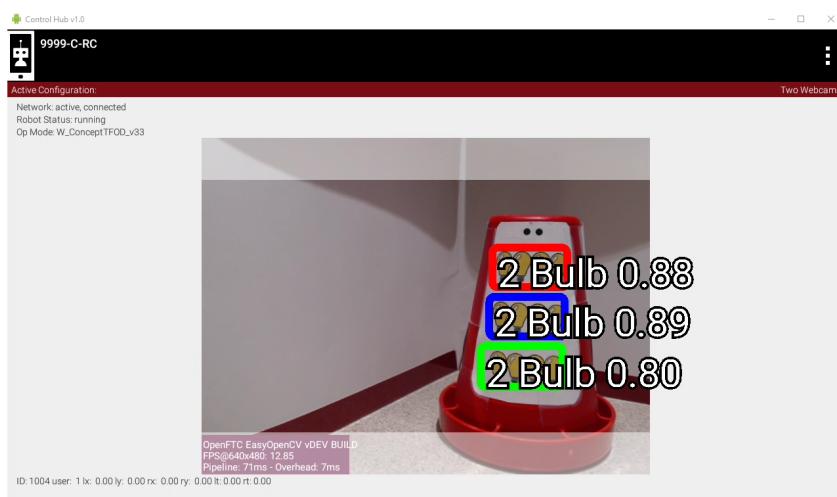


Fig. 21: LiveView demonstrating Grey Bands from Aspect Ratio mismatch

The **greyed bands** at top and bottom are from the **mismatch of aspect ratios**:

## FTC Docs

- 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**.

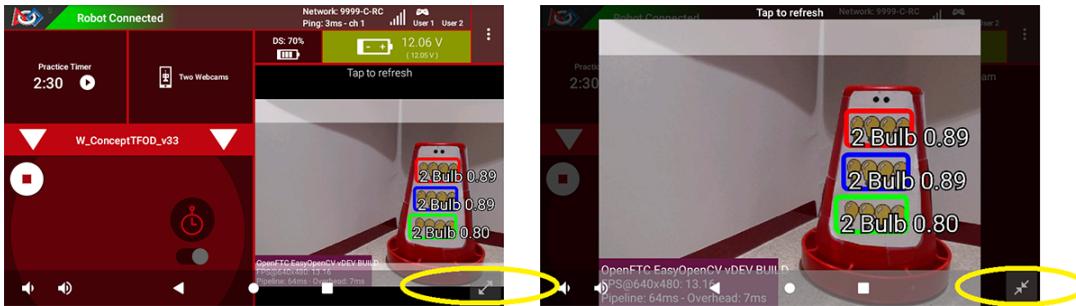


Fig. 22: Camera Stream preview enlargement buttons

**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](mailto:westsiderobotics@verizon.net)

### 3.2.4 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

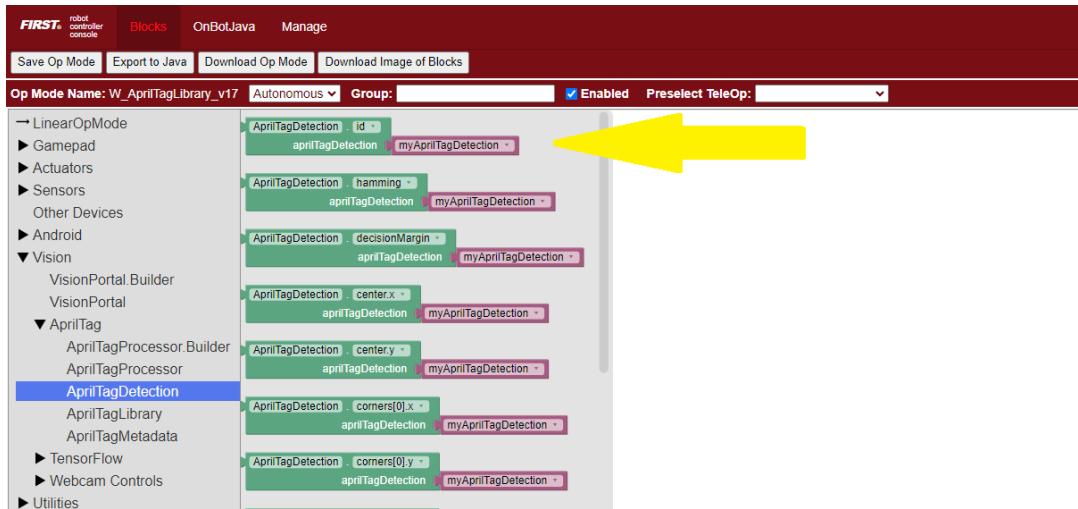


Fig. 23: Highlighting blocks for getting AprilTag ID Code

### 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

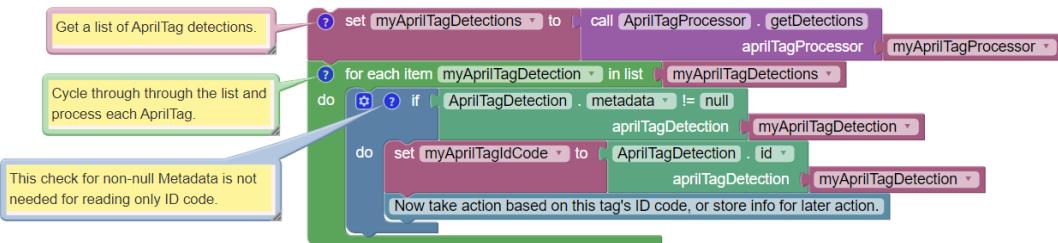


Fig. 24: All AprilTag Detections being read in a FOR Loop

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, u
    ↪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](mailto:westsiderobotics@verizon.net)*

### 3.2.5 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. 25: 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](mailto:westsiderobotics@verizon.net)

### 3.2.6 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).

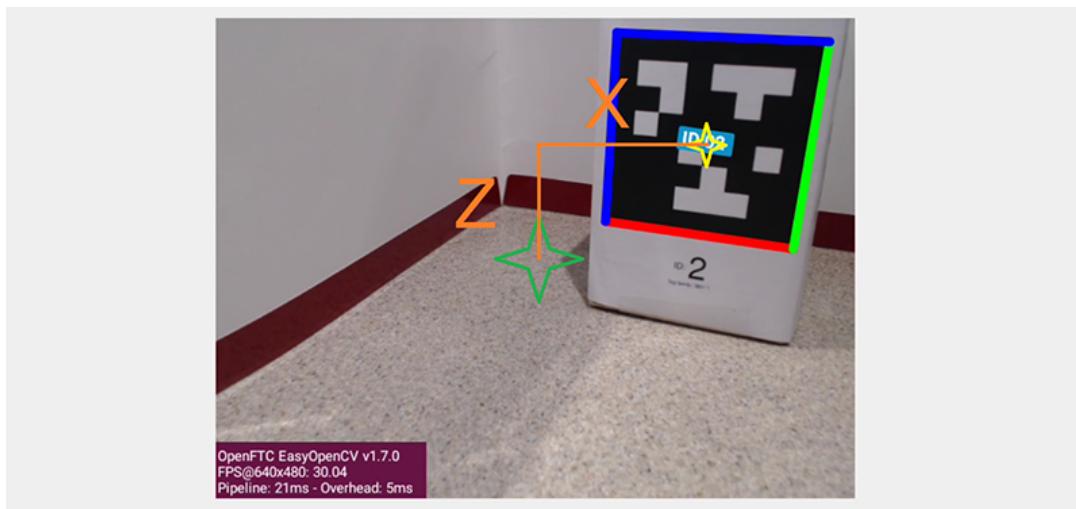


Fig. 26: Image depicting LiveView offsets

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:

- <https://ftc-docs.firstinspires.org/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](mailto:westsiderobotics@verizon.net)*

## 3.2.7 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:

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
  - (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).



Fig. 27: Warning of no camera calibration provided

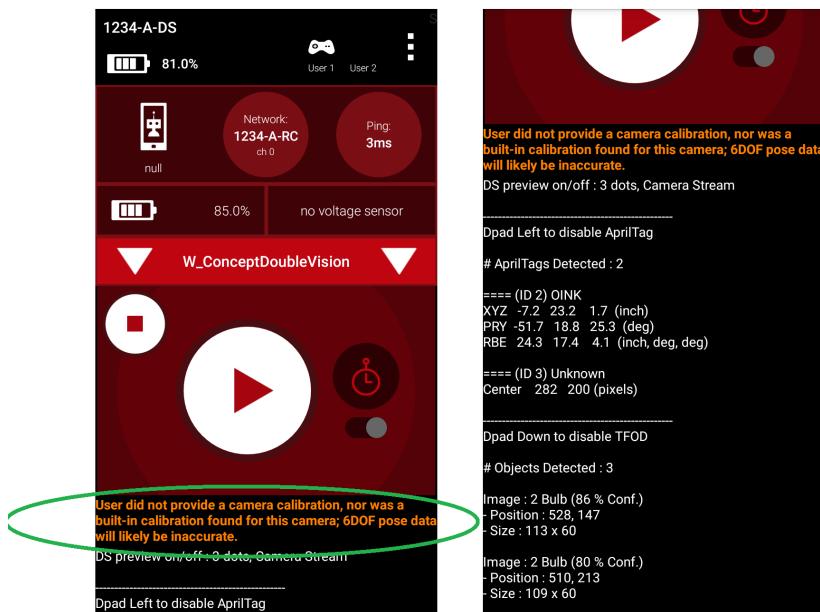


Fig. 28: Right-hand image shows that the warning still allows detections.

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:

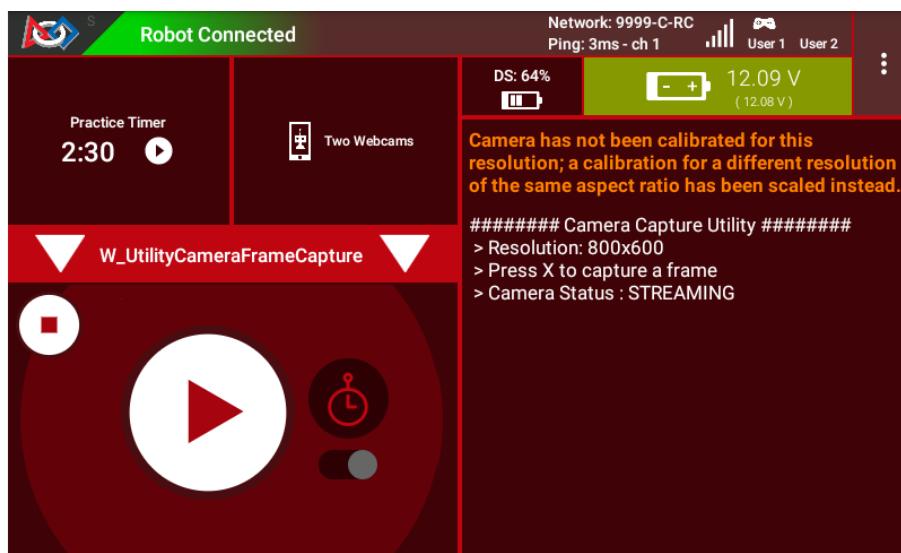


Fig. 29: Warning about no calibration at this resolution

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](mailto:westsiderobotics@verizon.net)*

### 3.2.8 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

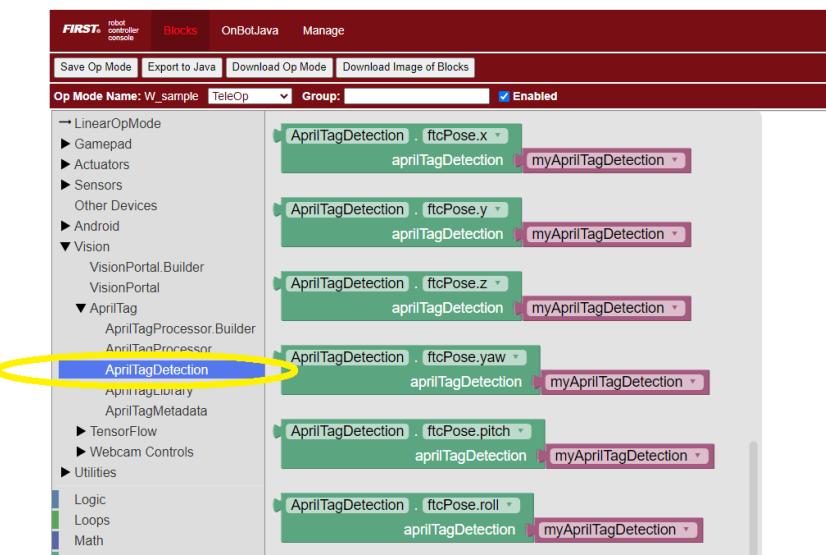


Fig. 30: AprilTag Pose Blocks

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:

- **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:

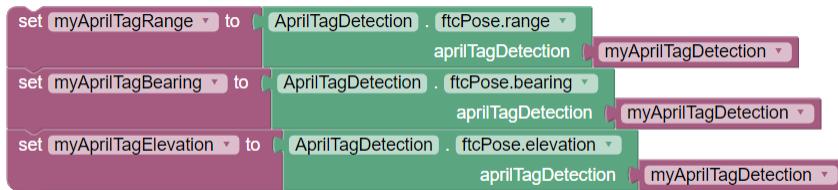


Fig. 31: AprilTag Range, Bearing, Elevation Blocks

## 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:

- <https://ftc-docs.firstinspires.org/apriltag-detection-values>

*Questions, comments and corrections to westsiderobotics@verizon.net*

## 3.2.9 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

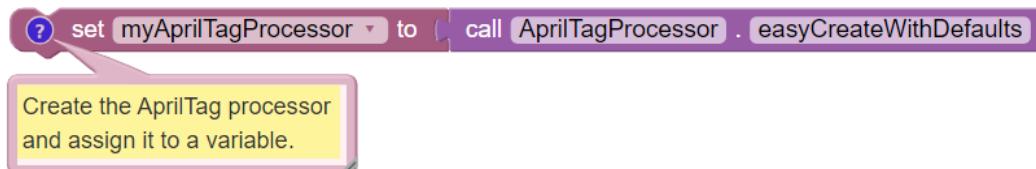


Fig. 32: Simple AprilTag Processor

### 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.

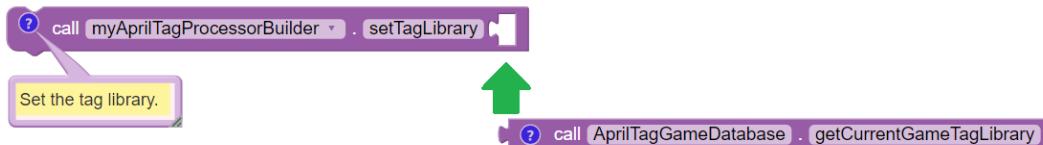


Fig. 33: Setting Current Game Tag Library

**Around this** (before and after), place one Block to **create** the Processor Builder, and another Block to **finalize** the process with `.build()`.

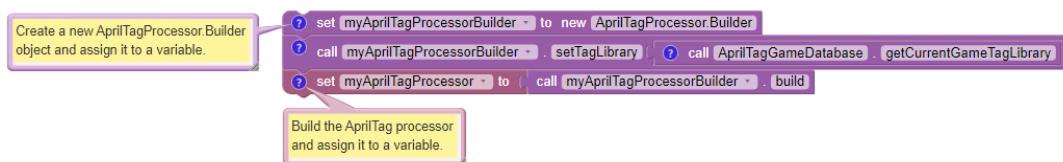


Fig. 34: Completing Builder

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.

```

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.

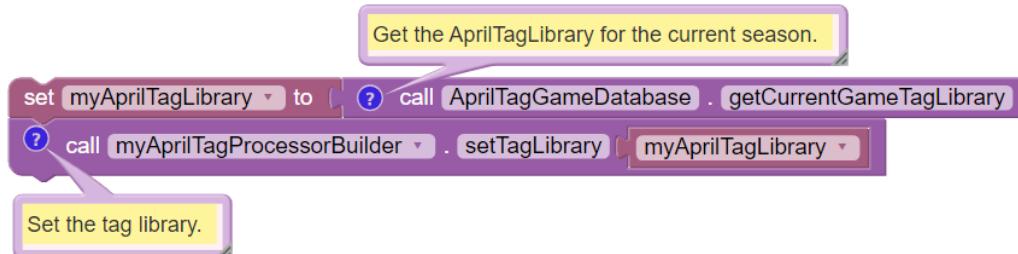


Fig. 35: Set the Tag Library

As before, **around this** (before and after), place one Block to **create** the Processor Builder, and another Block to **finalize** the process with `.build()`.

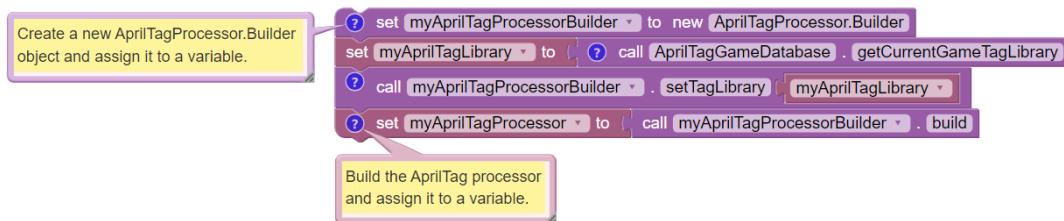


Fig. 36: Build the AprilTag Processor

```

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. 37: 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.

The final result is the same as the previous examples, but now you see how to use a Library Builder.

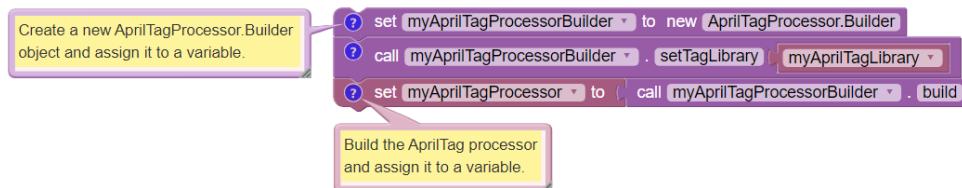


Fig. 38: Build the Tag Processor

## 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());

// 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.

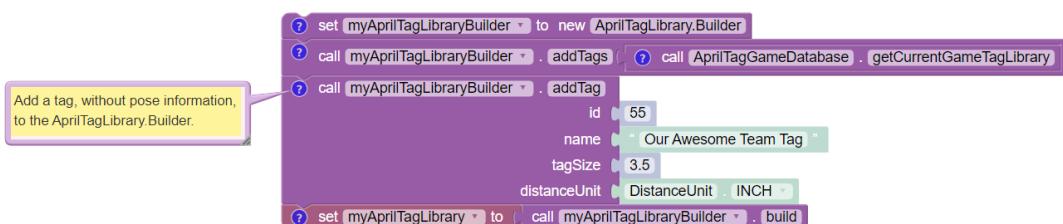


Fig. 39: Add custom tags to Tag Library

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);

// 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.

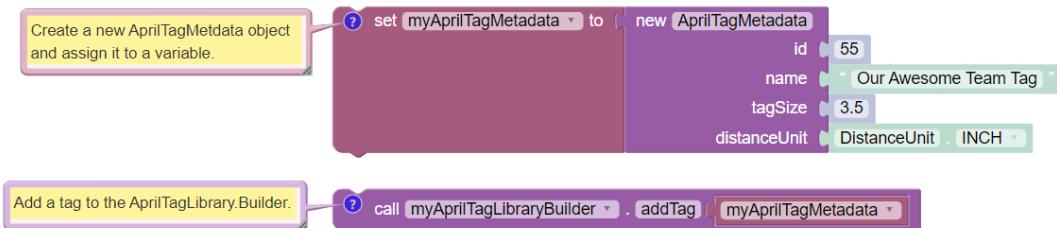


Fig. 40: Setting Metadata with Variable

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.

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);

// 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](mailto:westsiderobotics@verizon.net)*

## 3.2.10 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.

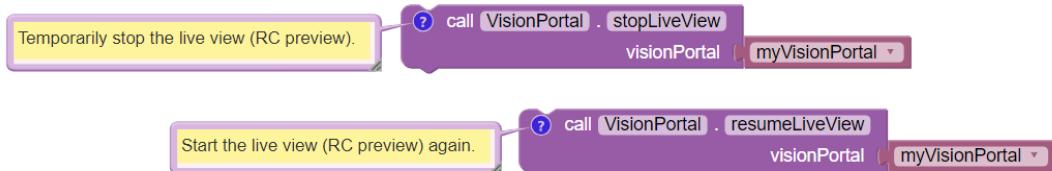


Fig. 41: Examples of Toggling LiveView

```
// 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.

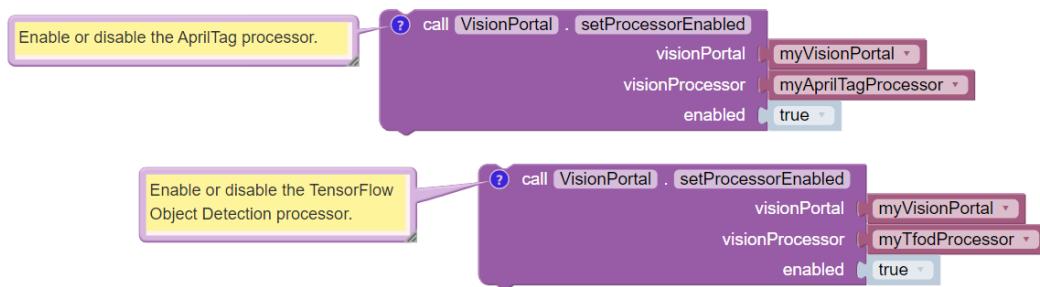


Fig. 42: Examples of Toggling Processors

**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.

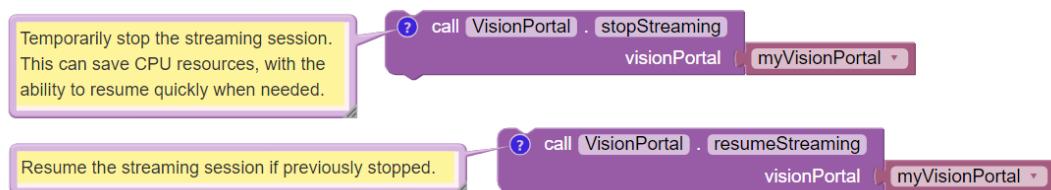


Fig. 43: Examples of Toggling Camera Stream

```
// 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.

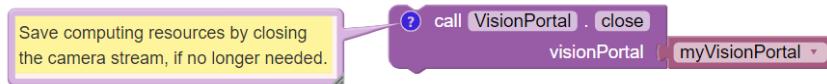


Fig. 44: Close VisionPortal Example

### 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](mailto:westsiderobotics@verizon.net)*

### 3.2.11 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.

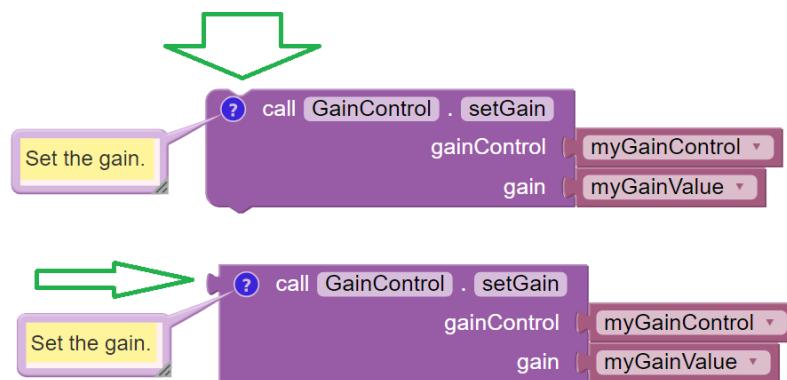


Fig. 45: Examples of Setter Blocks with toggable return values

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):

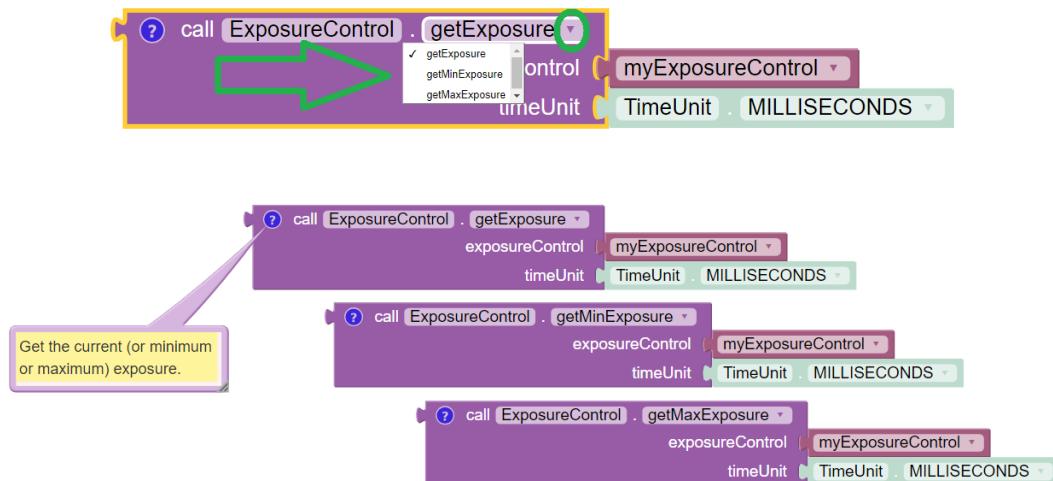


Fig. 46: Examples of Exposure Blocks with pull-down lists

This is used six places in the Webcam Controls section.

## Pan-Tilt Holder

See this Block with the NEW operator (green oval):



Fig. 47: 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.

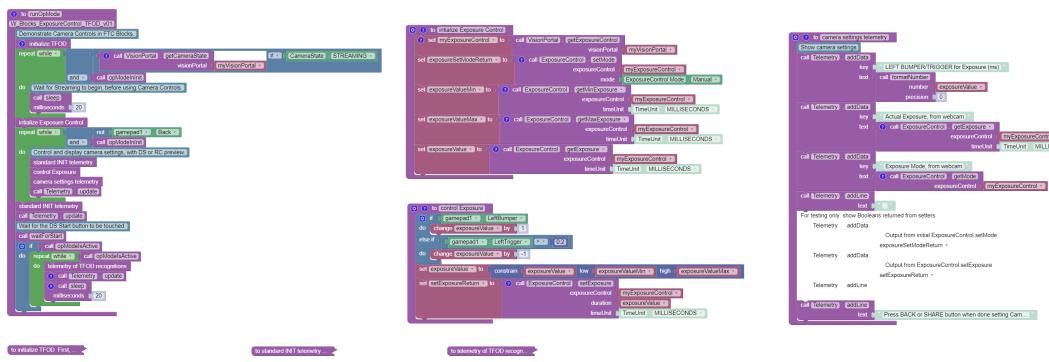


Fig. 48: Blocks Exposure OpMode Example

Questions, comments and corrections to [westsiderobotics@verizon.net](mailto:westsiderobotics@verizon.net)

## 3.2.12 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.

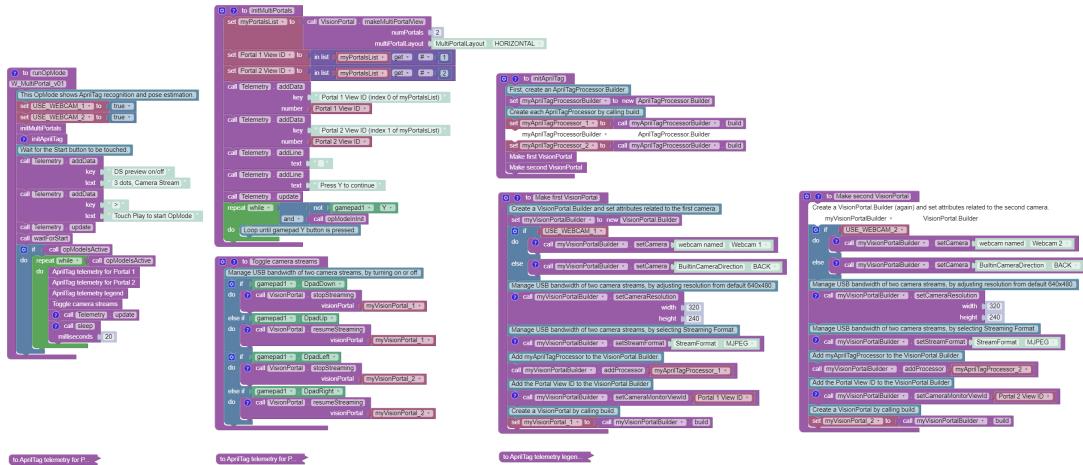


Fig. 49: Example Blocks Multiportal OpMode

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)

The dual RC previews can be displayed as VERTICAL, or side-by-side with the enum HORIZONTAL:

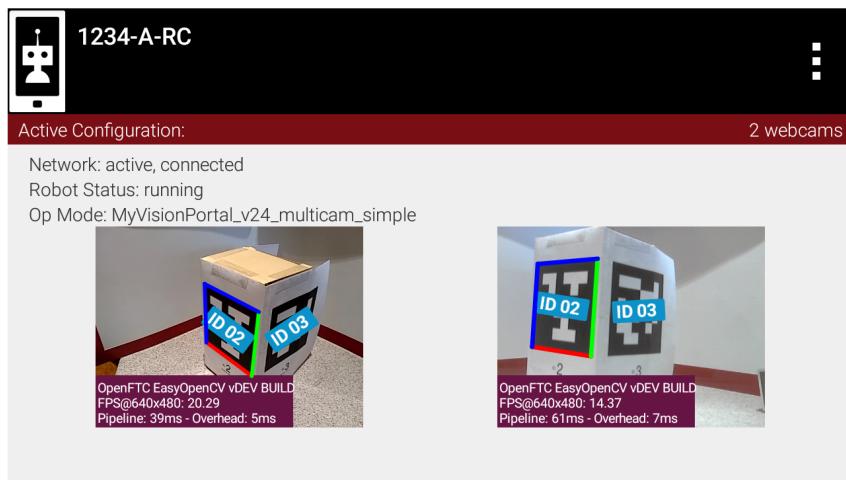


Fig. 50: Dual RC Previews

The DS Camera Stream preview can display only one camera's view (a known characteristic).

## 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 `scrpy`.

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](mailto:westsiderobotics@verizon.net)

### 3.2.13 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**:

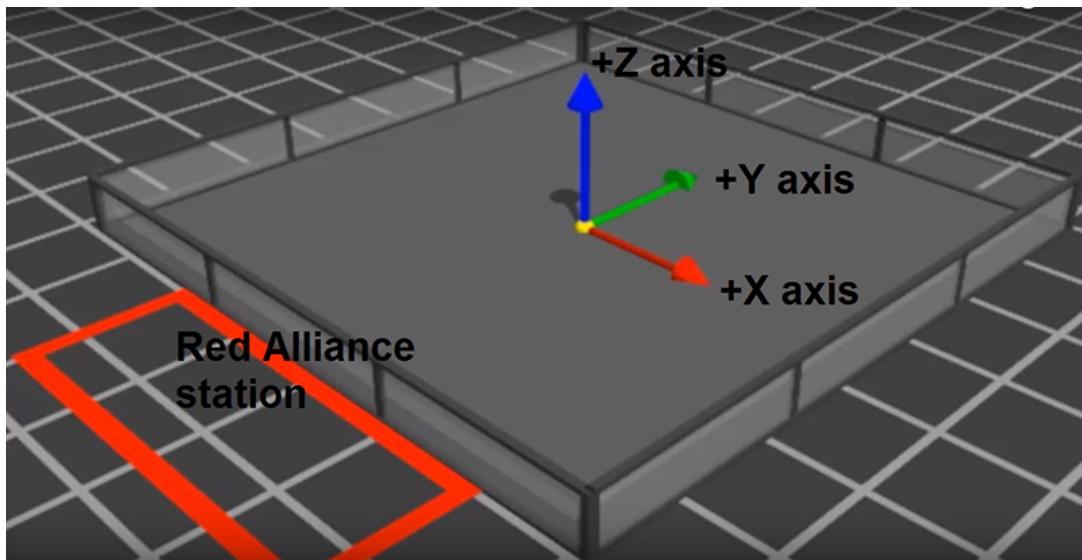


Fig. 51: Image Credit: Phil Malone

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;
    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:

- <https://ftc-docs.firstinspires.org/apriltag-detection-values>

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.

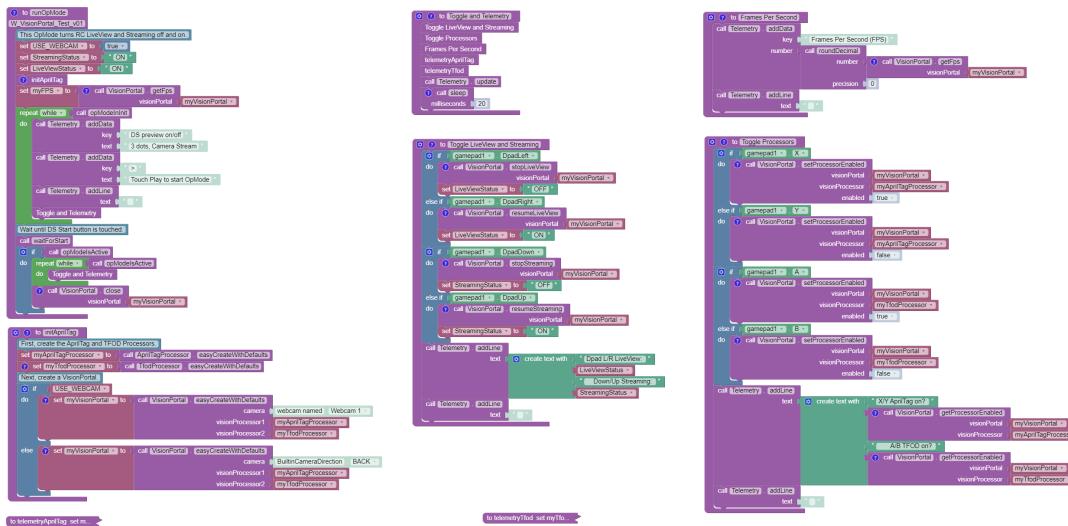


Fig. 52: VisionPortal Example OpMode

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:

- 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.

Questions, comments and corrections to [westsiderobotics@verizon.net](mailto: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](mailto:westsiderobotics@verizon.net)

### 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*.

#### 3.3.1 Logitech C270



Fig. 53: 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](#).

### 3.3.2 Logitech C310



Fig. 54: 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](#).

### 3.3.3 Logitech C920



Fig. 55: 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 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](#).

### 3.3.4 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.



Fig. 56: Microsoft LifeCam HD-3000 v1/v2

**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](#).

### 3.3.5 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.

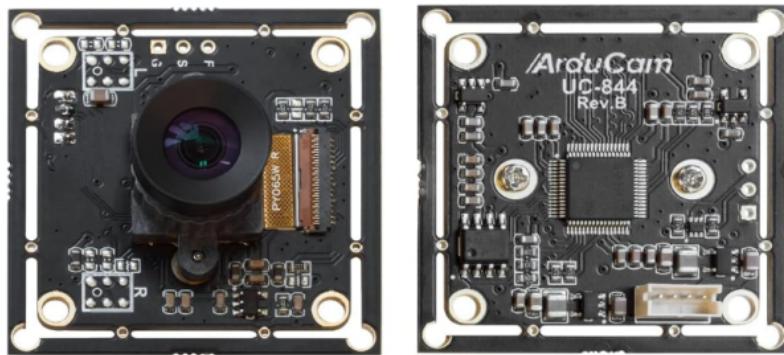


Fig. 57: Arducam GS 120 Camera

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/) <<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.

### 3.3.6 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 1: Cameras and Supported Resolutions

Camera	Features	Resolutions with Built-In Calibrations	Resolutions without Calibrations
<i>Logitech C270</i>	60 DegFOV, 30fps@720p	640x480	160x120, 176x144, 320x176, 320x240, 352x288, 432x240, 544x288, 640x360, 752x416, 800x448, 800x600, 864x480, 960x544, 960x720, 1024x576, 1184x656, 1280x720
<i>Logitech C310</i>	60 DegFOV, 30fps@720p	640x480, 640x360	All other resolutions
<i>Logitech C920</i>	78 DegFOV, 30fps@1080p	640x480, 640x360, 800x448, 800x600, 864x480, 1920x1080	160x90, 160x120, 176x144, 320x180, 320x240, 352x288, 432x240, 960x720, 1024x576, 1280x720, 1600x896, 2304x1296, 2304x1536
<i>Microsoft LifeCam HD-3000 v1/v2</i>	68.5 DegFOV, 30fps@720p	640x480	All other resolutions
<i>Arducam Global Shutter 120 FPS</i>	70 DegFOV, 120fps@1280x800 MJPG, Monochrome	No Built-In Calibrations	MJPEG: 320x240, 640x480, 800x600, 1280x720, 1280x800; YUY2: 1280x720, 1280x800
<i>Kayetton Global Shutter (Other Global Shutter Cameras)</i>	70 DegFOV, 120fps@720p MJPG, Monochrome	No Built-In Calibrations	All resolutions
<i>ELP Global Shutter (Other Global Shutter Cameras)</i>	70 DegFOV, 90fps@1920x1200 MJPG, Monochrome	No Built-In Calibrations	All resolutions

Questions, comments and corrections to [westsiderobotics@verizon.net](mailto:westsiderobotics@verizon.net)

## 3.4 Understanding AprilTag Detection Values

Last Updated: 7/05/2023

### 3.4.1 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.

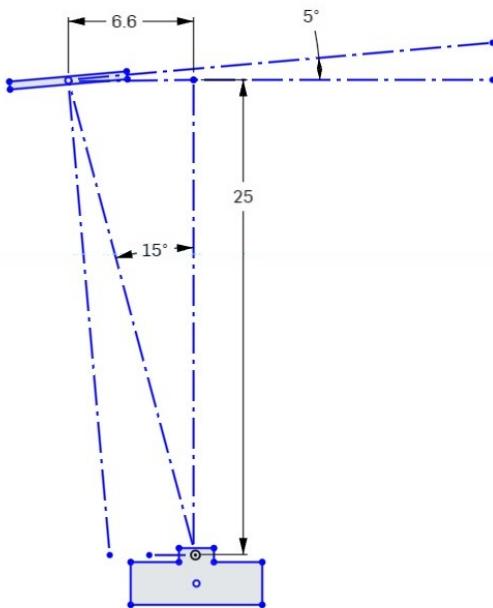


Fig. 58: Figure 1: Top view of AprilTag scenario

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).

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.

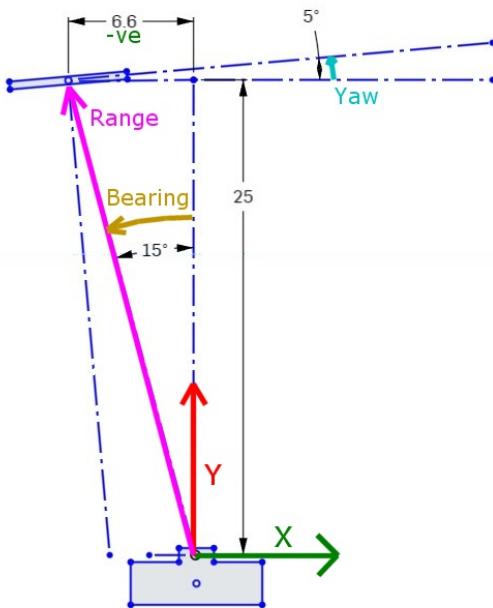


Fig. 59: Figure 2: Measured values associated with scenario

**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.

### 3.4.2 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 tag 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.

**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.

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.

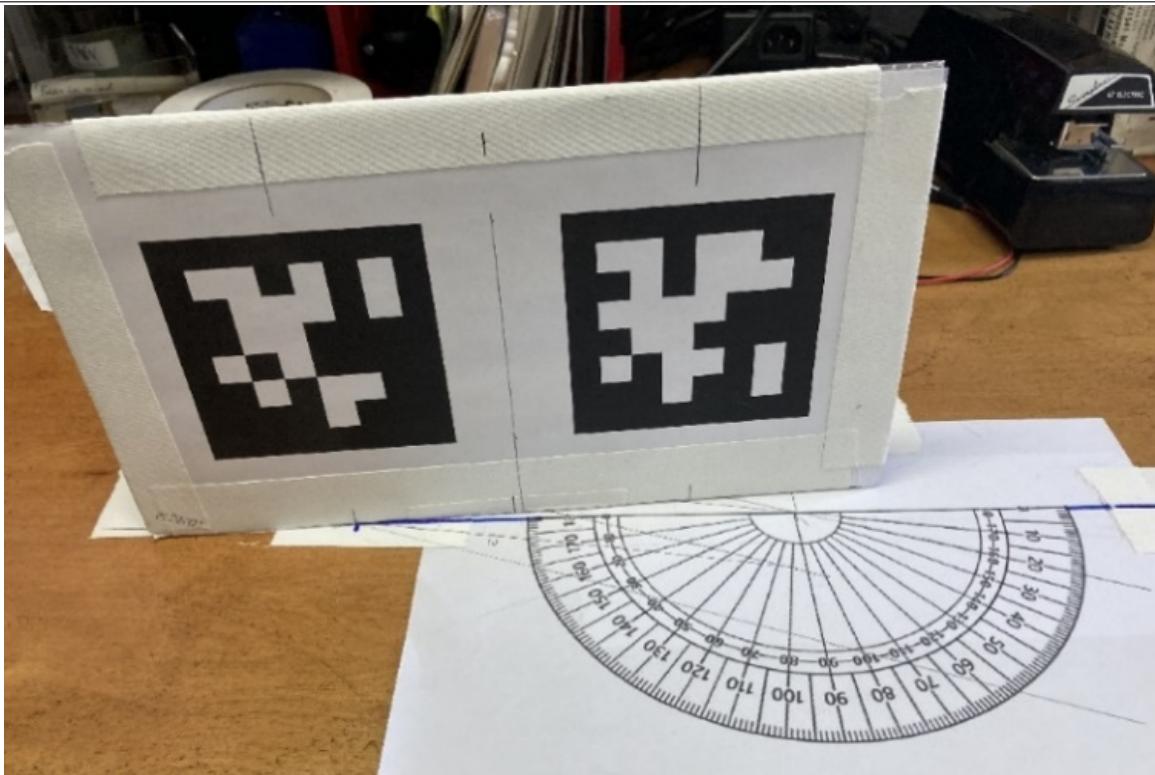


Fig. 60: Figure 3: Sample Tag setup for testing

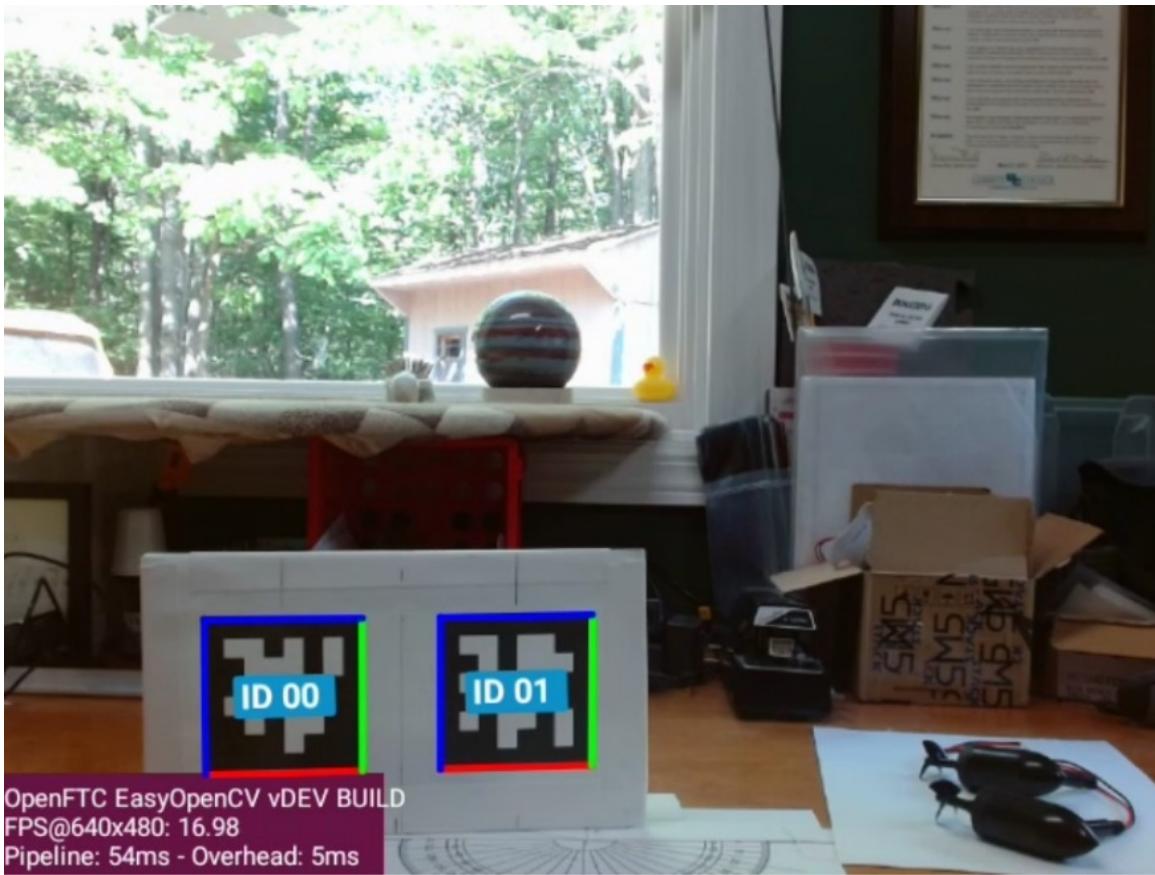


Fig. 61: Figure 4: Camera Preview showing two detected AprilTags

## Camera Ready

```
==== (ID 0) Nemo
XYZ -6.6 24.9 -5.7 (inch)
PRY 2.0 0.1 4.9 (deg)
RBE 25.7 14.8 -12.8 (inch, deg, deg)
```

```
==== (ID 1) Jonah
XYZ -1.5 25.5 -5.7 (inch)
PRY 0.7 -0.0 5.0 (deg)
RBE 25.6 3.3 -12.6 (inch, deg, deg)
```

### key:

**XYZ** = X (Right), Y (Forward), Z (Up) dist.  
**PRY** = Pitch, Roll & Yaw (XYZ Rotation)  
**RBE** = Range, Bearing & Elevation

Fig. 62: Figure 5: Values displayed by AprilTag OpMode

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.

### 3.4.3 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.

## 3.5 FIRST Tech Challenge AprilTag Testing Samples

### 3.5.1 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.

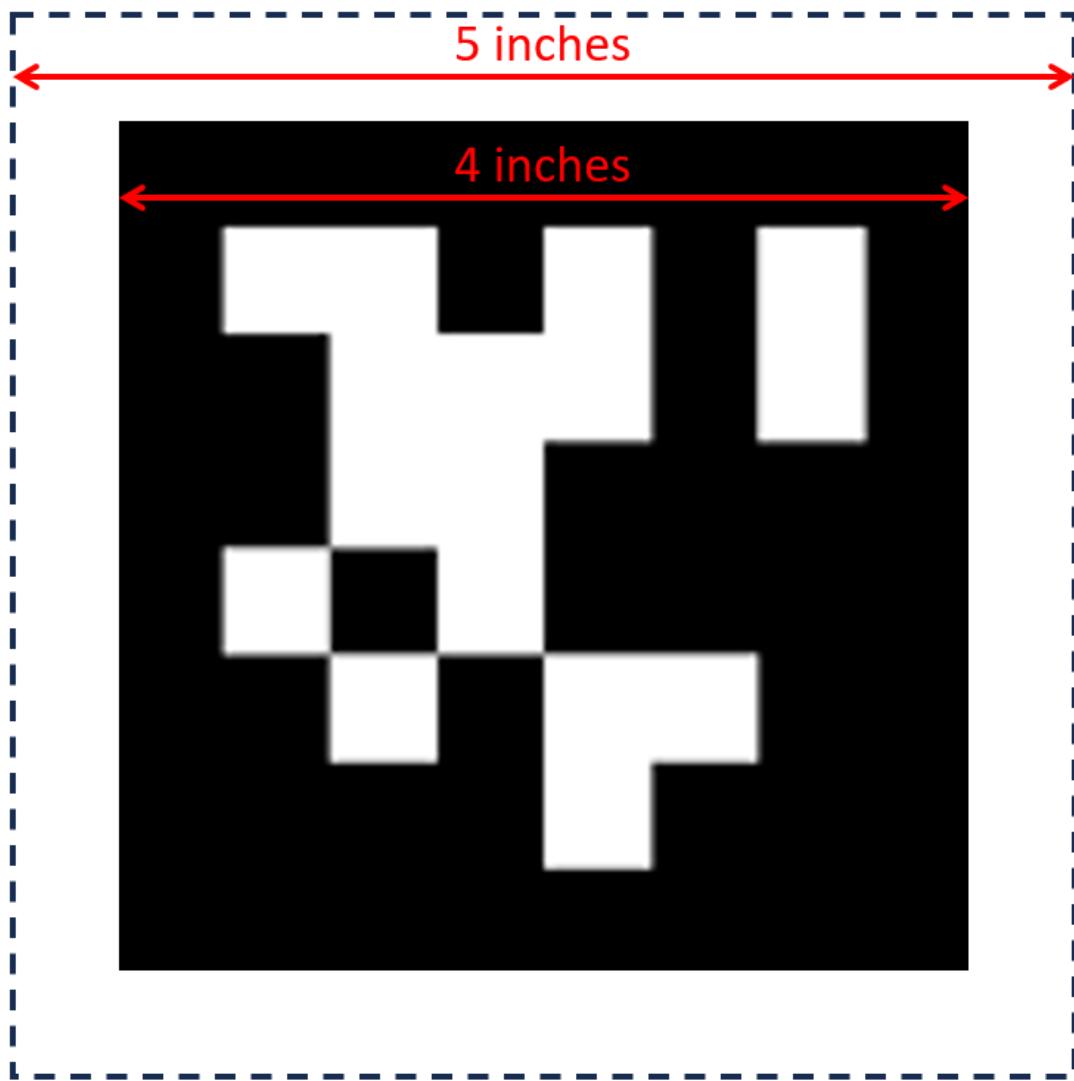


Fig. 63: Example sizing for 4-inch AprilTag

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 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.

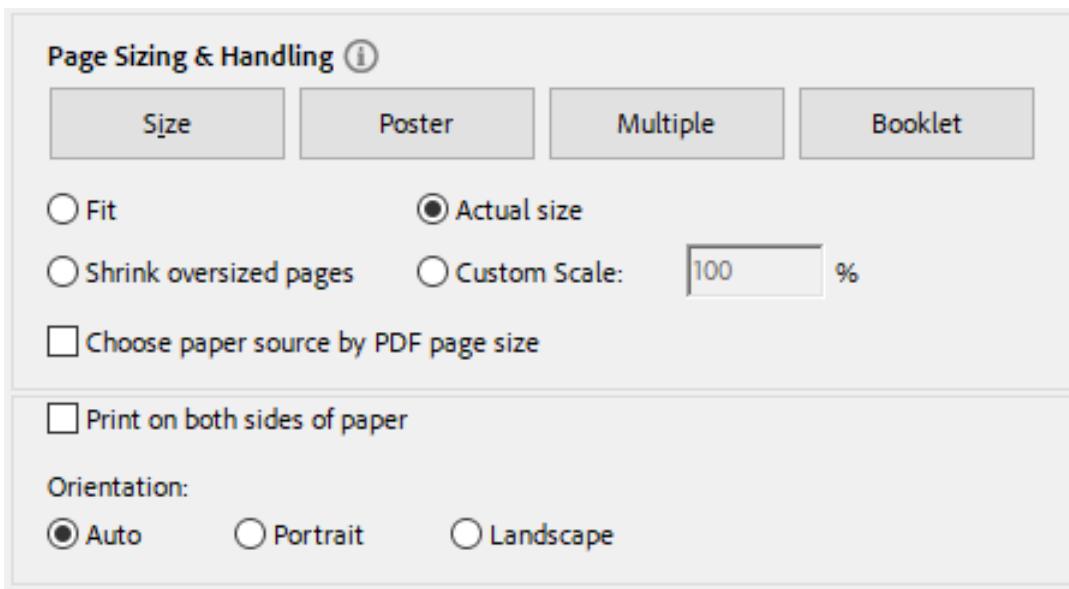


Fig. 64: Example printing settings for printing PDF at Actual Size

For more in-depth information about AprilTag detection values, and better understanding what they mean, please visit the following website:

- <https://ftc-docs.firstinspires.org/apriltag-detection-values>
- Download and print the official PDF

### 3.5.2 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.

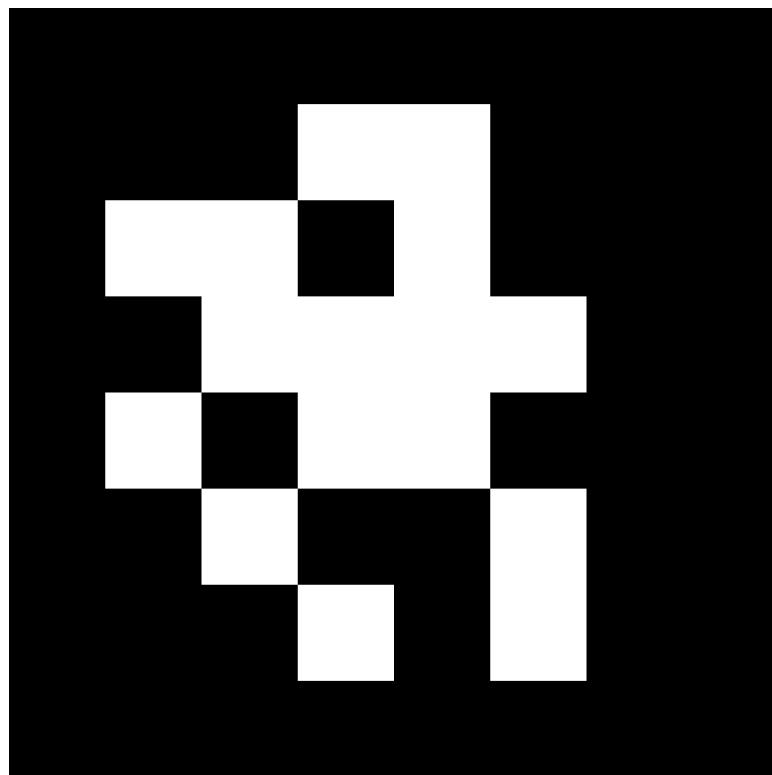


Fig. 65: Tag 583, "Nemo"

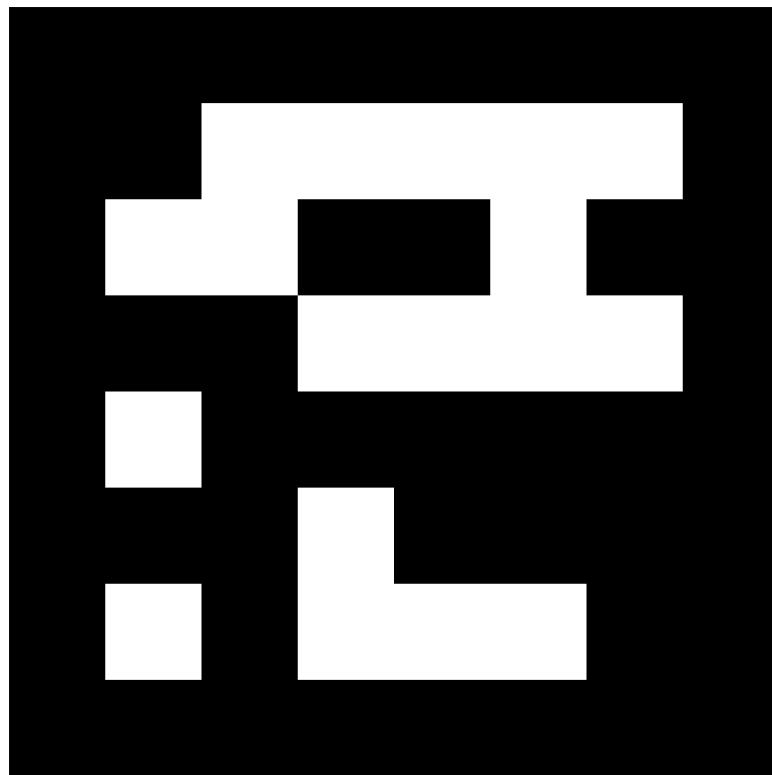


Fig. 66: Tag 584, "Jonah"

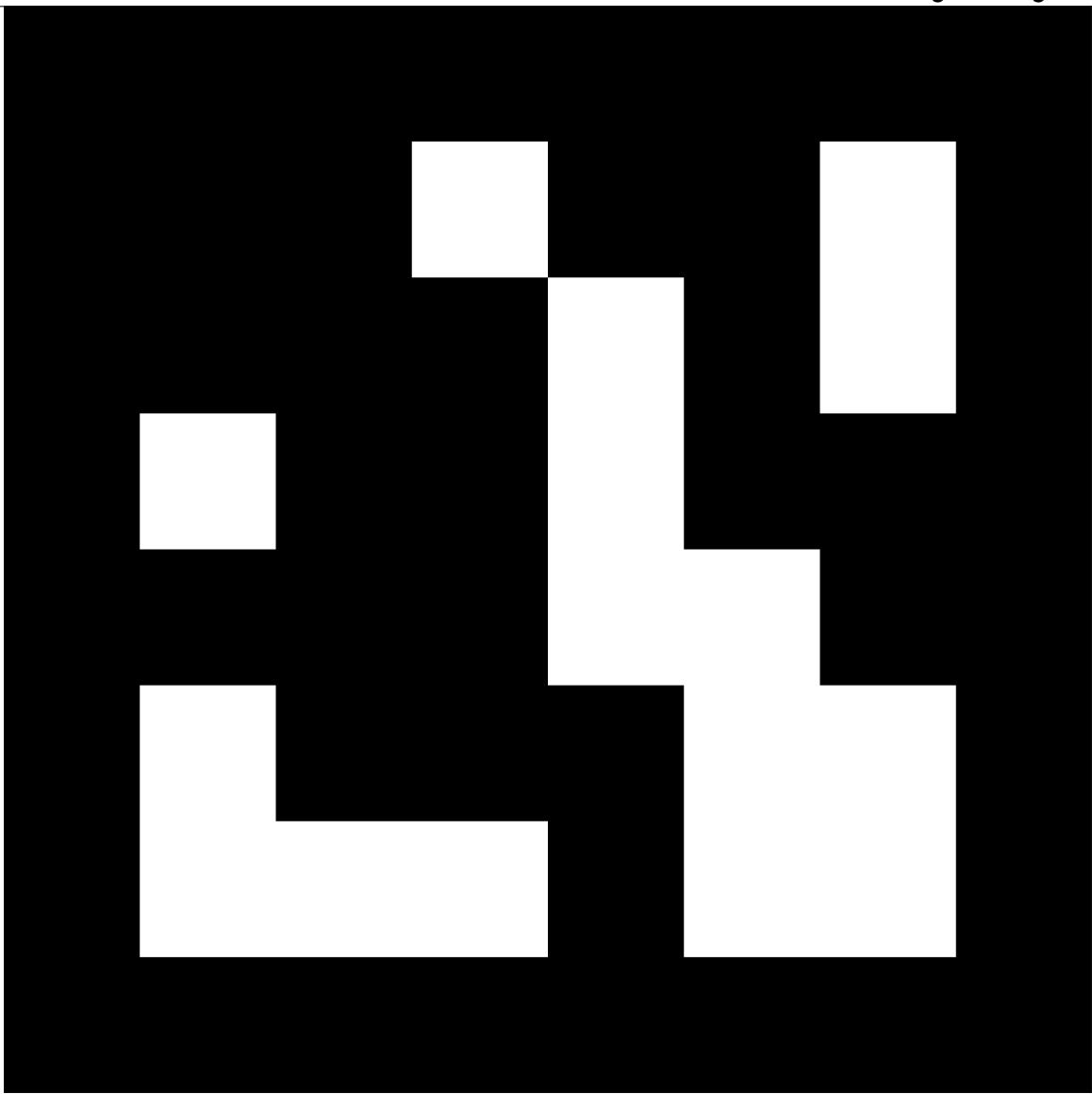


Fig. 67: Tag 585, "Cousteau"

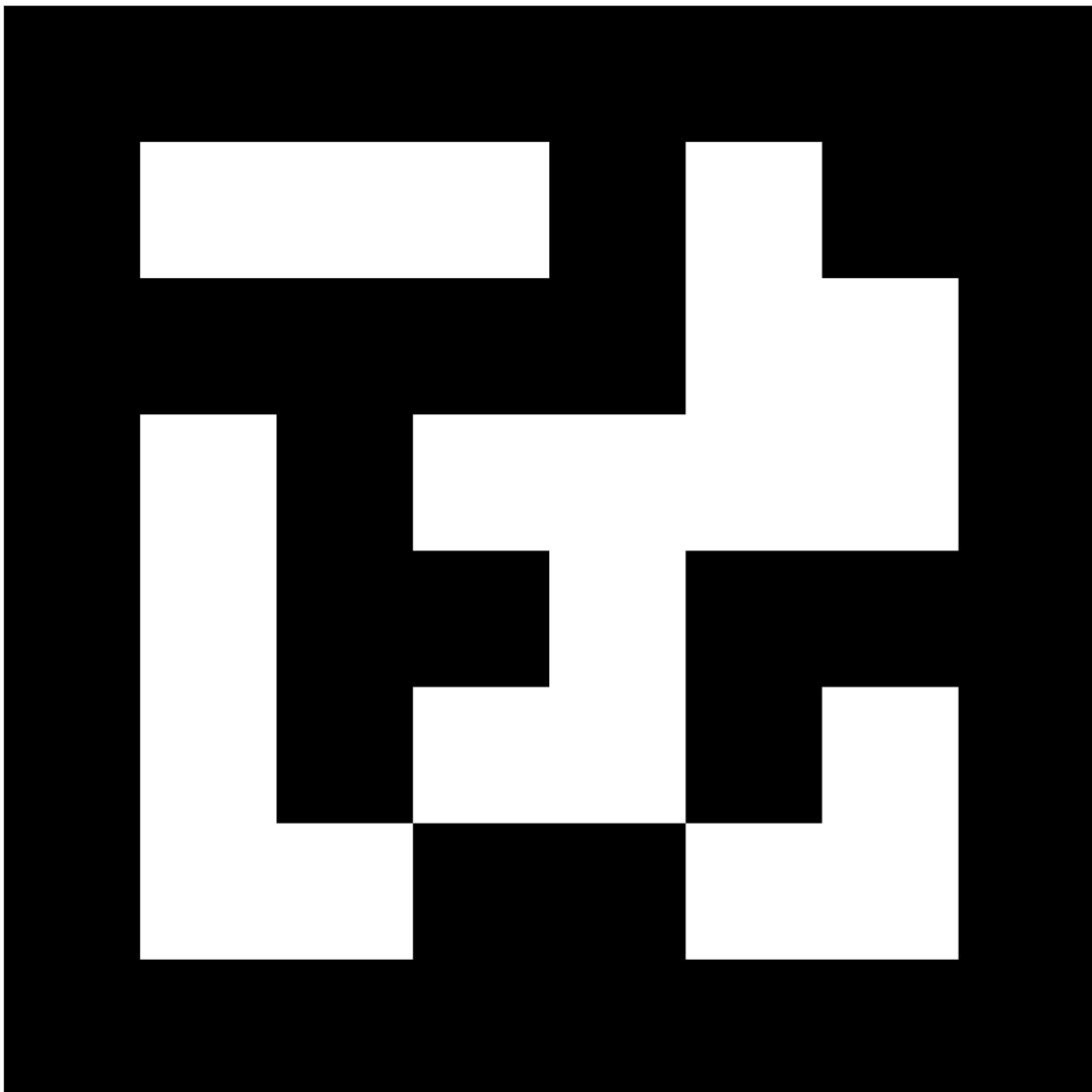


Fig. 68: Tag 586, "Ariel"

## Chapter 4

### TensorFlow Programming

Topics for programming with TensorFlow Object Detection (TFOD)

#### 4.1 TensorFlow for CENTERSTAGE presented by RTX

##### 4.1.1 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**.

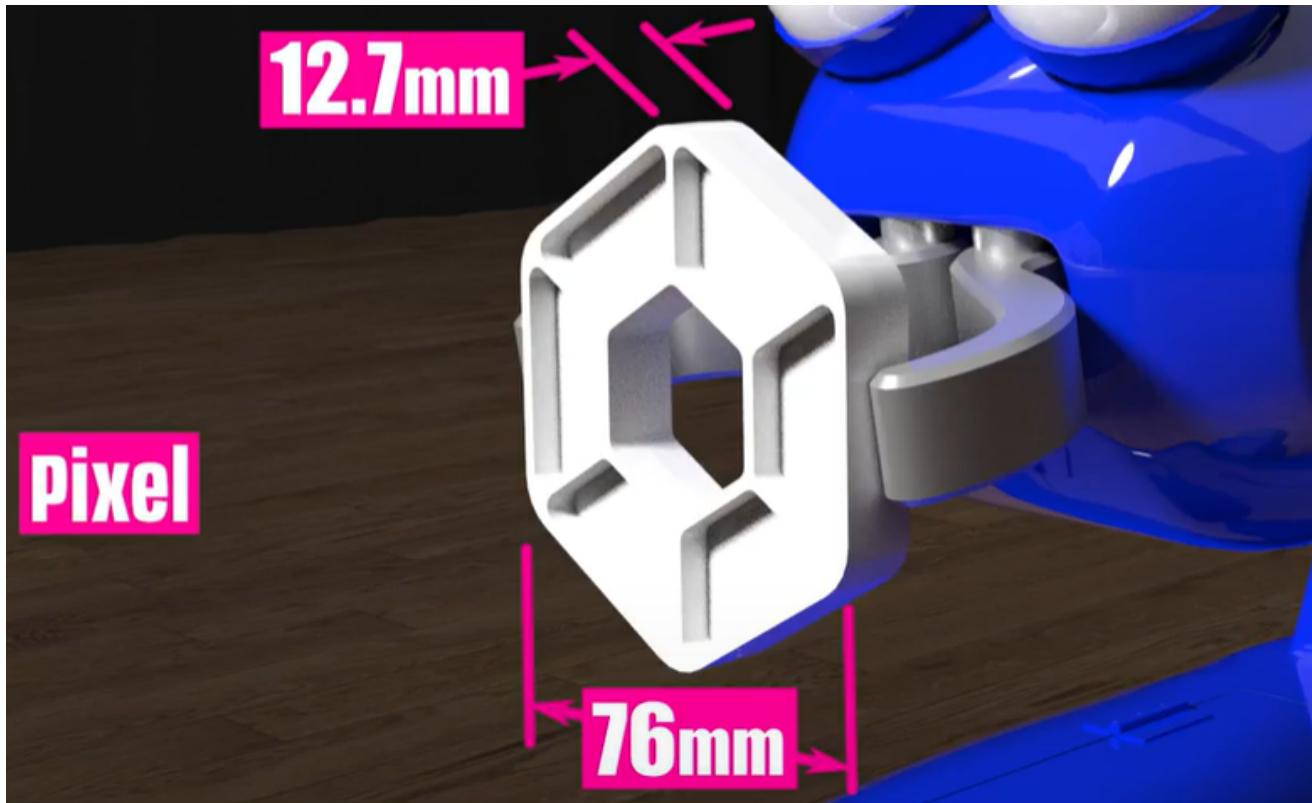


Fig. 1: This season's TFOD model can recognize a white Pixel

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).

#### 4.1.2 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.

#### 4.1.3 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.

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.

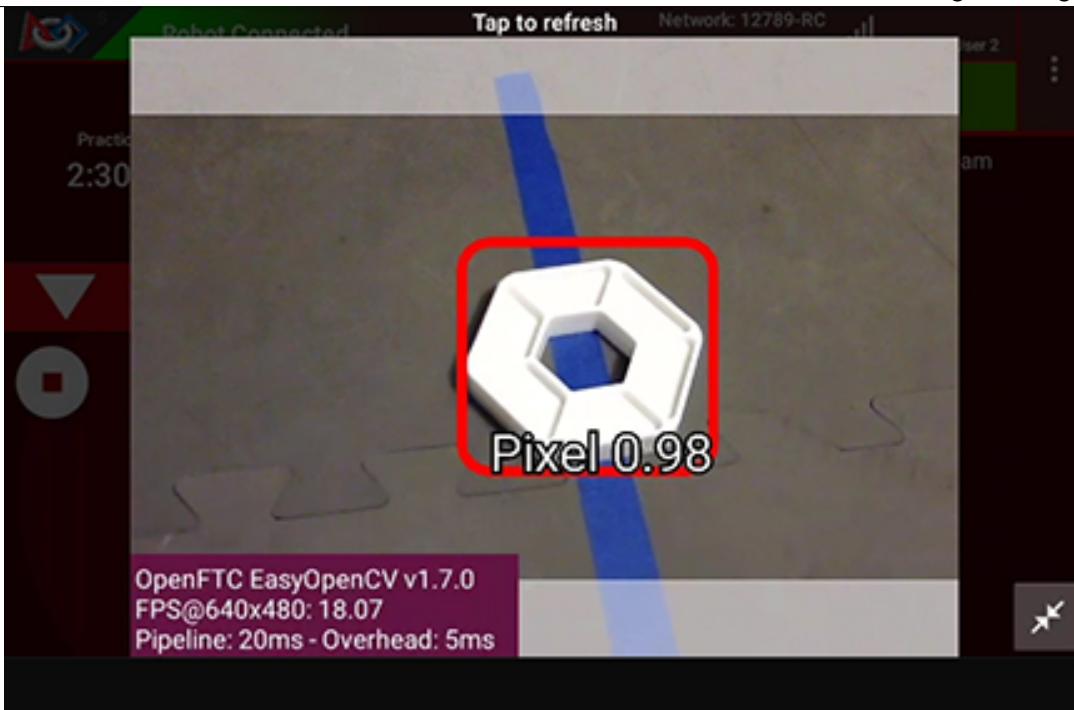


Fig. 2: Example Detection of a Pixel

#### 4.1.4 Notes on Training the CENTERSTAGE Model

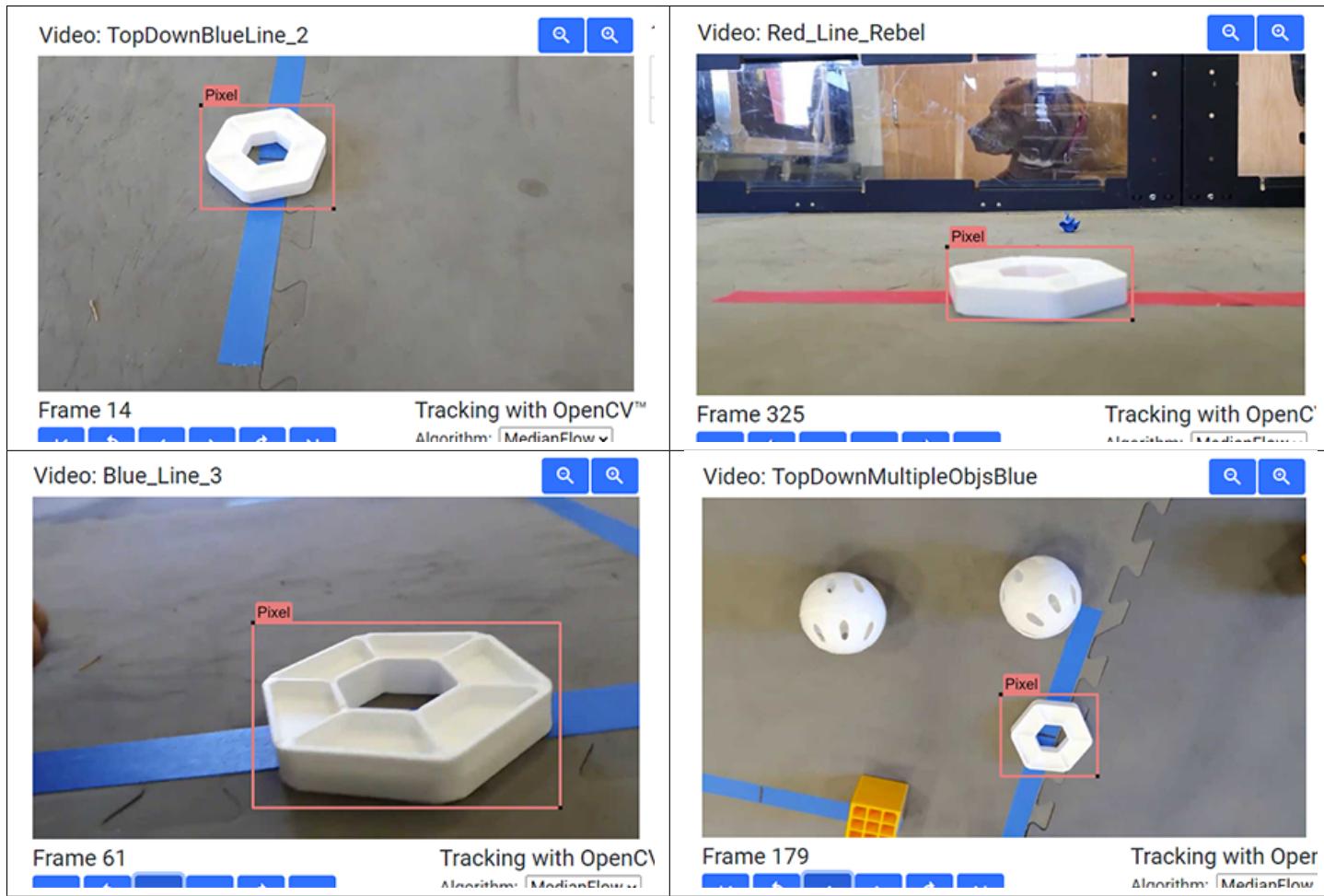
The Pixel game piece posed an interesting challenge for TensorFlow Object Detection (TFOD). As is warned in the Machine Learning Toolkit documentation, TFOD is not very good with recognizing and differentiating simple geometric shapes, nor distinguishing between specific colors; instead, TFOD is good 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 1: Examples of Challenging Scenarios



#### 4.1.5 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 2: 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.

## 4.1.6 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 no 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!

## 4.2 TensorFlow for POWERPLAY presented by Raytheon Technologies

### 4.2.1 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).

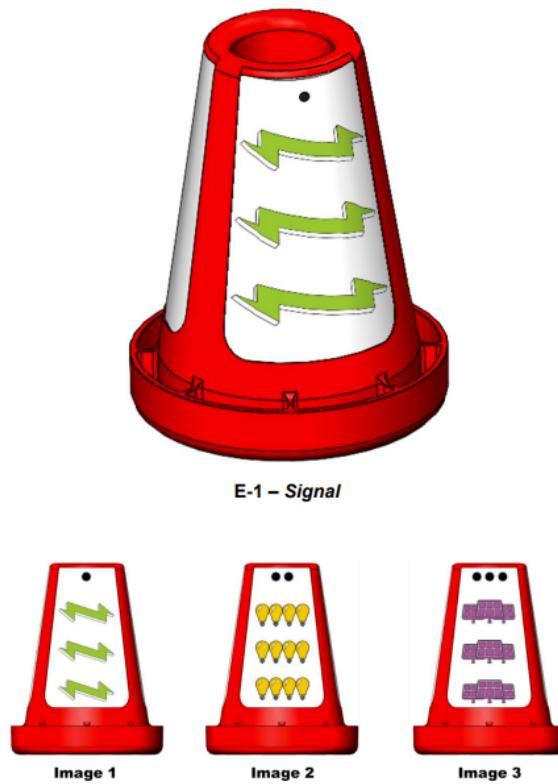


Fig. 3: This season's TFOD model can recognize Signal image elements

---

**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.

---

#### 4.2.2 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.

#### 4.2.3 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.

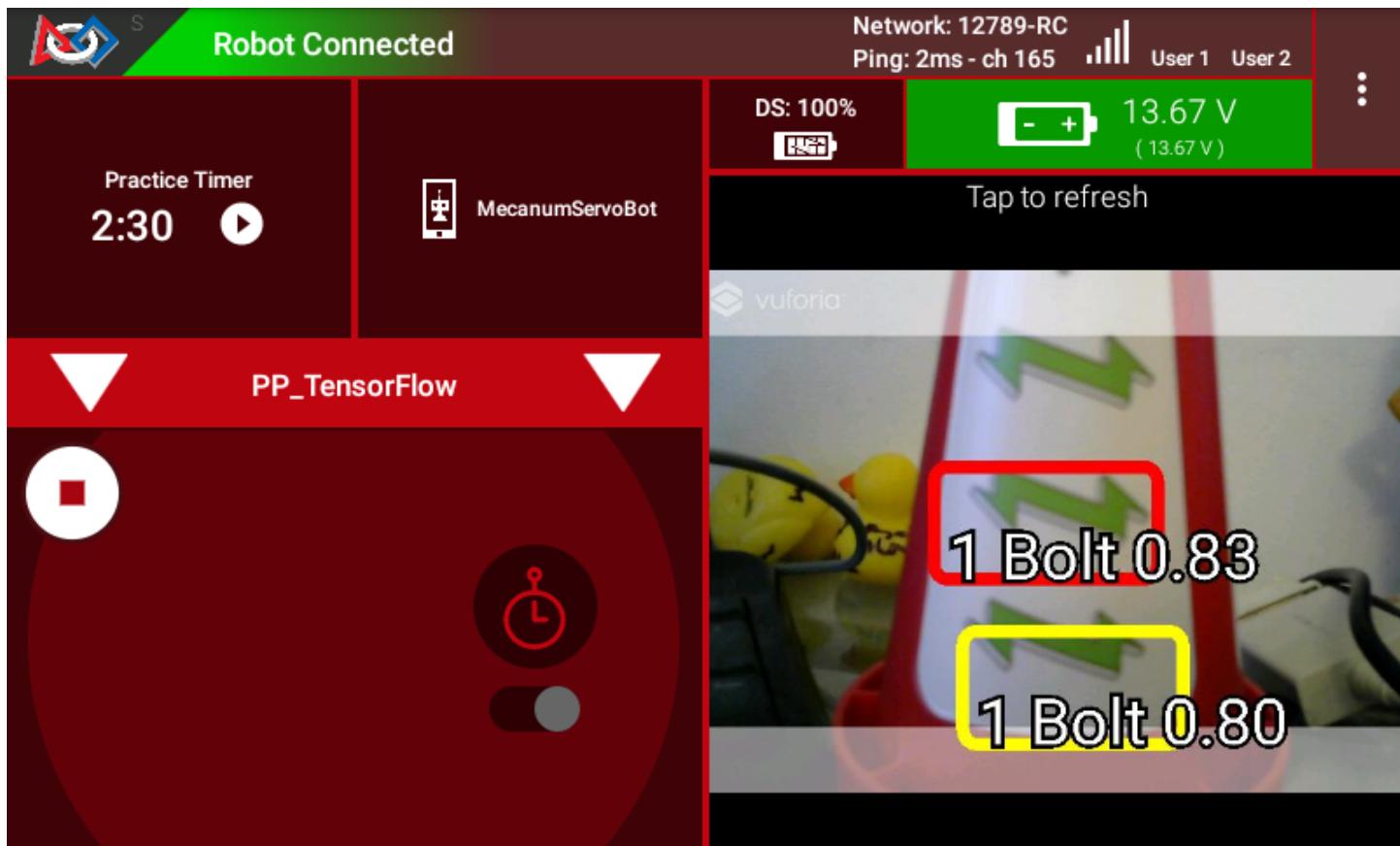
The *FIRST* Tech Challenge software (Robot Controller App and Android Studio Project) includes sample op modes (Blocks and Java versions) that demonstrate how to use the **default inference model**. These tutorials show how to use the sample op modes, using examples from previous *FIRST* Tech Challenge 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

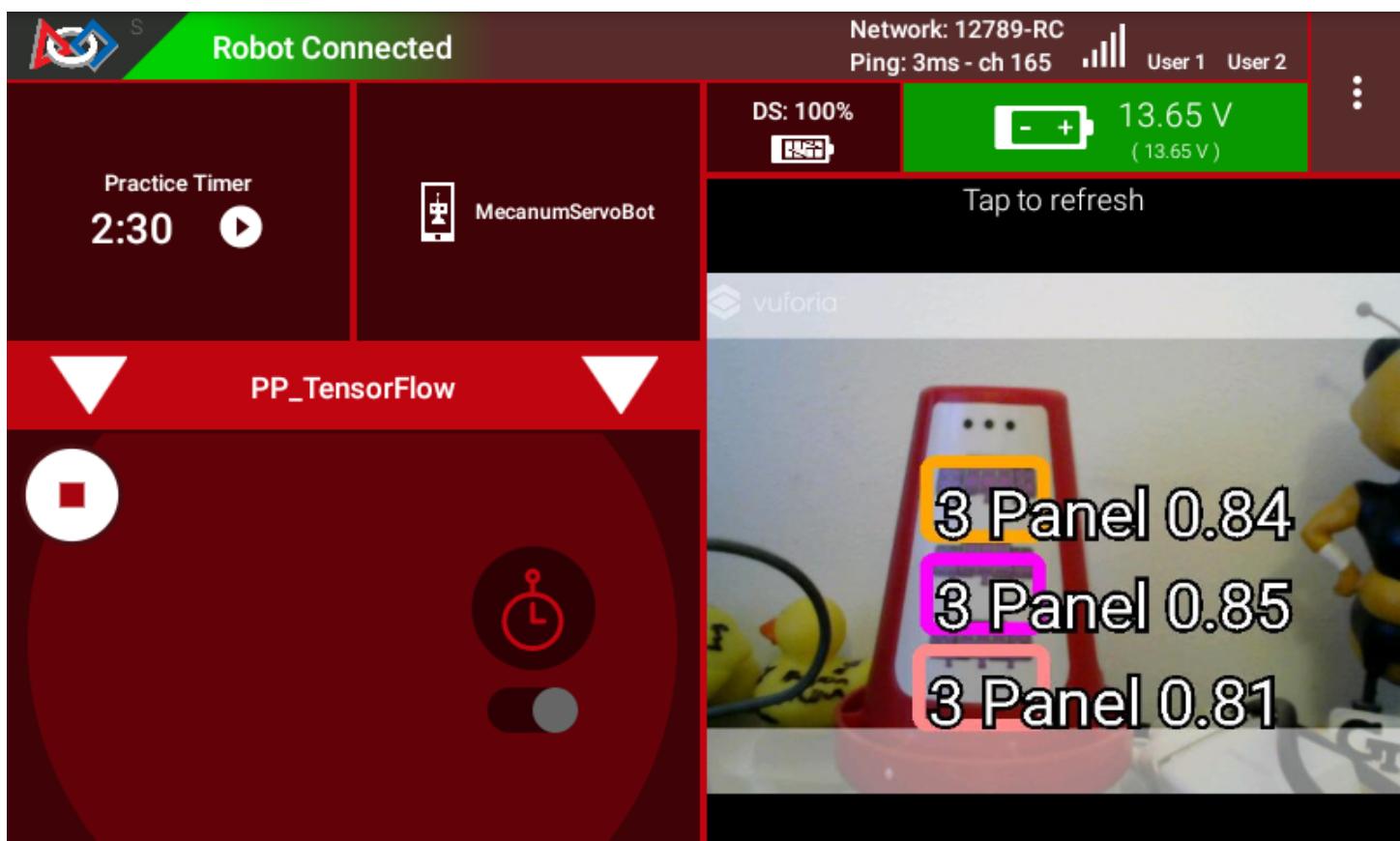
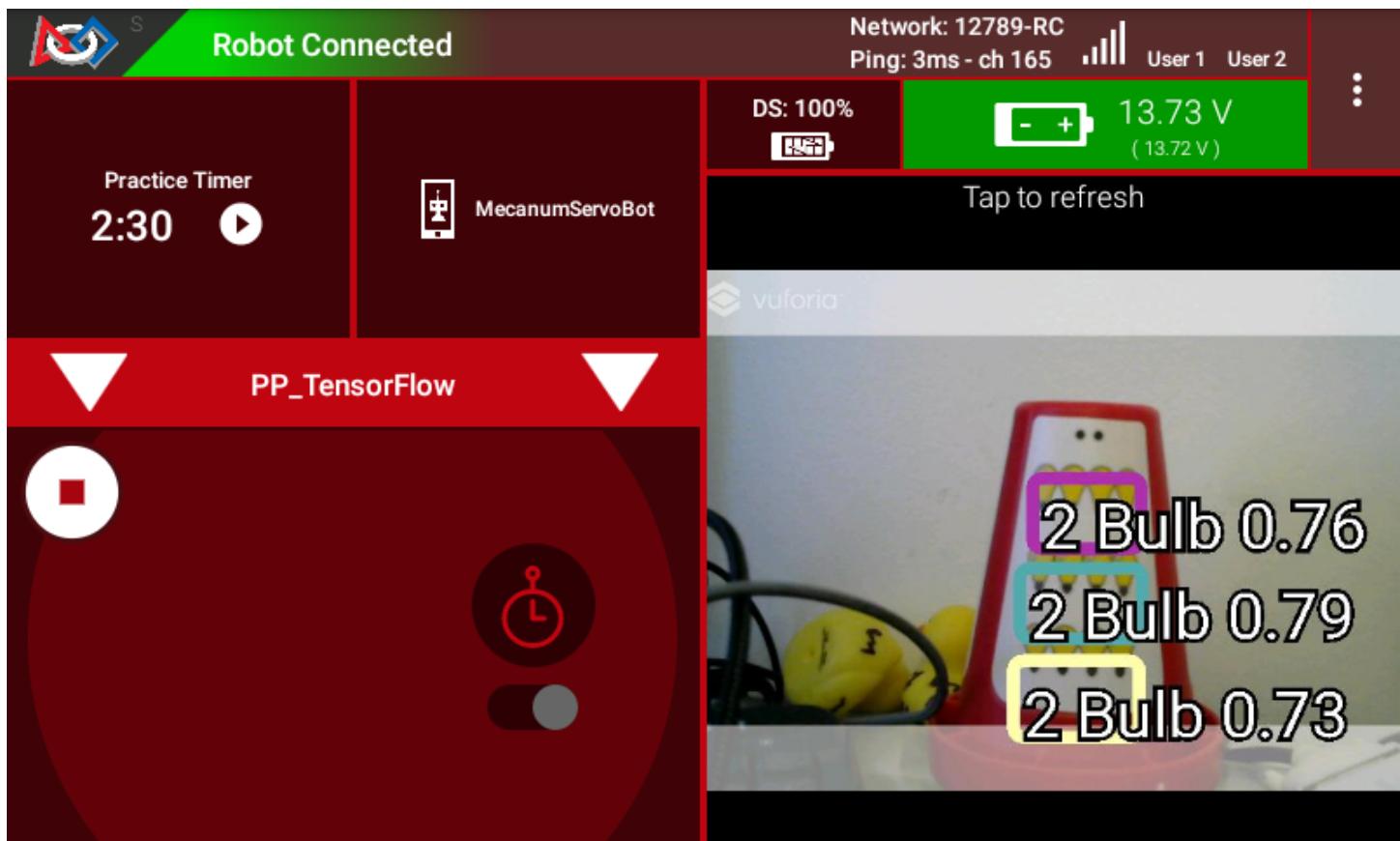
#### Example Image 3

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.



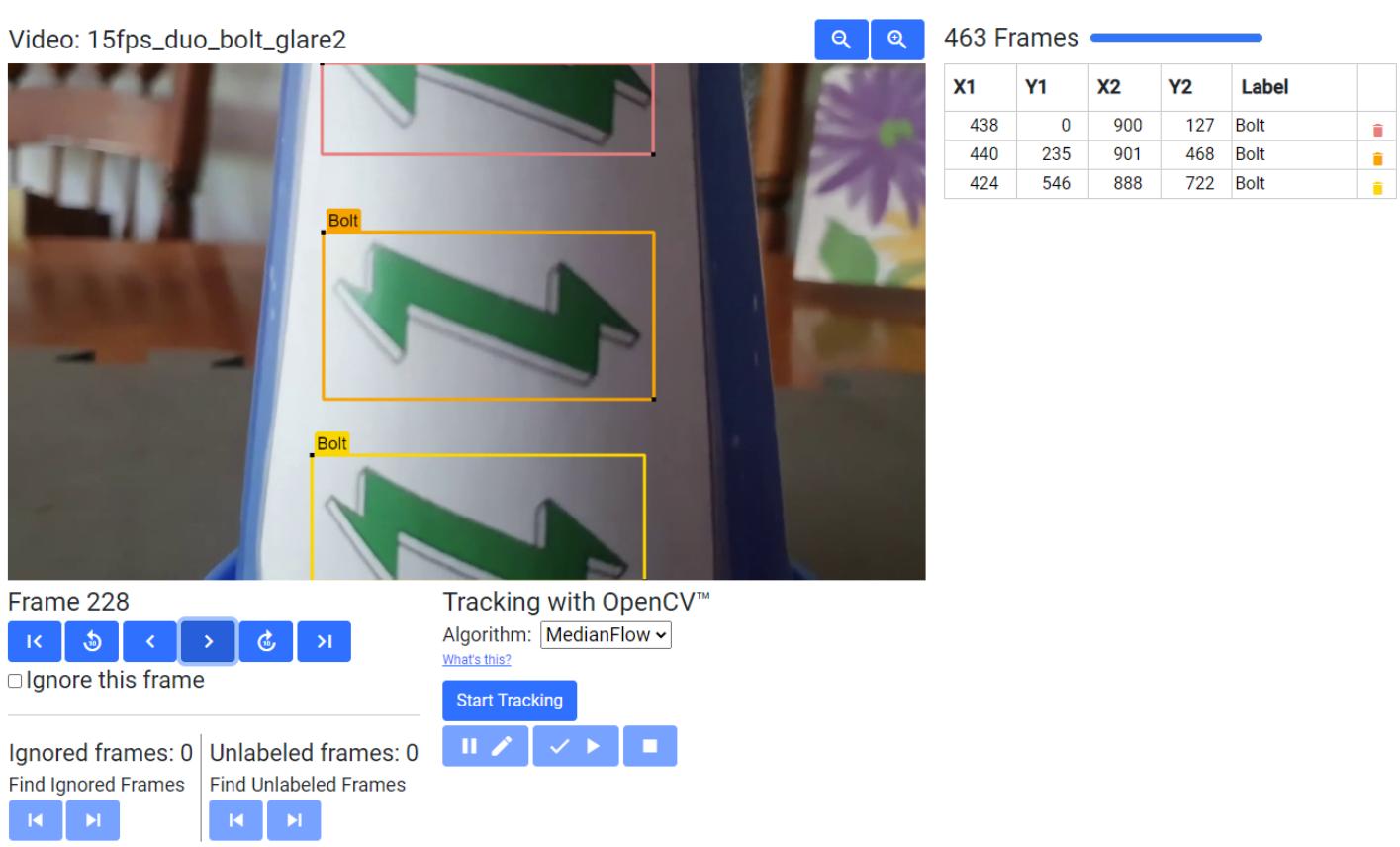
#### 4.2.4 Default POWERPLAY Model Detection Notes

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

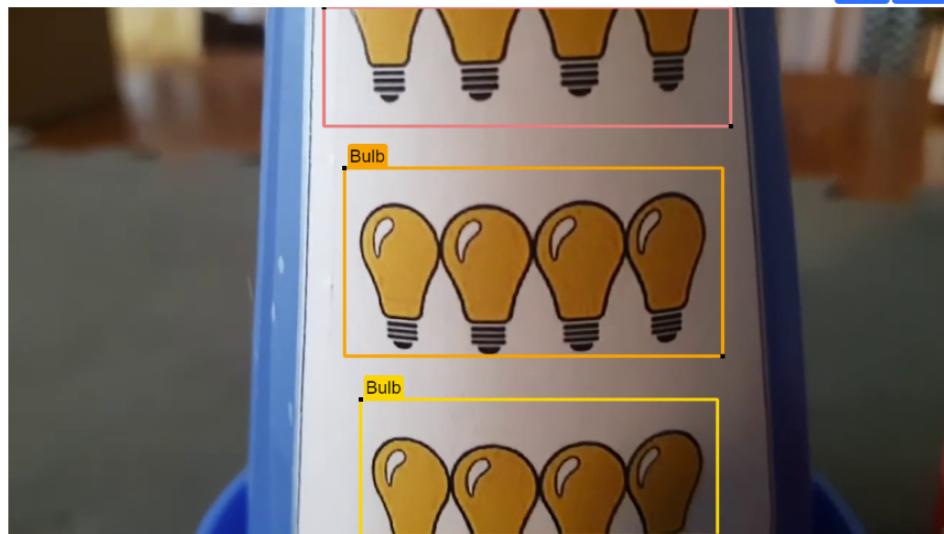
##### Example Training for a Panel

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	✖



Frame 194

 Ignore this frame

Tracking with OpenCV™

Algorithm: MedianFlow ▾

[What's this?](#)

Start Tracking



Ignored frames: 0 Unlabeled frames: 0

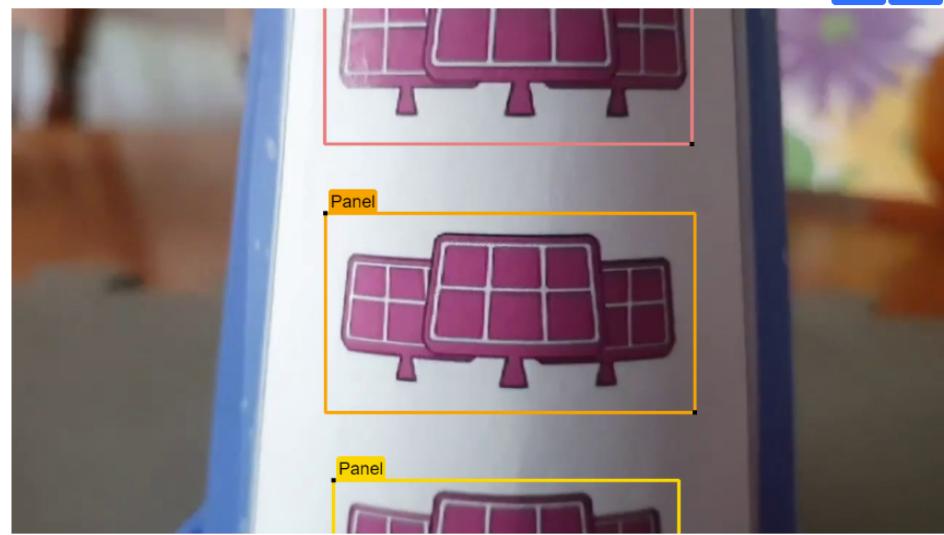
[Find Ignored Frames](#)[Find Unlabeled Frames](#)

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	✖



Frame 241

 Ignore this frame

Tracking with OpenCV™

Algorithm: CSRT ▾

[What's this?](#)

Start Tracking



Ignored frames: 0 Unlabeled frames: 0

[Find Ignored Frames](#)[Find Unlabeled Frames](#)

FOR INSPIRATION &amp; RECOGNITION OF SCIENCE &amp; TECHNOLOGY

Release 0.2 21/08/2024

## 4.2.5 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

Video: 15fps\_duo\_bolt\_wb2

258 Frames

X1	Y1	X2	Y2	Label	
303	151	450	235	Bolt	
310	247	461	331	Bolt	
301	346	461	432	Bolt	
981	164	1122	245	Bolt	
961	346	1100	419	Bolt	
973	252	1114	324	Bolt	

Frame 44

Ignore this frame

Tracking with OpenCV™

Algorithm: CSRT

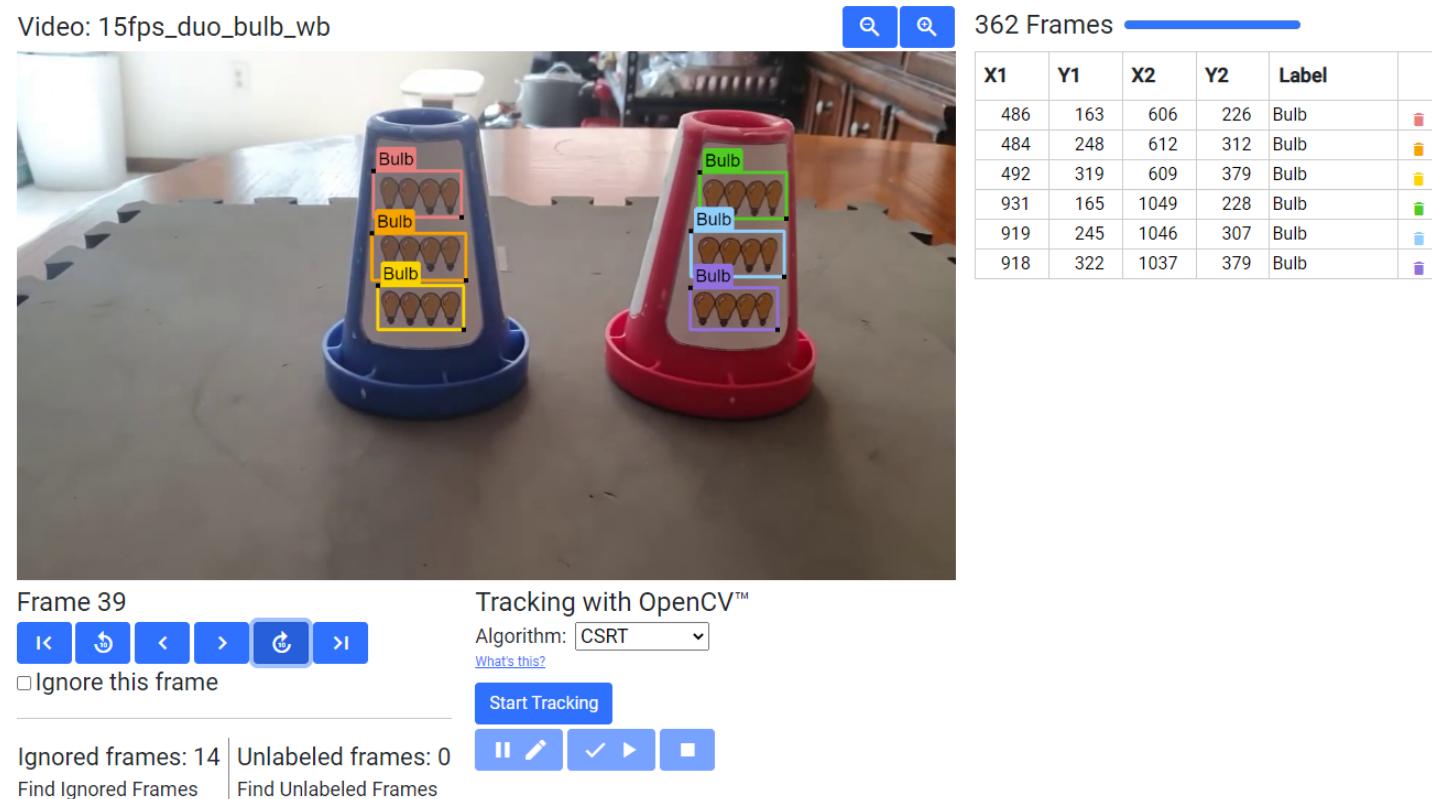
[What's this?](#)

Ignored frames: 0 | Unlabeled frames: 0

[Find Ignored Frames](#) | [Find Unlabeled Frames](#)

## 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.

#### 4.2.6 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.

Video: 15fps\_duo\_panel\_glare2

428 Frames

X1	Y1	X2	Y2	Label	
244	127	360	192	Panel	■
253	212	369	273	Panel	■
260	288	381	349	Panel	■
717	111	833	175	Panel	■
714	192	831	253	Panel	■
710	264	828	328	Panel	■

Frame 14

Tracking with OpenCV™

Algorithm: CSRT

Ignore this frame

Start Tracking

Ignored frames: 0 | Unlabeled frames: 0

[Find Ignored Frames](#) [Find Unlabeled Frames](#)

- 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 minimum 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!

## 4.3 TensorFlow for FREIGHT FRENZY presented by Raytheon Technologies

### 4.3.1 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.

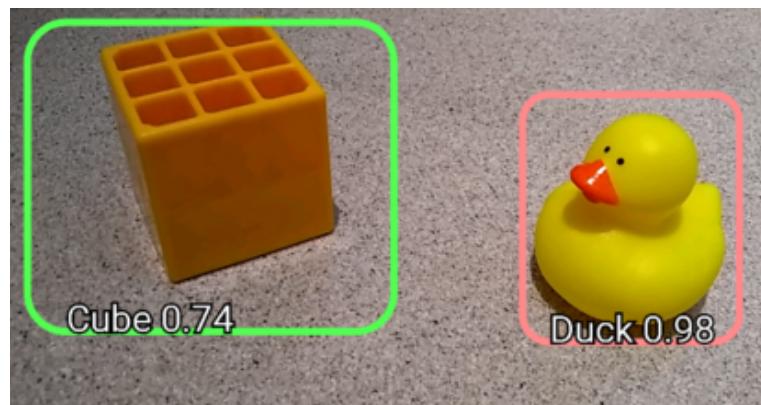


Fig. 4: This season's TFOD model can recognize Freight elements

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).

### 4.3.2 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.

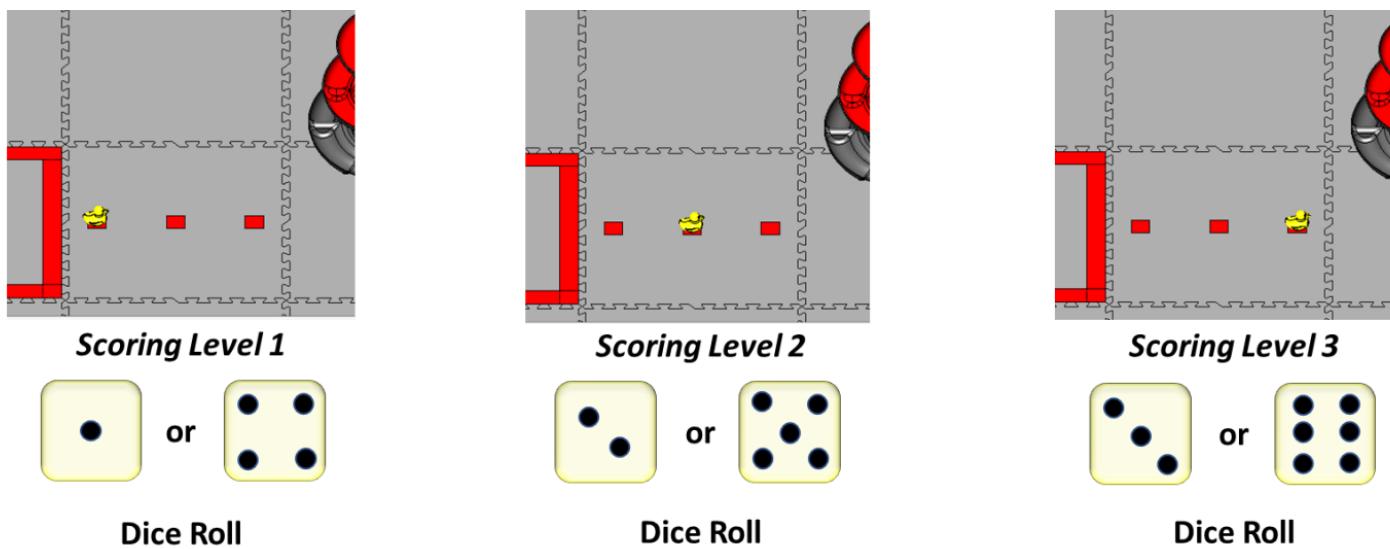


Fig. 5: Randomization

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.

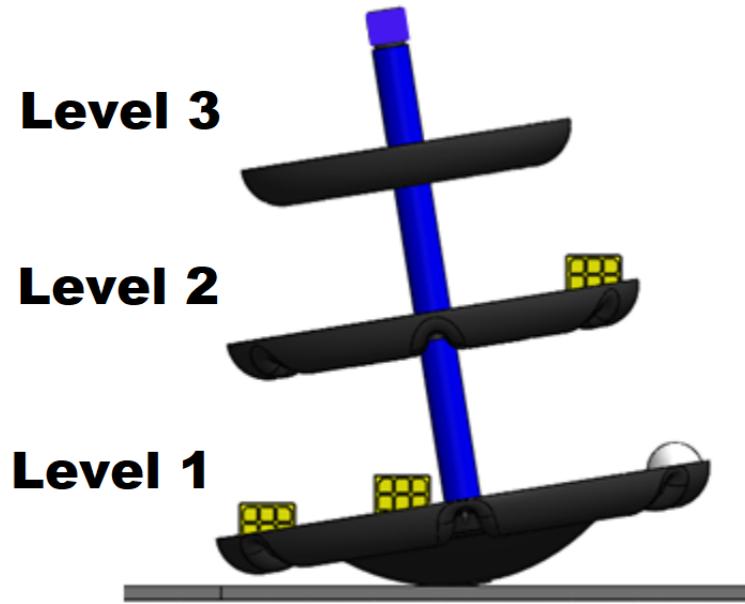


Fig. 6: Alliance Shipping Hub

#### 4.3.3 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.

#### 4.3.4 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](#)

#### 4.3.5 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.

## 4.3.6 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

## 4.4 Blocks Sample OpMode for TFOD

### 4.4.1 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](#)

### 4.4.2 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).

### 4.4.3 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,

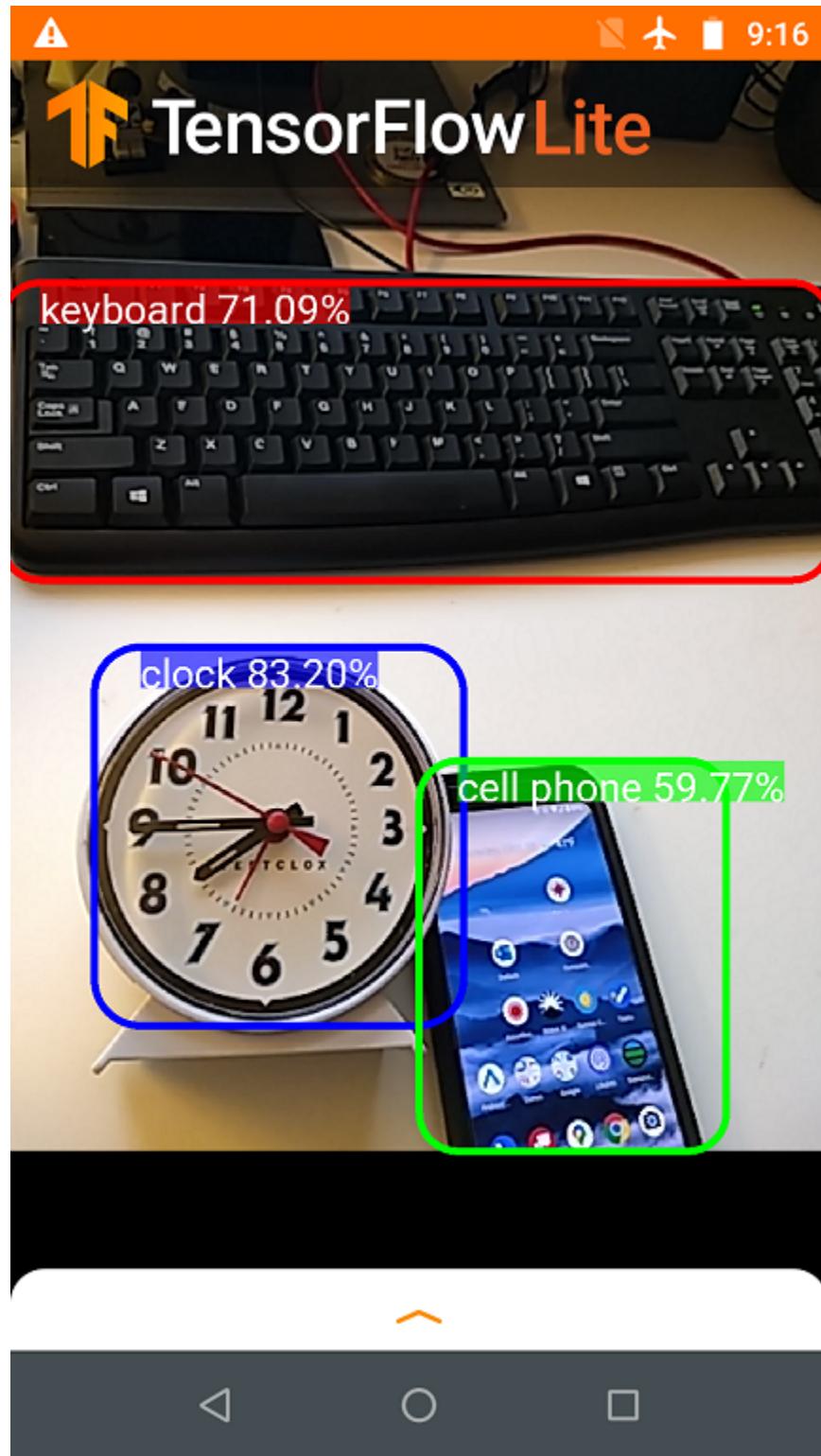


Fig. 7: TensorFlow can recognize everyday objects

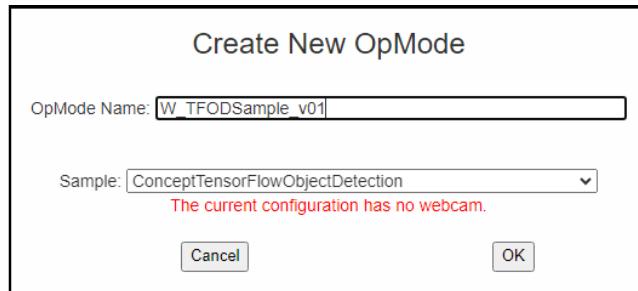


Fig. 8: Creating a New OpMode

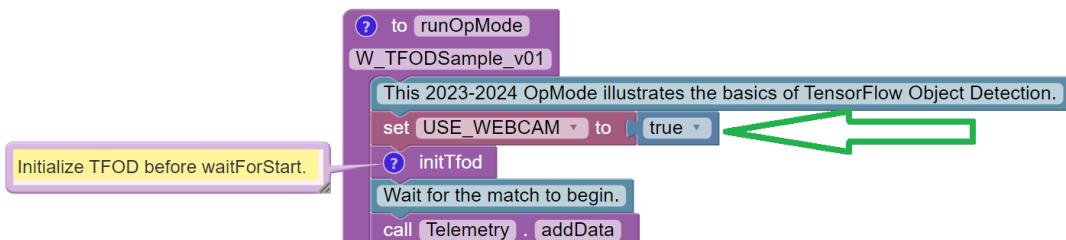


Fig. 9: Sample OpMode

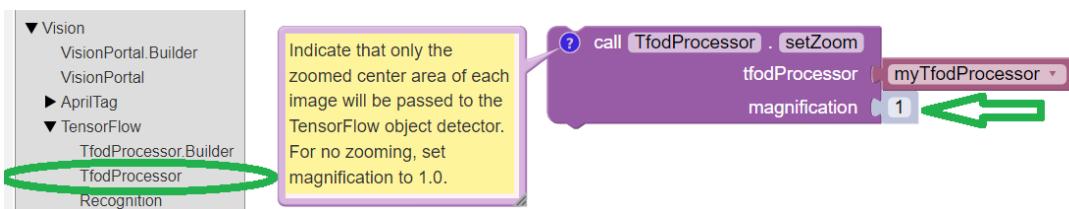


Fig. 10: Setting the Zoom Factor

- immediately after the call to the `initTfod` Function, or
- 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.

#### 4.4.4 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).

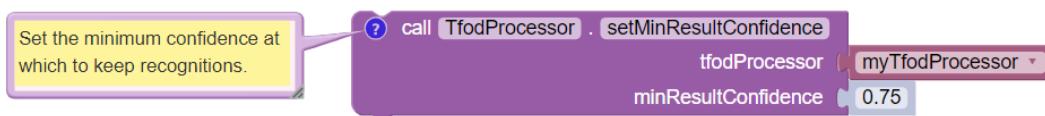


Fig. 11: Setting the Minimum Confidence

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.

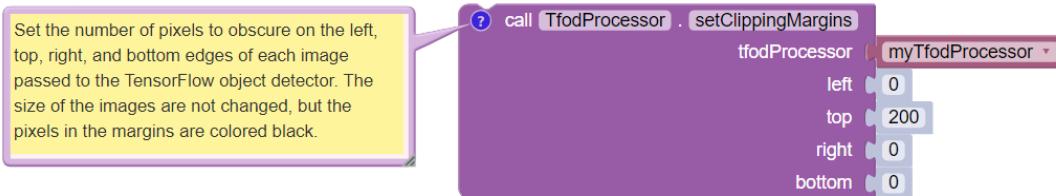


Fig. 12: Setting Clipping Margins

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.

## 4.4.5 Command Flow in this Sample

After the `waitForStart` Block, this OpMode contains the main program loop:

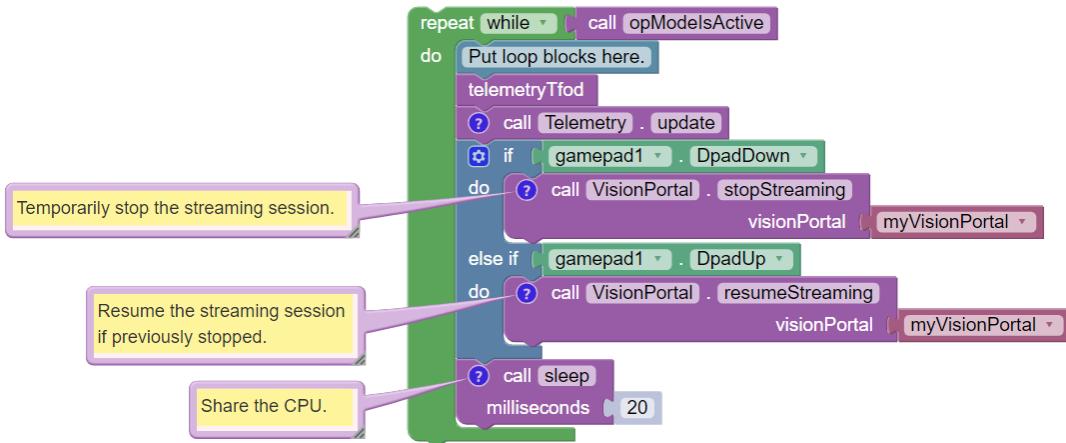


Fig. 13: OpMode Main Loop

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](#).

## 4.4.6 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.

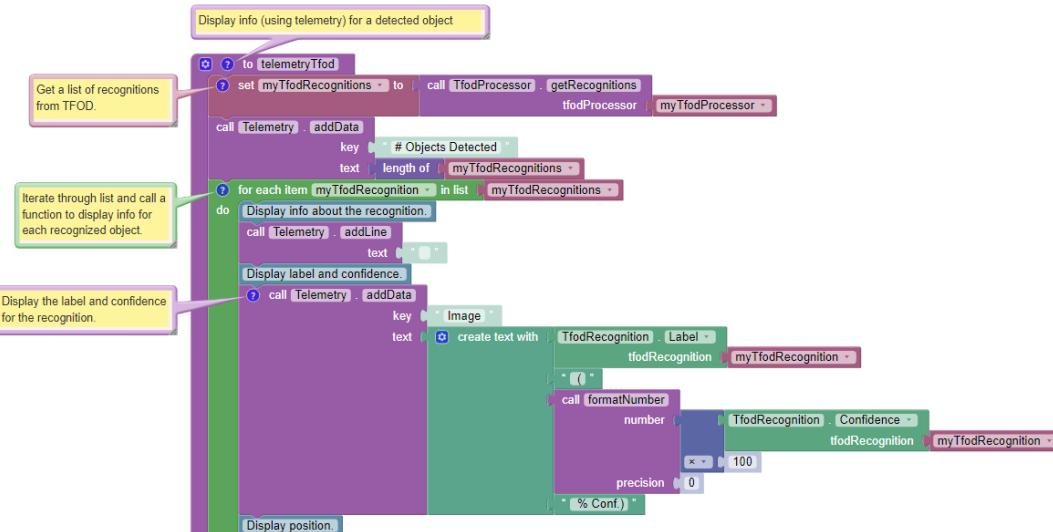


Fig. 14: Telemetry TFOD

#### 4.4.7 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**.

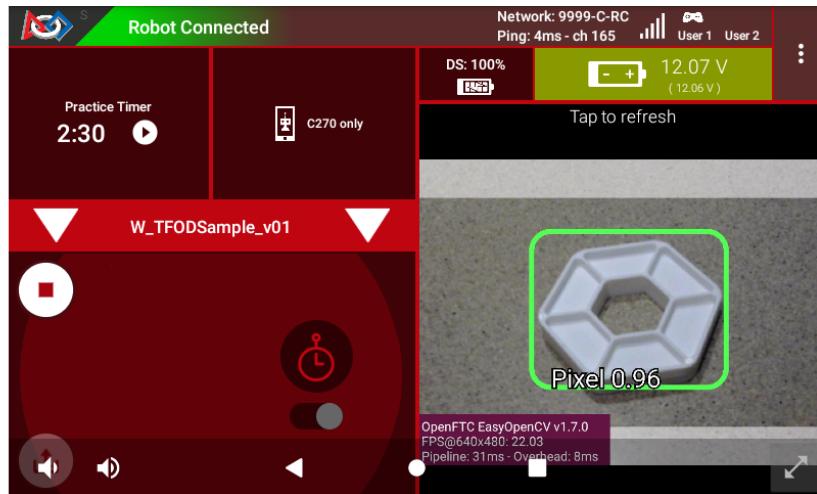


Fig. 15: Sample DS Camera Stream

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.

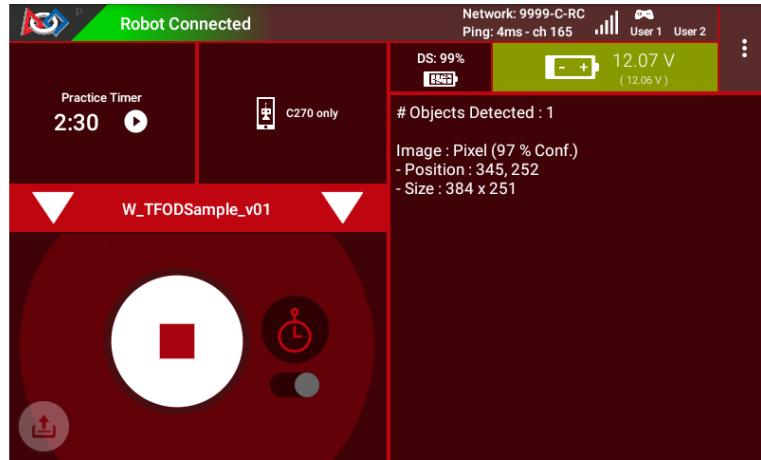


Fig. 16: Sample DS Telemetry



Fig. 17: Sample RC LiveView

If you don't have a physical Pixel on hand, try pointing the camera at this image:



Fig. 18: Sample Pixel

#### 4.4.8 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](mailto:westsiderobotics@verizon.net)

## 4.5 Blocks Custom Model Sample OpMode for TFOD

### 4.5.1 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.

### 4.5.2 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](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.

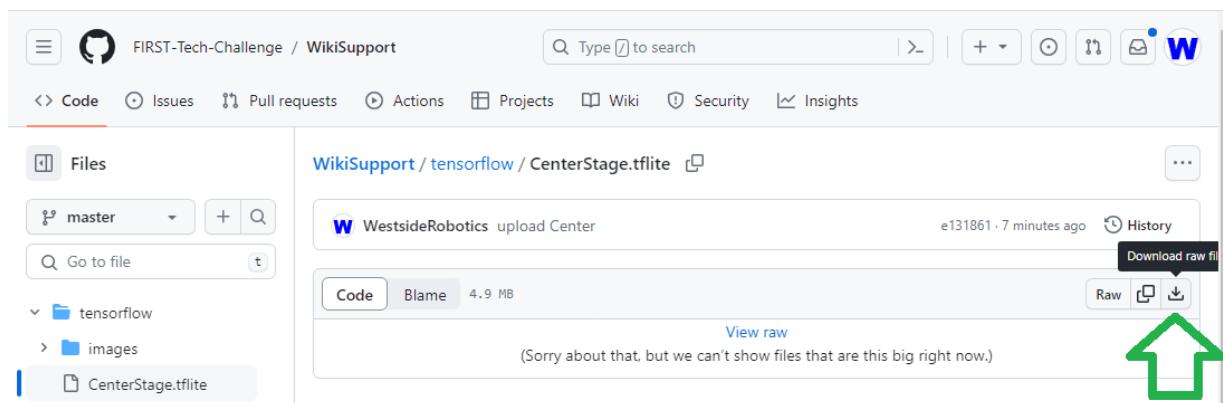


Fig. 19: Public repo for CenterStage tflite file

### 4.5.3 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.

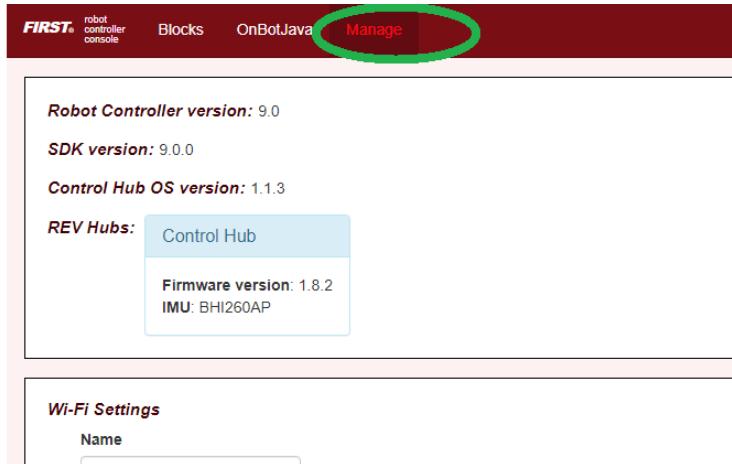


Fig. 20: Example of the Manage Page

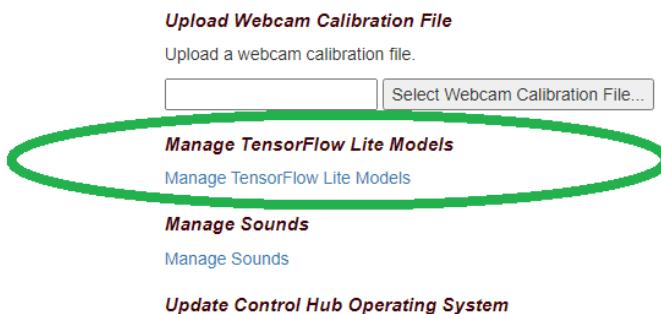


Fig. 21: Manage TFLITE Models Link

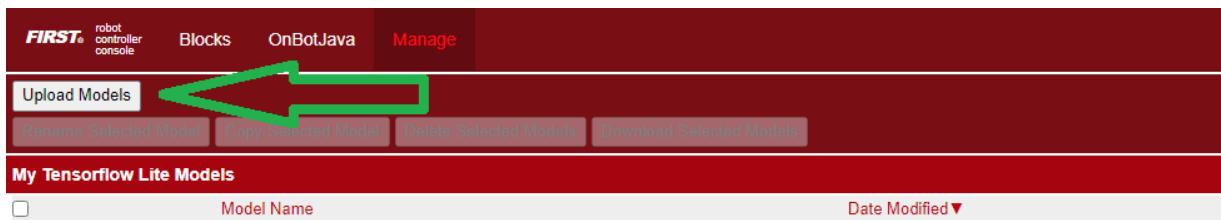


Fig. 22: Upload TFLITE Models Button

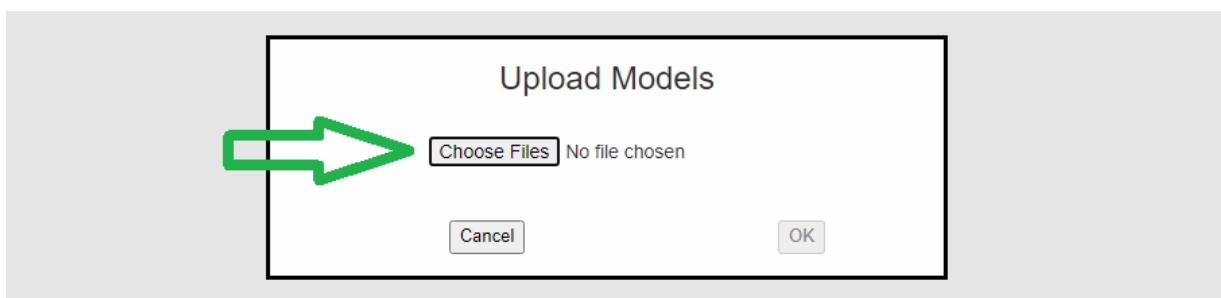


Fig. 23: Upload TFLITE Models Button

The screenshot shows the 'Manage' tab of the FTC Programming Resources. Under 'My Tensorflow Lite Models', there is a table with one row. The first column has an unchecked checkbox. The second column contains the 'Model Name' 'CenterStage.tflite'. The third column contains the 'Date Modified' 'September 5, 2023, 1:20:02 PM'.

	Model Name	Date Modified
<input type="checkbox"/>	CenterStage.tflite	September 5, 2023, 1:20:02 PM

Fig. 24: TFLITE Model Listed

#### 4.5.4 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.

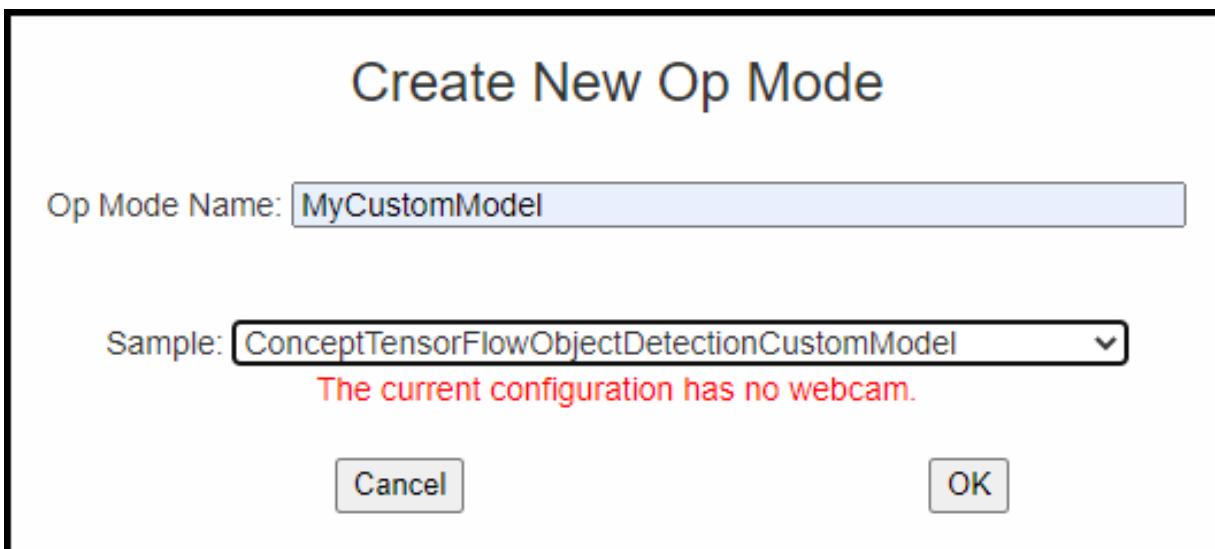


Fig. 25: Create New OpMode

The new OpMode should appear in edit mode in your browser.

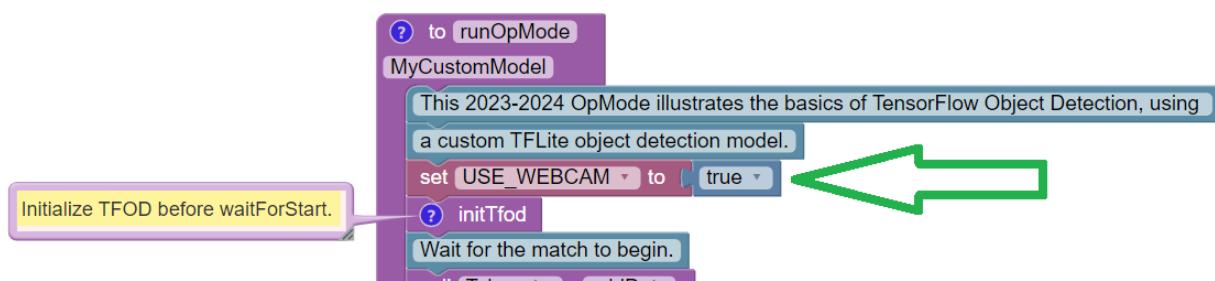


Fig. 26: Sample OpMode

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).

#### 4.5.5 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.

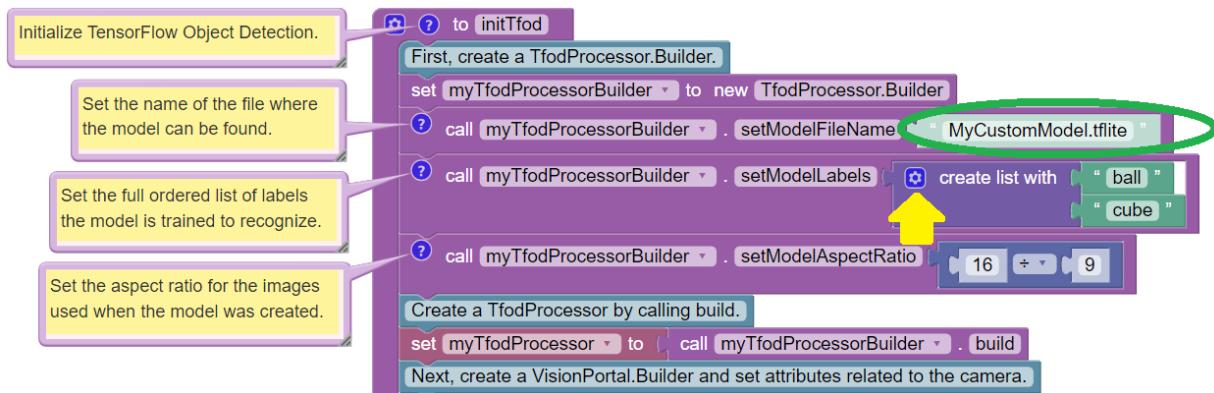


Fig. 27: Init TFOD Function

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).

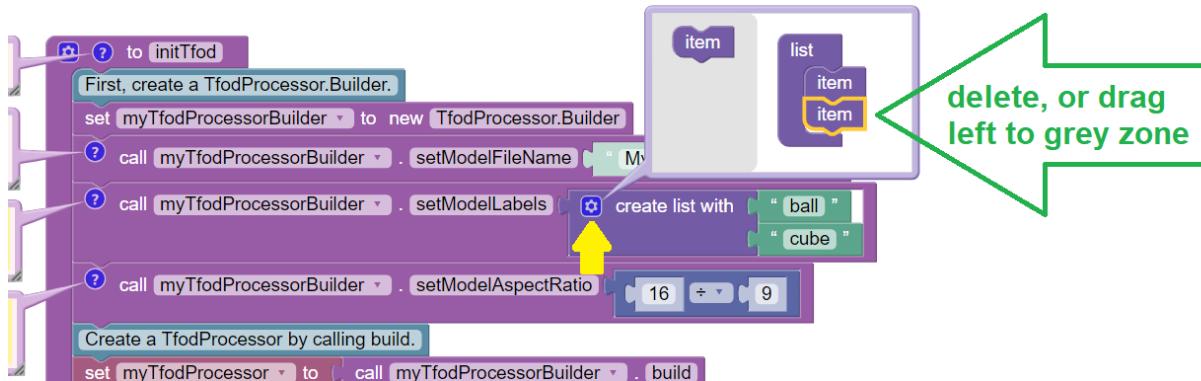


Fig. 28: Blue Gear Delete

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:

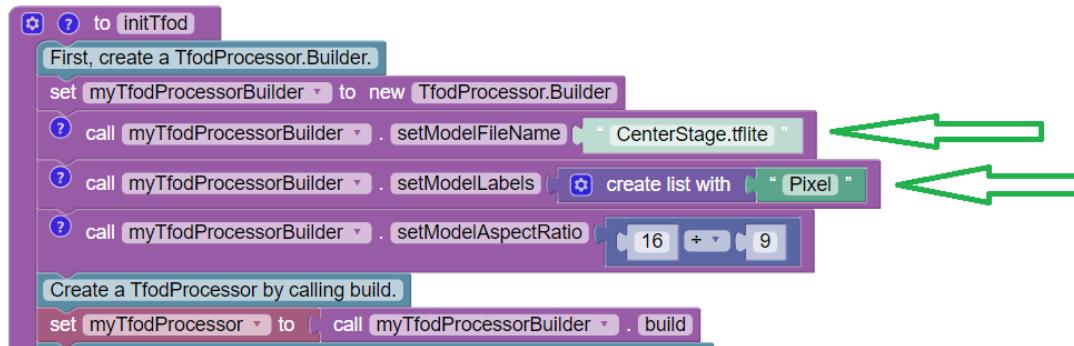


Fig. 29: Adding Pixel Label

#### 4.5.6 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.

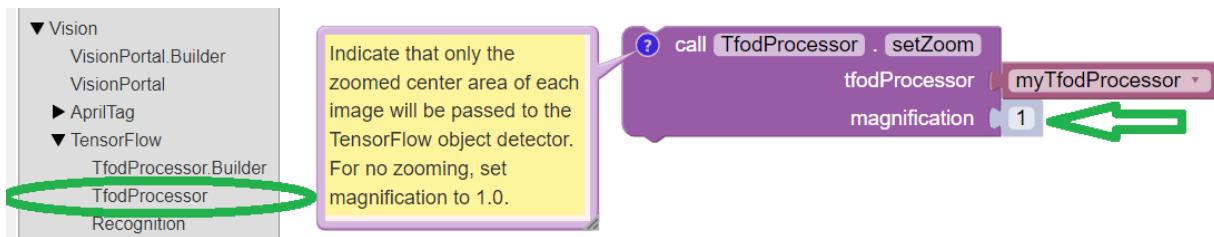


Fig. 30: Set Zoom

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.

#### 4.5.7 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.

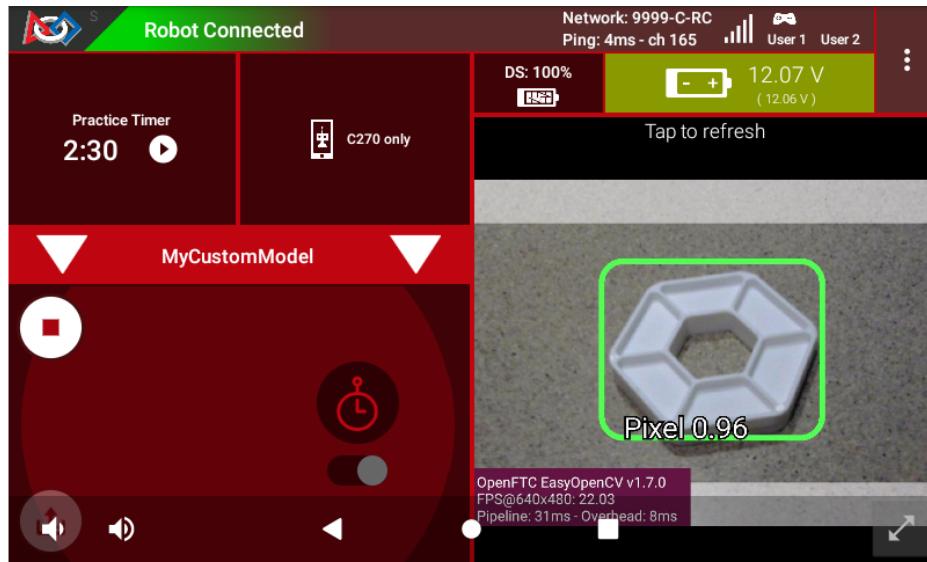


Fig. 31: DS Camera Stream

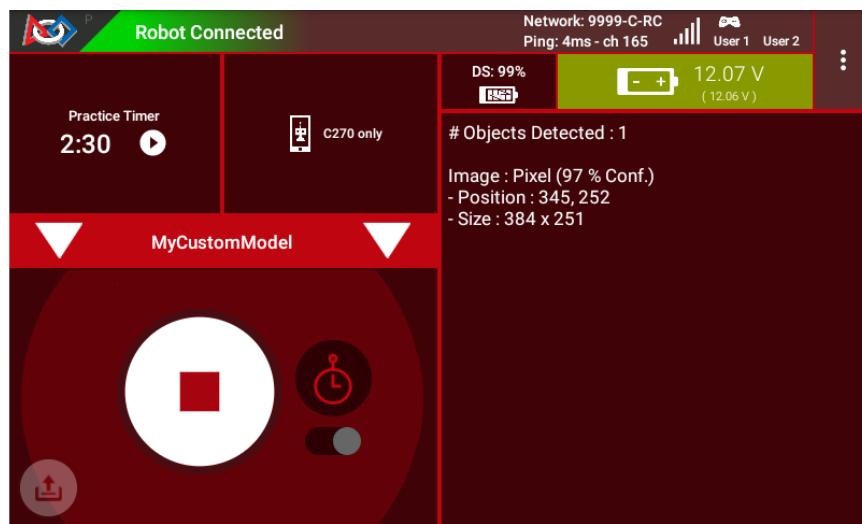


Fig. 32: DS Telemetry

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. 33: 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:

#### 4.5.8 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](mailto:westsiderobotics@verizon.net)



Fig. 34: Sample Pixel

## 4.6 Java Easy Sample OpMode for TFOD

### 4.6.1 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.

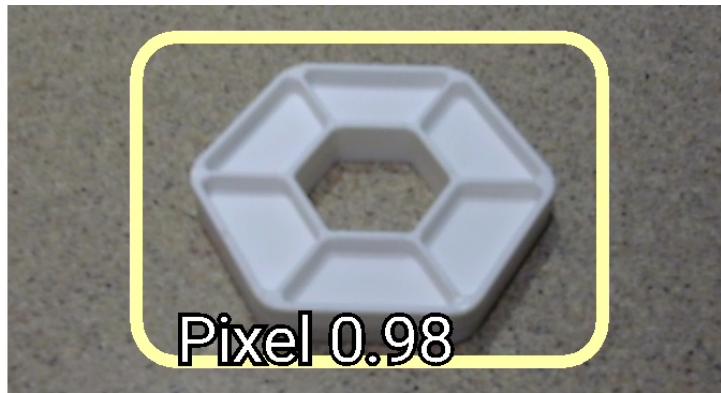


Fig. 35: Sample TFOD Recognition

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.

### 4.6.2 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).

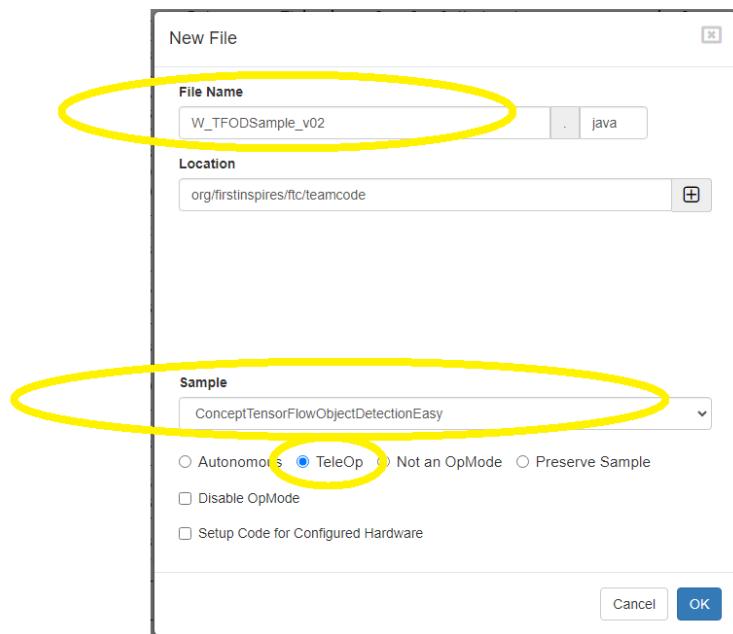


Fig. 36: New File Dialog

Screenshot of the FTC Programming interface showing the code editor. A green circle highlights the file tab for 'W\_TFODSample\_v02.java'. A red circle highlights the line of code 'private static final boolean USE\_WEBCAM = true;'. The code is as follows:

```

30 package org.firstinspires.ftc.teamcode;
31
32 import com.qualcomm.robotcore.eventloop.opmode.Disabled;
33 import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
34 import com.qualcomm.robotcore.eventloop.opmode.TeleOp;
35 import org.firstinspires.ftc.robotcore.external.hardware.camera.BuiltinCameraDirection;
36 import org.firstinspires.ftc.robotcore.external.hardware.camera.WebcamName;
37 import org.firstinspires.ftc.robotcore.external.tfod.Recognition;
38 import org.firstinspires.ftc.vision.VisionPortal;
39 import org.firstinspires.ftc.vision.tfod.TfodProcessor;
40
41 import java.util.List;
42
43 /*
44  * This OpMode illustrates the basics of TensorFlow Object Detection, using
45  * the easiest way.
46  *
47  * Use Android Studio to Copy this Class, and Paste it into your team's code folder with a new name.
48  * Remove or comment out the @Disabled line to add this OpMode to the Driver Station OpMode list.
49  */
50 @TeleOp(name = "Concept: TensorFlow Object Detection Easy", group = "Concept")
51
52 public class W_TFODSample_v02 extends LinearOpMode {
53
54     private static final boolean USE_WEBCAM = true; // true for webcam, false for phone camera
55
56

```

Fig. 37: Opening New Sample

### 4.6.3 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**.

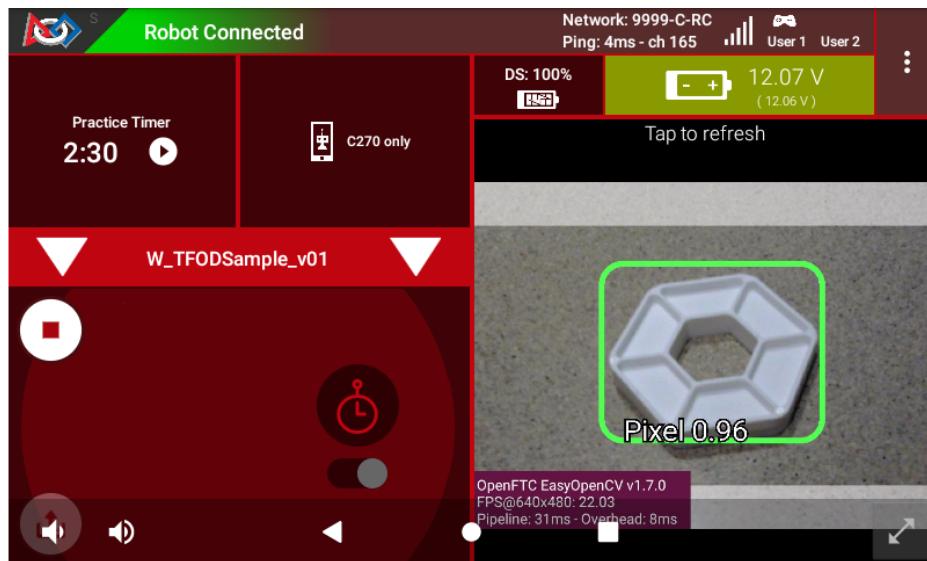


Fig. 38: DS Camera Stream

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:

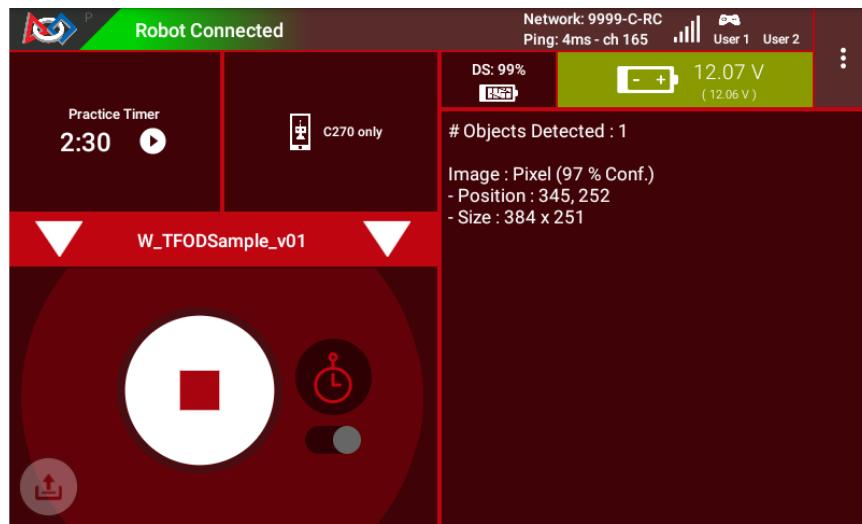


Fig. 39: DS Telemetry Display



Fig. 40: Sample RC LiveView



Fig. 41: Example of a Pixel

## 4.6.4 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.

## 4.6.5 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)

```

        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.

#### 4.6.6 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.

## 4.6.7 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.

## 4.6.8 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.

## 4.6.9 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](mailto:westsiderobotics@verizon.net)

## 4.7 Java Custom Model Sample OpMode for TFOD

### 4.7.1 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. 42: Example Pixel Recognition using TFOD

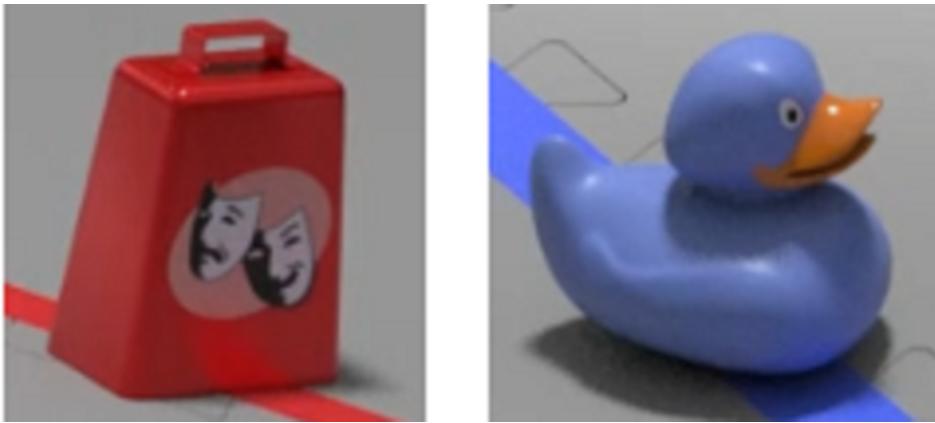


Fig. 43: Example Team Props

For extra points, FTC teams may instead use their own custom TFOD models of game elements, called **Team Props** in CENTERSTAGE.

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.

#### 4.7.2 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.

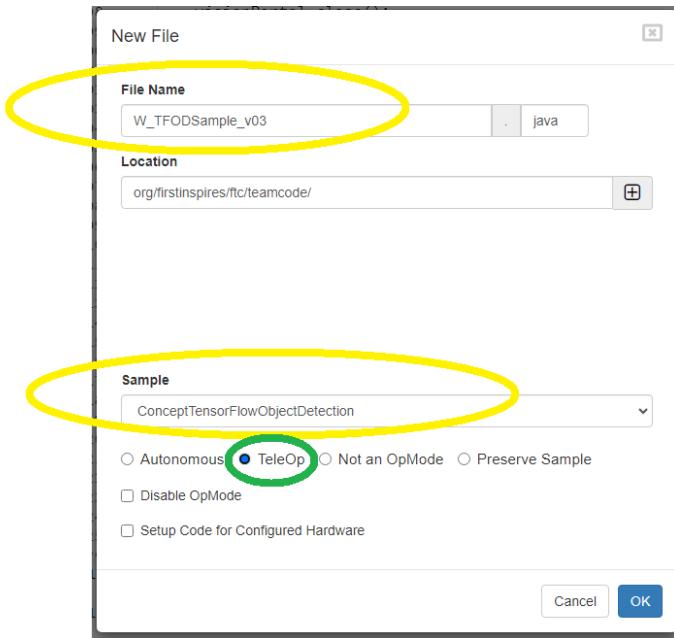


Fig. 44: Example New File Dialog

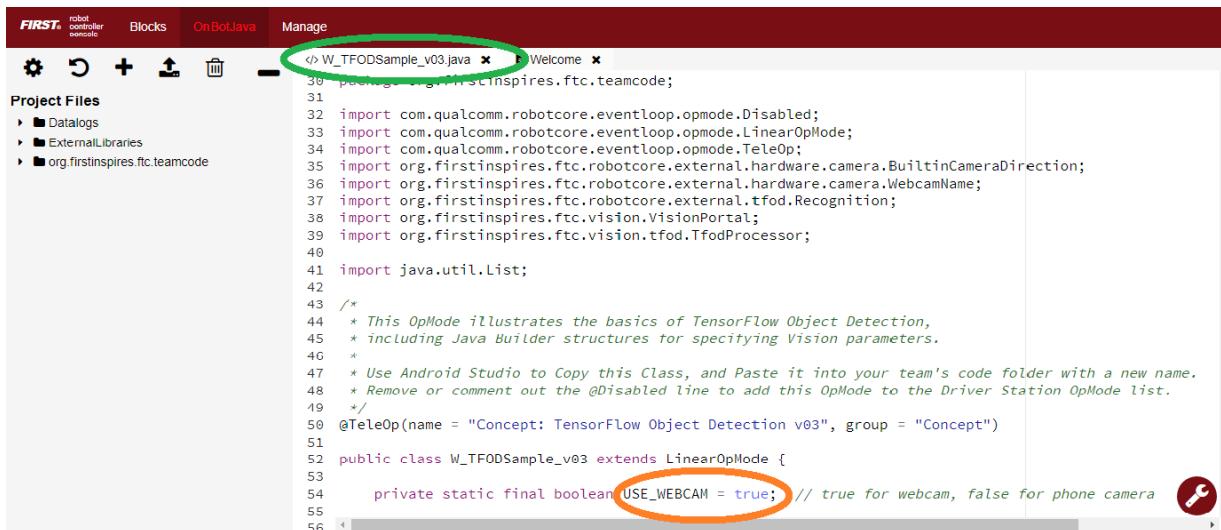
Specify a name for your new OpMode. Select “**ConceptTensorFlowObjectDetection**” as the Sample OpMode to be the template for your new OpMode.

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.



```

FIRST® robot controller
Blocks OnBotJava Manage
Project Files
  ▶ Datalogs
  ▶ ExternalLibraries
  ▶ org.firstinspires.ftc.teamcode

< W_TFODSample_v03.java x Welcome x
30 package org.firstinspires.ftc.teamcode;
31
32 import com.qualcomm.robotcore.eventloop.opmode.Disabled;
33 import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
34 import com.qualcomm.robotcore.eventloop.opmode.TeleOp;
35 import org.firstinspires.ftc.robotcore.external.hardware.camera.BuiltinCameraDirection;
36 import org.firstinspires.ftc.robotcore.external.hardware.camera.WebcamName;
37 import org.firstinspires.ftc.robotcore.external.tfod.Recognition;
38 import org.firstinspires.ftc.vision.VisionPortal;
39 import org.firstinspires.ftc.vision.tfod.TfodProcessor;
40
41 import java.util.List;
42
43 /**
44 * This OpMode illustrates the basics of TensorFlow Object Detection,
45 * including Java Builder structures for specifying Vision parameters.
46 *
47 * Use Android Studio to Copy this Class, and Paste it into your team's code folder with a new name.
48 * Remove or comment out the @Disabled line to add this OpMode to the Driver Station OpMode list.
49 */
50 @TeleOp(name = "Concept: TensorFlow Object Detection v03", group = "Concept")
51
52 public class W_TFODSample_v03 extends LinearOpMode {
53
54     private static final boolean USE_WEBCAM = true; // true for webcam, false for phone camera
55
56

```

Fig. 45: 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).

#### 4.7.3 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**.

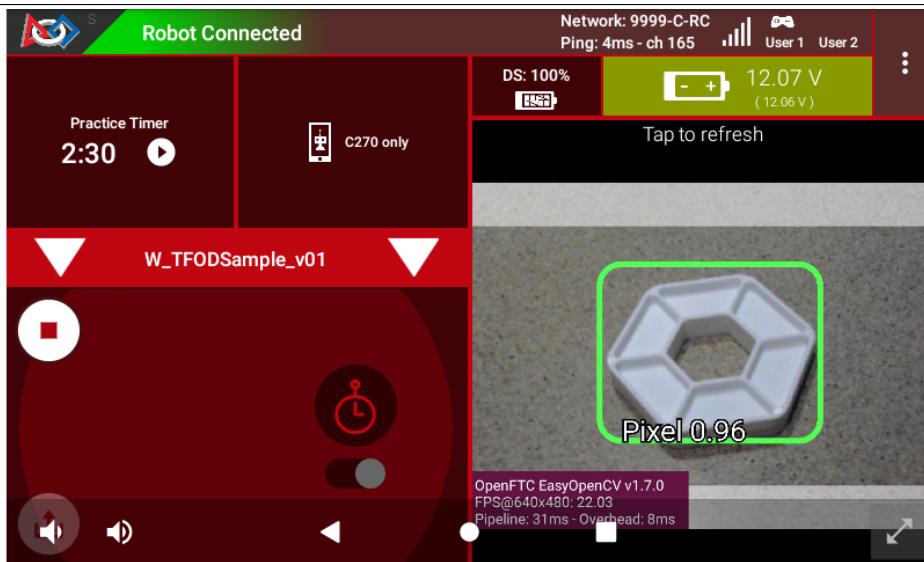


Fig. 46: Sample DS Camera Stream

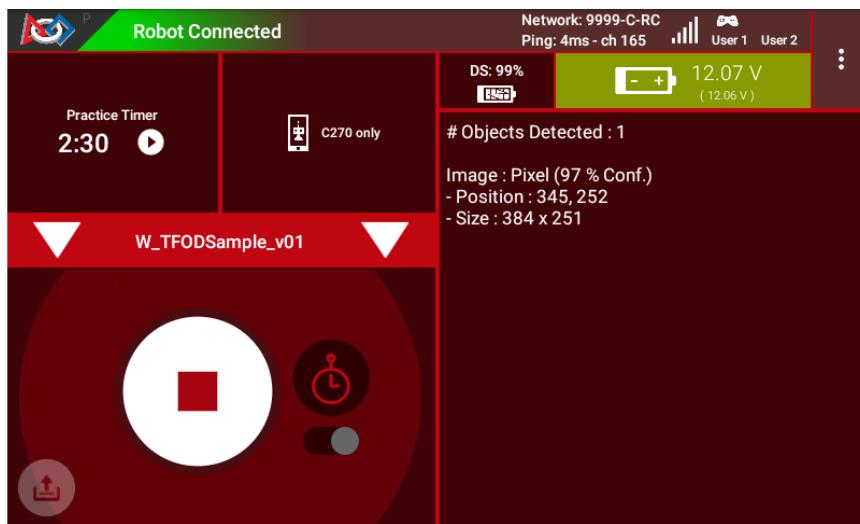


Fig. 47: Sample DS Telemetry



Fig. 48: Sample 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:



Fig. 49: Sample Pixel

**Congratulations!** At this point the Sample OpMode and your camera are working properly. Ready for a custom model?

#### 4.7.4 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](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.

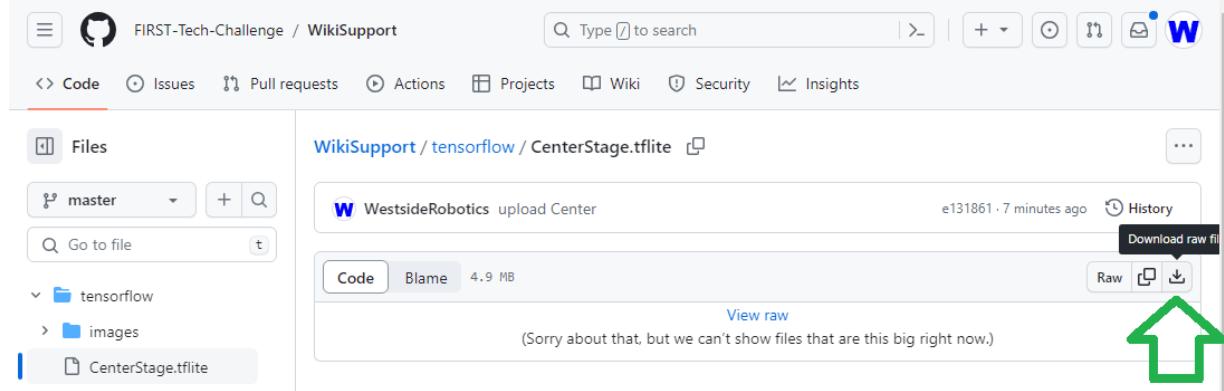


Fig. 50: Public Repo

#### 4.7.5 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:

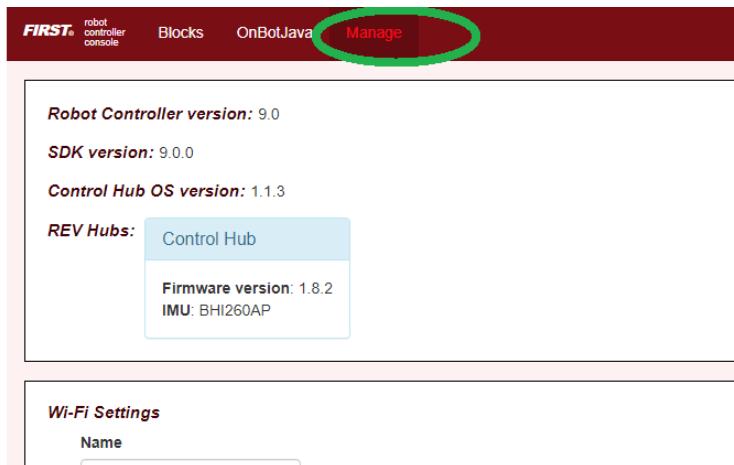


Fig. 51: Robot Controller Manage Page

**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.

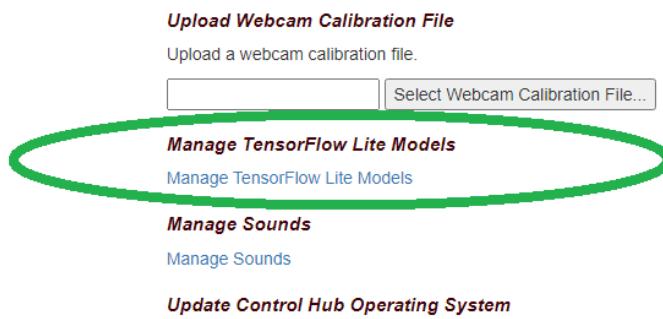


Fig. 52: TensorFlow Lite Model Management

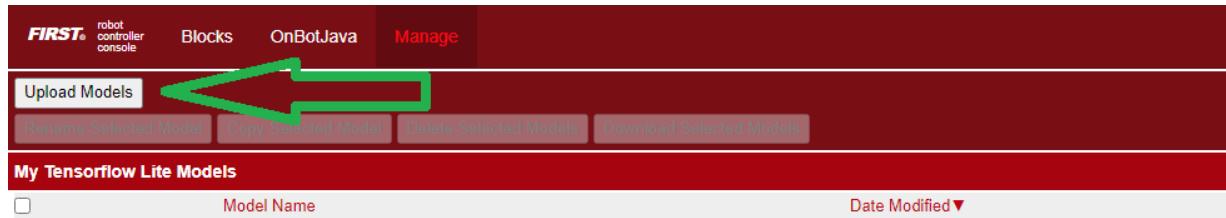


Fig. 53: Upload Models

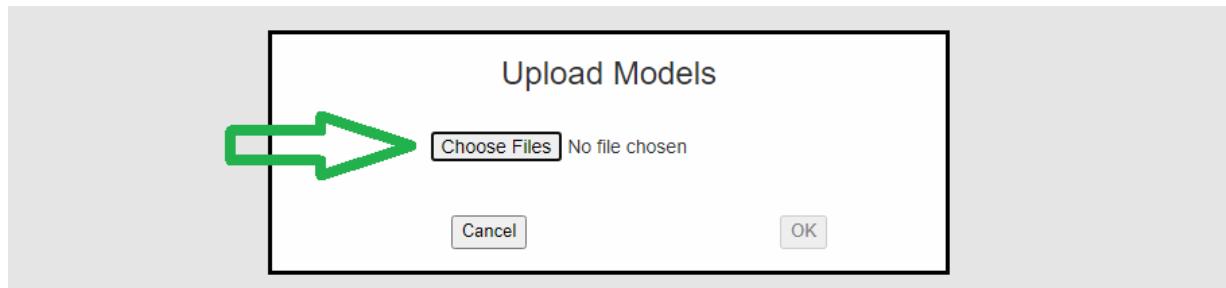


Fig. 54: Choose Files

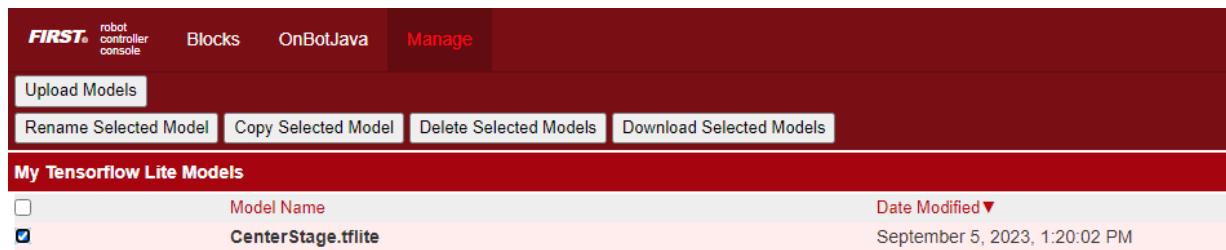


Fig. 55: CENTERSTAGE TFLITE File Uploaded

## 4.7.6 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.

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.

```

116     private void initTfod() {
117
118         // Create the TensorFlow processor by using a builder.
119         tfod = new TfodProcessor.Builder()
120
121             // With the following lines commented out, the default TfodProcessor Builder
122             // will load the default model for the season. To define a custom model to load,
123             // choose one of the following:
124             // Use setModelAssetName() if the custom TF Model is built in as an asset (AS only).
125             // Use setModelFilename() if you have downloaded a custom team model to the Robot Controller.
126             //.setModelAssetName(TFOD_MODEL_ASSET)
127             .setModelFileName(TFOD_MODEL_FILE)
128
129             // The following default settings are available to un-comment and edit as needed to
130             // set parameters for custom models.
131             .setModelLabels(LABELS)
132             //.setIsModelTensorFlow(true)
133             //.setIsModelQuantized(true)
134             //.setModelInputSize(300)
135             //.setModelAspectRatio(16.0 / 9.0)
136
137         .build();
138

```

Fig. 56: Builder Pattern Settings

#### 4.7.7 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.

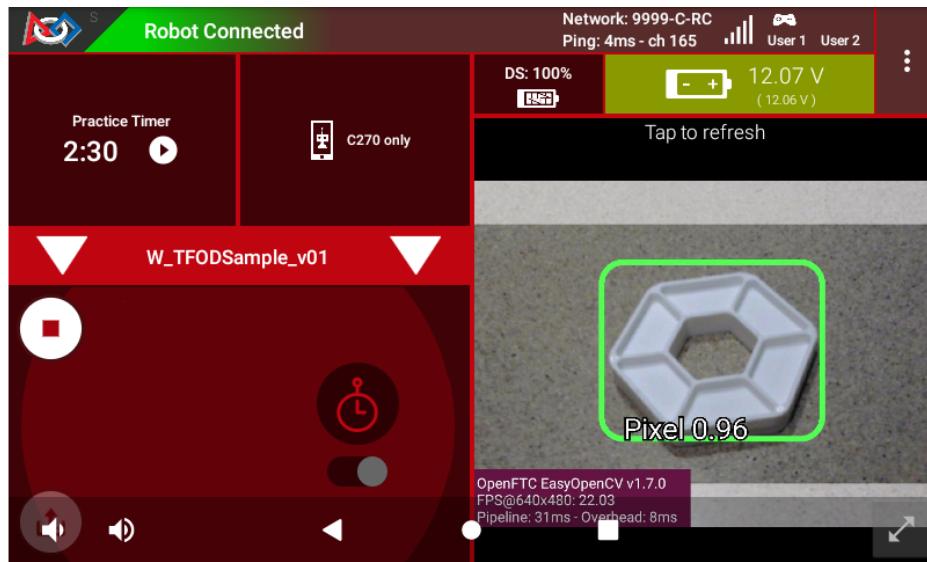


Fig. 57: Sample DS Camera Stream

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:

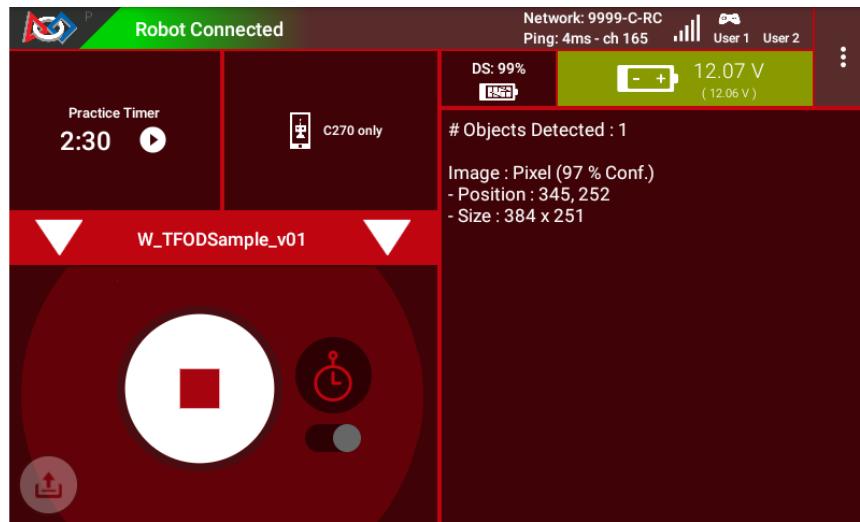


Fig. 58: Sample DS Telemetry



Fig. 59: Sample RC LiveView



Fig. 60: Sample Pixel

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.

## 4.7.8 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 `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.

## 4.7.9 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.

## 4.7.10 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.

#### 4.7.11 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.

#### 4.7.12 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.

#### 4.7.13 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](mailto:westsiderobotics@verizon.net)

## Chapter 5

---

### Vision Programming

Learning more about using vision

## 5.1 Computer Vision Overview

### 5.1.1 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.

### 5.1.2 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.

```

Camera Ready

==== (ID 0) Nemo
XYZ -6.6 24.9 -5.7 (inch)
PRY 2.0 0.1 4.9 (deg)
RBE 25.7 14.8 -12.8 (inch, deg, deg)

==== (ID 1) Jonah
XYZ -1.5 25.5 -5.7 (inch)
PRY 0.7 -0.0 5.0 (deg)
RBE 25.6 3.3 -12.6 (inch, deg, deg)

key:
XYZ = X (Right), Y (Forward), Z (Up) dist.
PRY = Pitch, Roll &Yaw (XYZ Rotation)
RBE = Range, Bearing & Elevation

```

Fig. 1: 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.

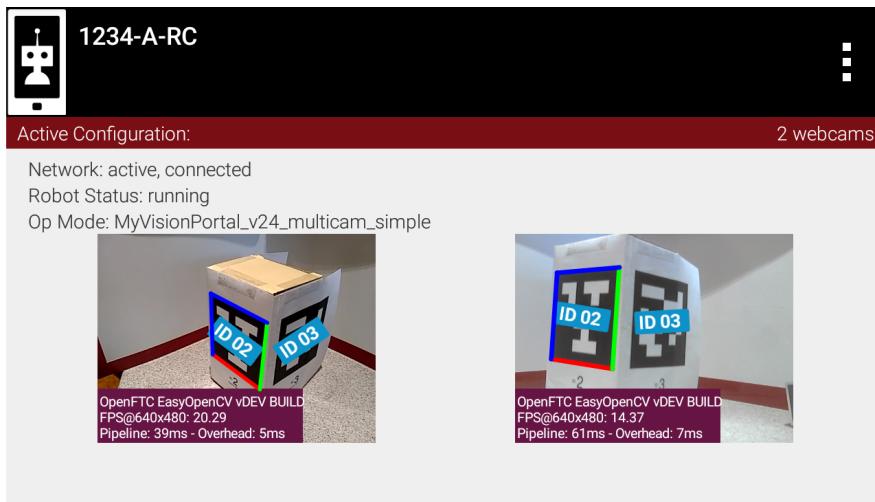


Fig. 2: AprilTags not in Tag Library detected, but no pose data available

### 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.

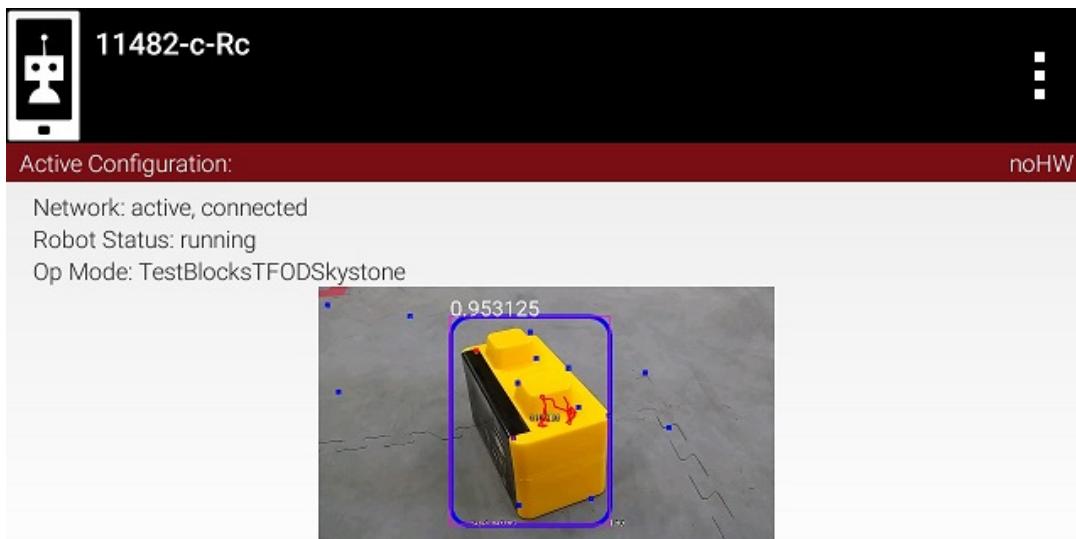


Fig. 3: TensorFlow can recognize actual objects (and not just 2D image targets).



Fig. 4: TensorFlow can be taught to distinguish between similar looking objects.

## 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.

## 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.

### 5.2.1 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.

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.

The screenshot shows the Javadoc interface for the RobotCore 7.0.0 API. The left sidebar lists 'All Classes' and 'Packages'. Under 'Packages', several packages are listed, including com.qualcomm.robotcore.eventloop, com.qualcomm.robotcore.eventloop.opmode, com.qualcomm.robotcore.exception, com.qualcomm.robotcore.factory, com.qualcomm.robotcore.hardware, com.qualcomm.robotcore.hardware.configuration, com.qualcomm.robotcore.hardware.configuration.annotati, com.qualcomm.robotcore.hardware.configuration.typecon, com.qualcomm.robotcore.hardware.usb, com.qualcomm.robotcore.hardware.usb.ftdi, and CameraControl. The 'CameraControl' package is circled in yellow. The main content area displays the 'RobotCore 7.0.0 API' documentation, specifically the 'Packages' section, which contains a table mapping package names to their descriptions.

Package	Description
com.qualcomm.robotcore.eventloop	RobotCore event loop library.
com.qualcomm.robotcore.eventloop.opmode	
com.qualcomm.robotcore.exception	RobotCore exception library.
com.qualcomm.robotcore.factory	
com.qualcomm.robotcore.hardware	RobotCore hardware library.
com.qualcomm.robotcore.hardware.configuration	
com.qualcomm.robotcore.hardware.configuration.annotations	

Fig. 5: RobotCore Javadoc API

## 5.2.2 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  $1/60 \times 1000 = 16$  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 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. 6: 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.

### 5.2.3 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:

## FTC Docs

- 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.

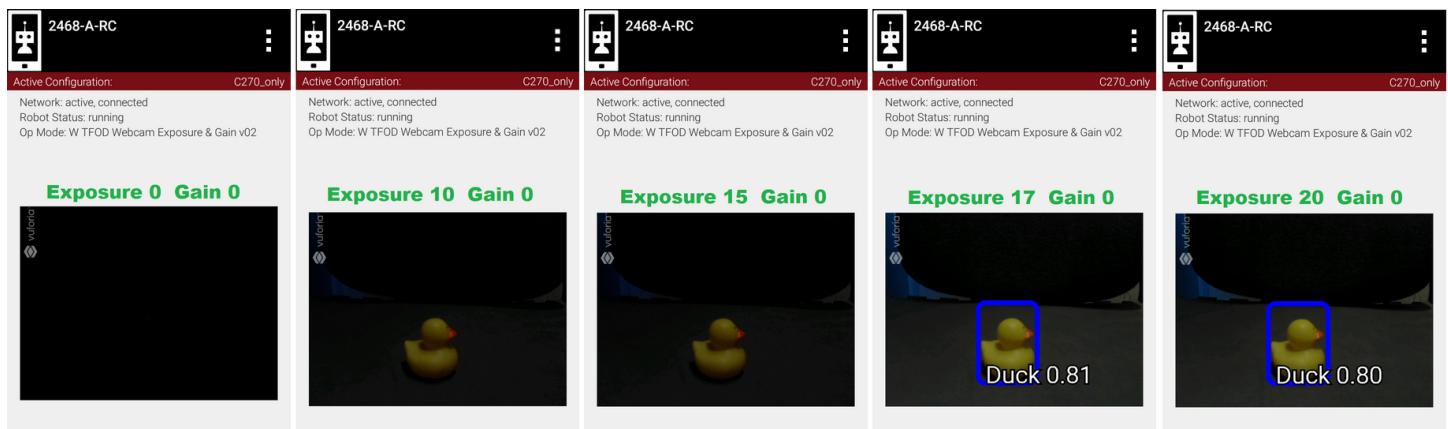


Fig. 7: Gain 0, Exp 0 -> 20

**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.

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.**

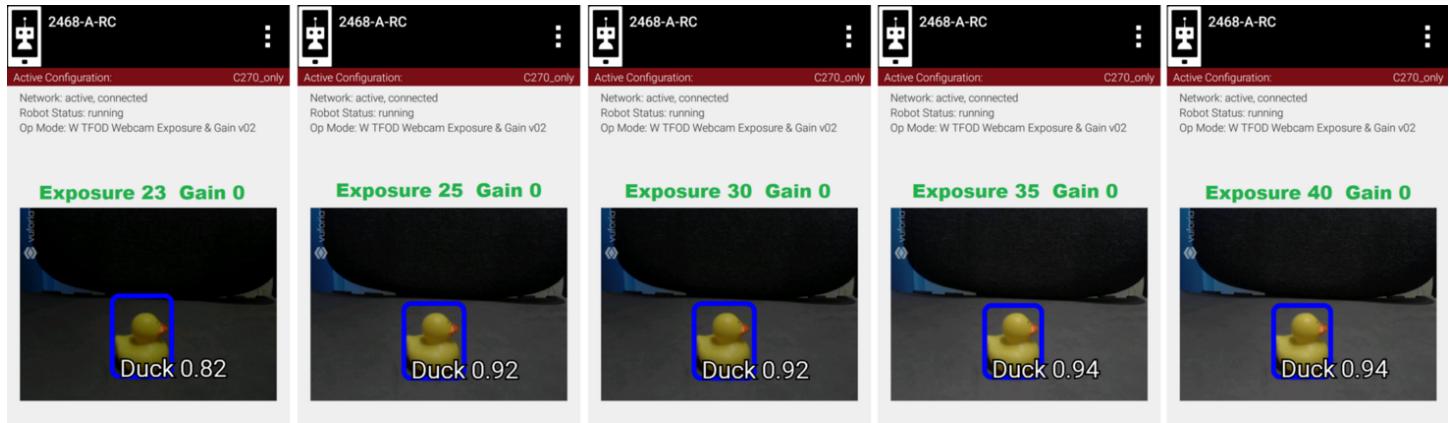


Fig. 8: Gain 0, Exp 23 -&gt; 40

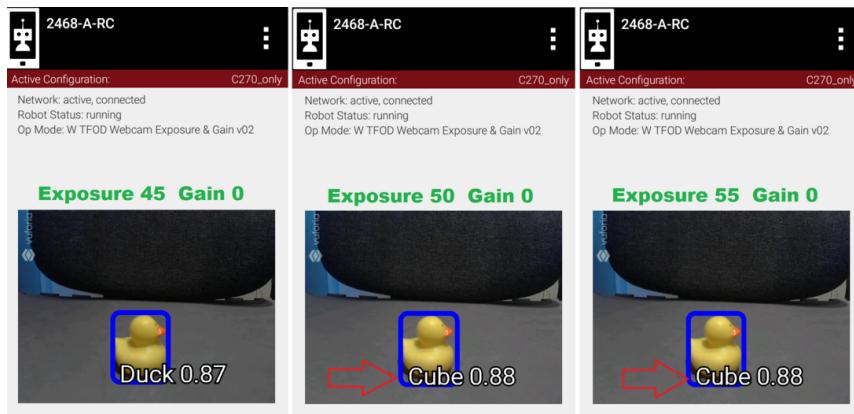


Fig. 9: Gain 0, Exp 45 -&gt; 55

<b>TFOD Duck Confidence, instant (&lt; 1 sec.)</b>						<b>Gain = 0</b>
<b>Exposure</b>	<b>Test Round</b>					
	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	
0	none	none	none	none	none	none
10	none	none	none	none	none	none
15	none	none	none	none	none	none
17	0.81	none	0.81	0.80	none	none
20	0.81	0.80	0.80	0.81	0.81	0.81
23	0.82	none	0.81	0.80	none	none
25	0.92	0.92	0.91	0.91	0.91	0.91
30	0.92	0.91	0.91	0.92	0.91	0.91
35	0.94	0.93	0.92	0.93	0.93	0.93
40	0.94	0.93	0.93	0.93	0.93	0.92
45	0.87	none	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. 10: Five readings at each exposure level

## 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?

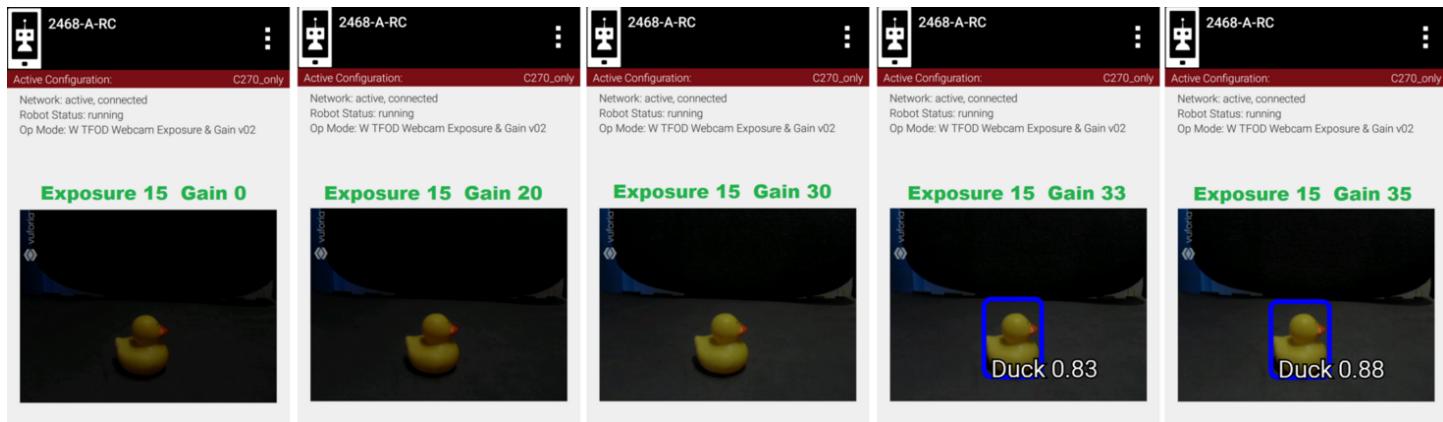


Fig. 11: Exp 15, Gain 000 -> 035

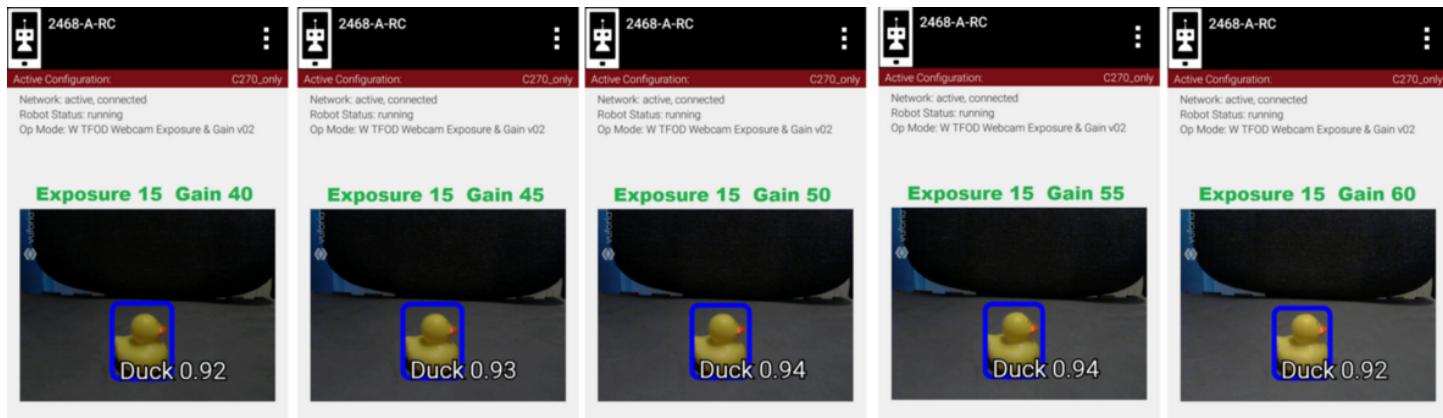


Fig. 12: Exp 15, Gain 040 -> 060

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.

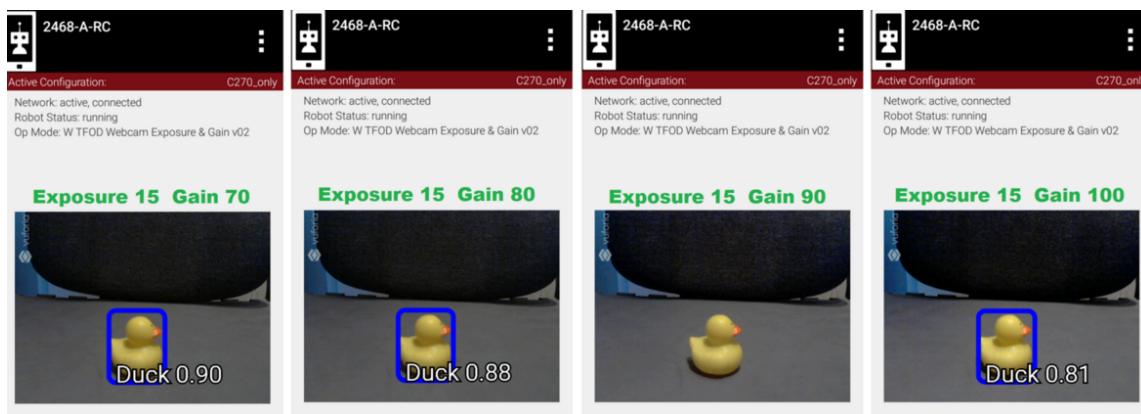


Fig. 13: Exp 15, Gain 070 -&gt; 100

TFOD Duck Confidence, instant (< 1 sec.)		Exposure = 15				
		Test Round				
Gain		1	2	3	4	5
0	none	none	none	none	none	none
20	none	none	none	none	none	none
30	none	none	none	none	none	none
33	0.84	0.83	0.84	0.82	0.82	0.82
35	0.88	0.87	0.86	0.88	0.87	0.87
40	0.91	0.91	0.91	0.91	0.91	0.91
45	0.93	0.92	0.92	0.92	0.92	0.92
50	0.93	0.93	0.93	0.93	0.93	0.92
55	0.94	0.94	0.93	0.94	0.94	0.94
60	0.94	0.93	0.94	0.94	0.94	0.94
70	0.90	0.90	0.91	0.90	0.91	0.91
80	0.88	0.85	none	0.88	0.88	0.83
90	none	none	none	none	none	none
100	0.81	Cube .85	Cube .83	Cube .83	Cube .83	Cube .88

Fig. 14: Five readings at each gain level

### Example 3: An odd preview

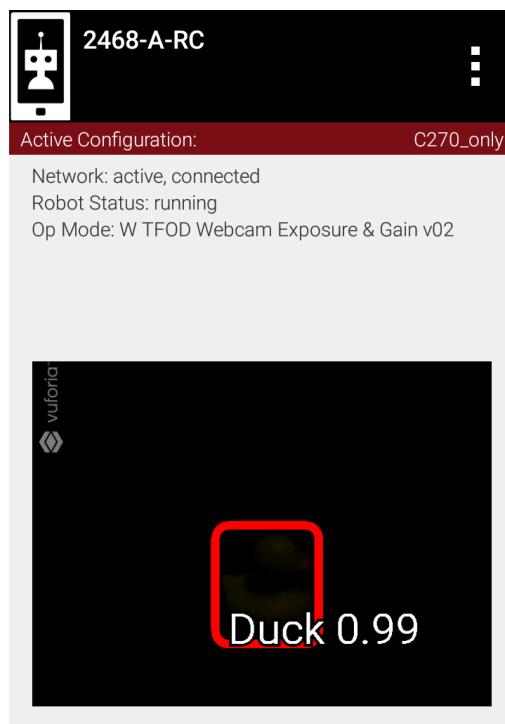


Fig. 15: Did TFOD make this recognition?

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.

#### 5.2.4 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.

## 5.2.5 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](#).

## 5.2.6 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.

### 5.2.7 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.

## 5.2.8 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.

## 5.2.9 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.

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;
```

(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 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();

        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);
    }
}

```

(continues on next page)

```

// 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;

    // 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.
}

```

(continues on next page)

(continued from previous page)

```

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();
    }

} // 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;

@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,

```

(continues on next page)

```

/*
 * 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:
 *   ... yIgIzTqZ4mlWjk9wd3cZ09T1axEqzuhxoGf00I2dRzKS4T0hQ8kT ...
 * 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

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 ***

```

(continues on next page)

(continued from previous page)

```

@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();

    // display exposure mode and starting values to user
    telemetry.addData("\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 ***

```

(continues on next page)

(continued from previous page)

```

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;

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

```

(continues on next page)

```

        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;
    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() {*

```
    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)

\*/

```

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.addData("Current 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);
        curWhiteBalanceTemp = Math.min(curWhiteBalanceTemp, maxWhiteBalanceTemp);
    }

    // update the color temperature setting
    wasTemperatureSet = myWBControl.setWhiteBalanceTemperature(curWhiteBalanceTemp);

    // display live feedback while user observes preview image
    telemetry.addData("Adjust temperature with blue X (cooler) & red B (warmer)");
}

```

(continues on next page)

(continued from previous page)

```

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;

@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

```

(continues on next page)

(continued from previous page)

```

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();
    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();
}

```

(continues on next page)

(continued from previous page)

```

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();

    sleep(100);

} // end main while() loop

} // end OpMode

// display Focus Modes supported by this webcam
private void checkFocusModes() {

```

(continues on next page)

```

        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](mailto: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;

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")
```

(continues on next page)

```

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).
         */
        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;
    }
}

```

(continues on next page)

```

// 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

    if (gamepad1.y) {
        curZoom += zoomIncrement;
    } else if (gamepad1.a) {
        curZoom -= zoomIncrement;
    }

    // ensure inputs are within webcam limits, if provided
    if (useLimits) {

```

(continues on next page)

```

        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.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

```

## 5.2.10 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](mailto:westsiderobotics@verizon.net)

## 5.3 Camera Calibration for FIRST Tech Challenge

### 5.3.1 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.

### 5.3.2 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.

### 5.3.3 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.

## Chapter 6

---

### Advanced Topics

Advanced Topics for Programmers

#### 6.1 Changing PID Coefficients

The REV Robotics Expansion Hub allows a user to change the PID coefficients used for closed loop motor control. The PID coefficients are channel and mode specific. Note that the Modern Robotics and Hitechnic DC motor controllers do not support adjustable PID coefficients.

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 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 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.PIDCoefficients;

/**
 * Created by tom on 9/26/17.
 * This assumes that you are using a REV Robotics Expansion Hub
 * as your DC motor controller. This op mode uses the extended/enhanced
 * PID-related functions of the DcMotorEx class. The REV Robotics Expansion Hub
 * supports the extended motor functions, but other controllers (such as the
 * Modern Robotics and Hitechnic DC Motor Controllers) do not.
 */

@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;
```

(continues on next page)

```

public static final double NEW_D = 0.2;

public void runOpMode() {
    // get reference to DC motor.
    // since we are using the 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;
import com.qualcomm.robotcore.hardware.PIDCoefficients;

/**
 * Created by tom on 9/26/17.
 * This assumes that you are using a REV Robotics Expansion Hub
 * as your DC motor controller. This op mode uses the extended/enhanced
 * PID-related functions of the DcMotorControllerEx class.
 * The REV Robotics Expansion Hub supports the extended motor controller
 * functions, but other controllers (such as the Modern Robotics and
 * Hitechnic DC Motor Controllers) do not.
 */

@Autonomous(name="Concept: Change PID Controller", group = "Examples")
public class ConceptChangePIDController extends LinearOpMode {

```

(continues on next page)

```

// 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 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();
    }
}
}

```

## 6.2 Changing PIDF Coefficients

The 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 Expansion Hub. If you need your changes to persist, consider modifying your OpMode to store state information on the 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 Expansion Hub
 * as your DC motor controller. This OpMode uses the extended/enhanced
 * PIDF-related functions of the DcMotorEx class. The REV Robotics Expansion Hub
 * supports the extended motor functions, but other controllers (such as the
 * deprecated Modern Robotics and Hitechnic DC Motor Controllers) do not.
 */

@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 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 pidf0rig = 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);
    }
}
```

(continues on next page)

```
// Re-read coefficients and verify change.
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 Expansion Hub
 * as your DC motor controller. This OpMode uses the extended/enhanced
 * PIDF-related functions of the DcMotorControllerEx class.
 * The REV Robotics Expansion Hub supports the extended motor controller
 * functions, but other controllers (such as the deprecated Modern Robotics
 * and Hitechnic DC Motor Controllers) do not.
 */

@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();
    }
}
```

(continues on next page)

```

    // Get a reference to the motor controller and cast it as an extended functionality.
    // We assume it's a 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 PIDF coefficients for the RUN_USING_ENCODER RunMode.
    PIDFCoefficients pidf0orig = 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",
            pidf0orig.p, pidf0orig.i, pidf0orig.d, pidf0orig.f);
        telemetry.addData("P,I,D,F (modified)", "%.04f, %.04f, %.04f, %.04f",
            pidfModified.p, pidfModified.i, pidfModified.d, pidfModified.f);
        telemetry.update();
    }
}
}

```

Note 1: As of SDK 7.0, the former PID-only methods are still available, but deprecated.

Note 2: the deprecated Modern Robotics and Hitechnic DC motor controllers do not support adjustable PID or PIDF coefficients.

## 6.3 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 of 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.

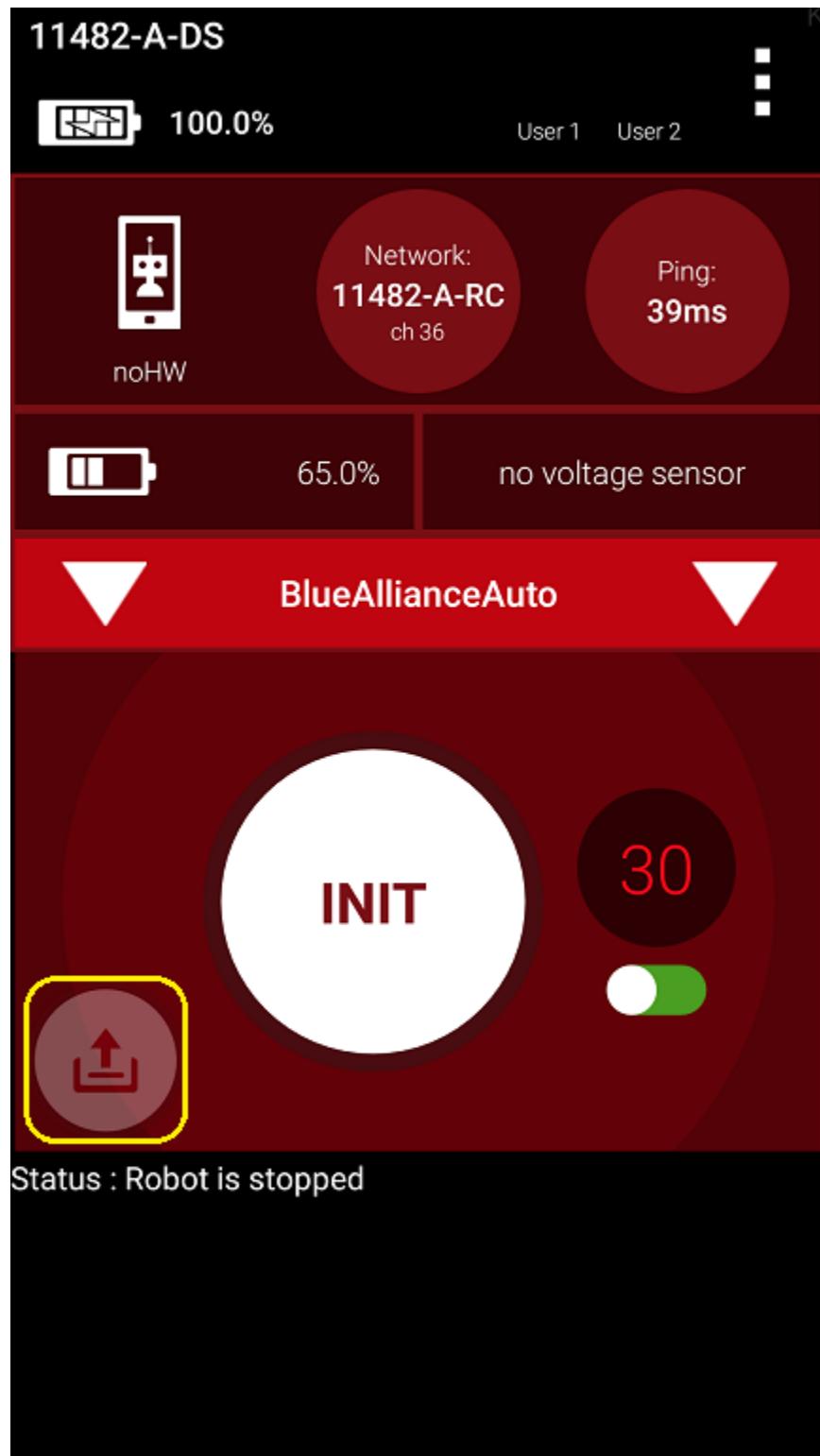


Fig. 1: The preselect button will appear once an autonomous op mode has been selected.



Fig. 2: The selected op mode must be designated as Autonomous in order for the preselect button to be visible.

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.

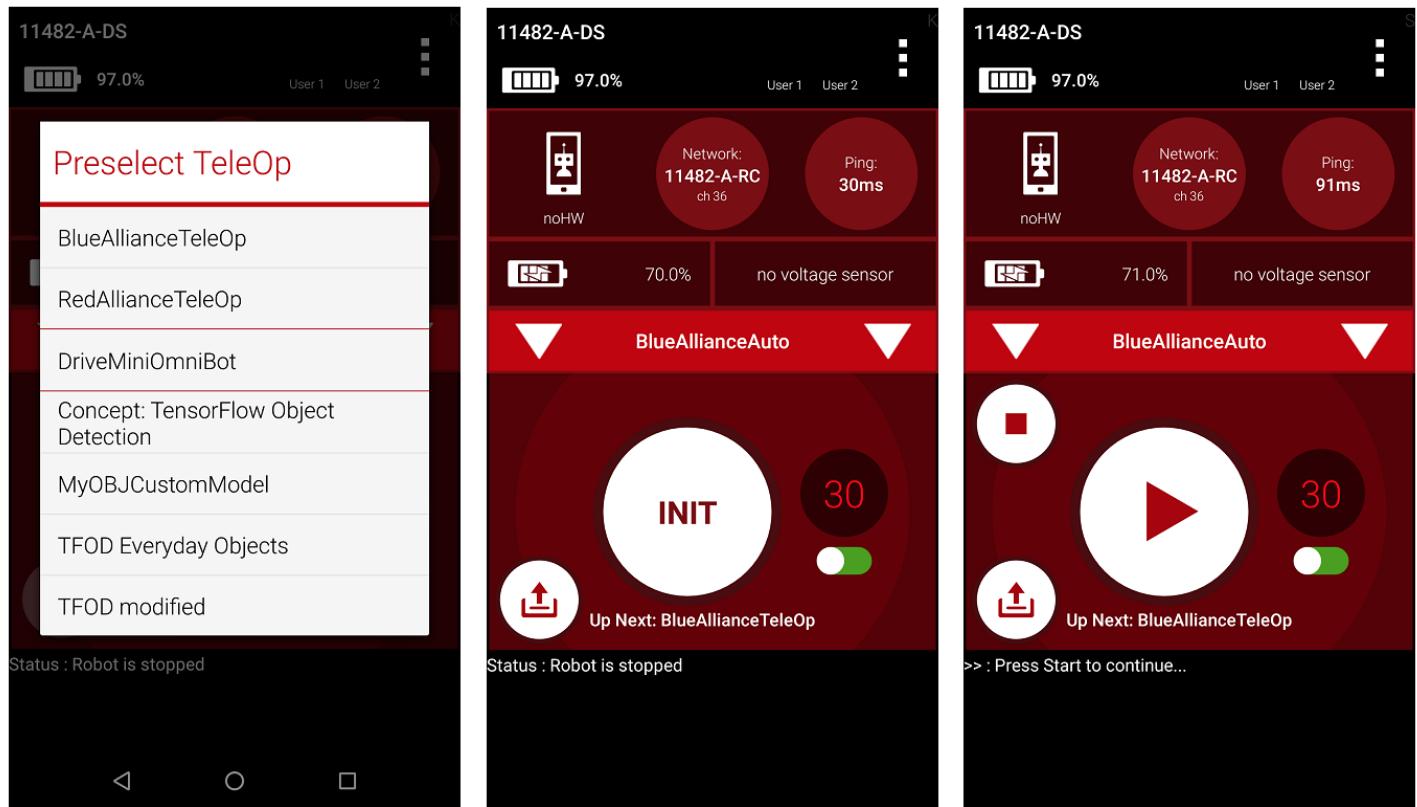


Fig. 3: The driver controlled op mode to be auto-loaded.

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 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.

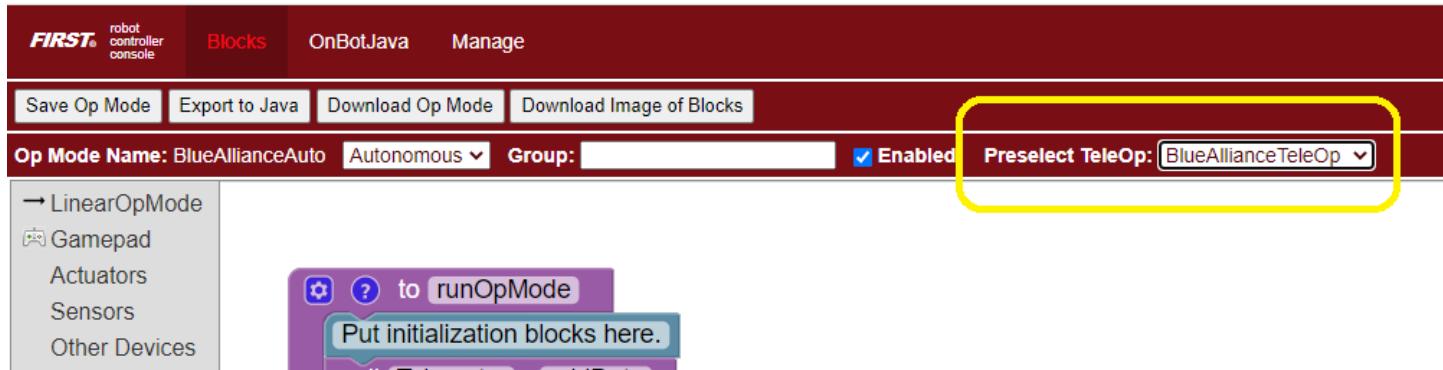


Fig. 4: You can preselect a teleop op mode mode using the Blocks editor.

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.

## 6.4 Custom Blocks (myBlocks)

### 6.4.1 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!

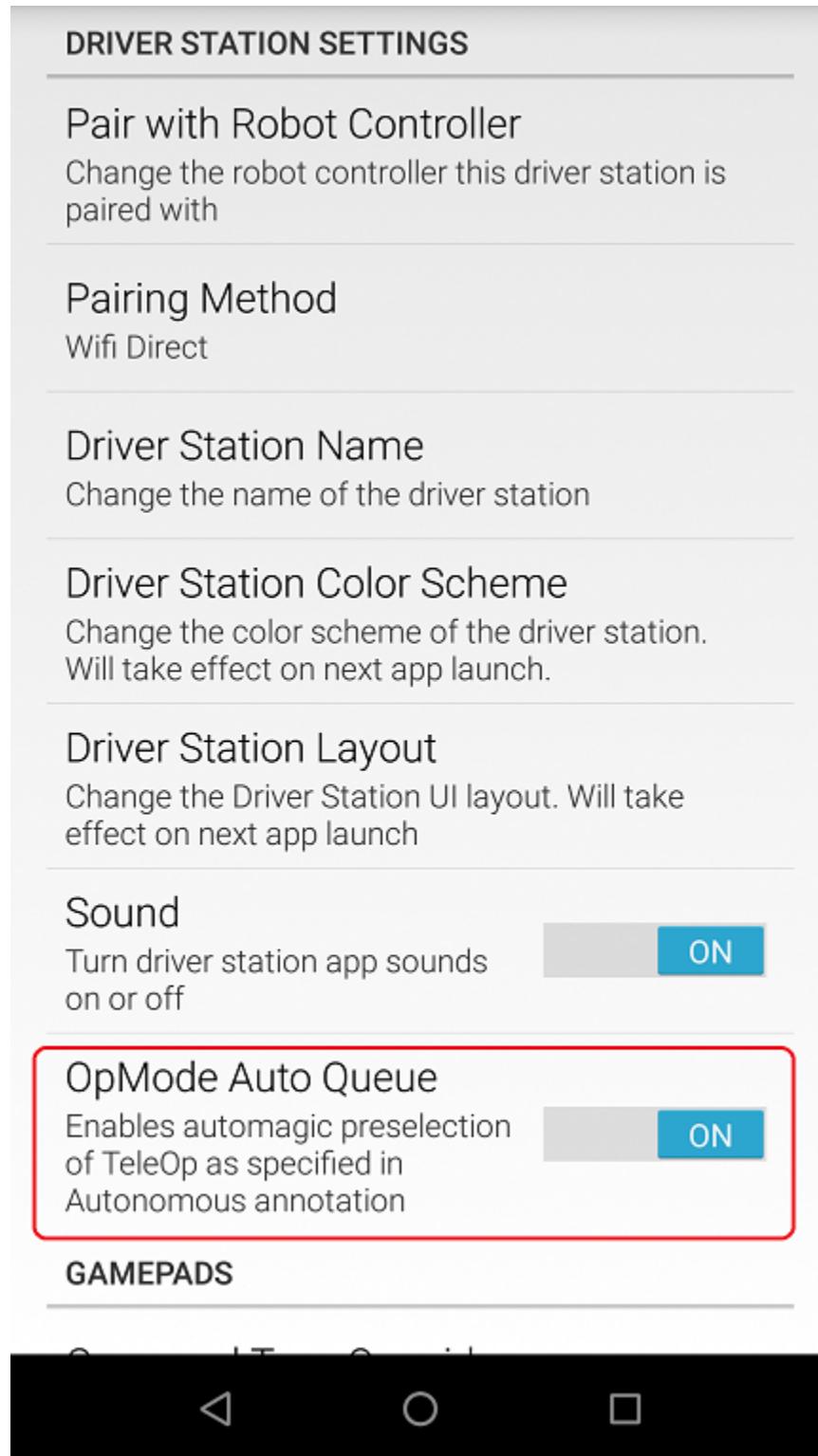


Fig. 5: If the OpMode Auto Queue option is enabled, the Driver Station will automatically load the preselectTeleOp op mode.

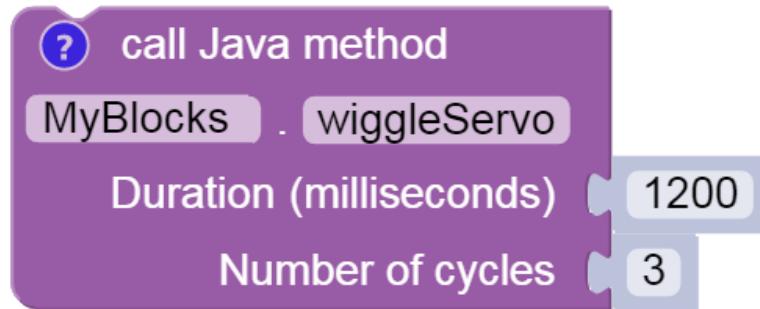


Fig. 6: sample myBlock: operate a servo, no value returned

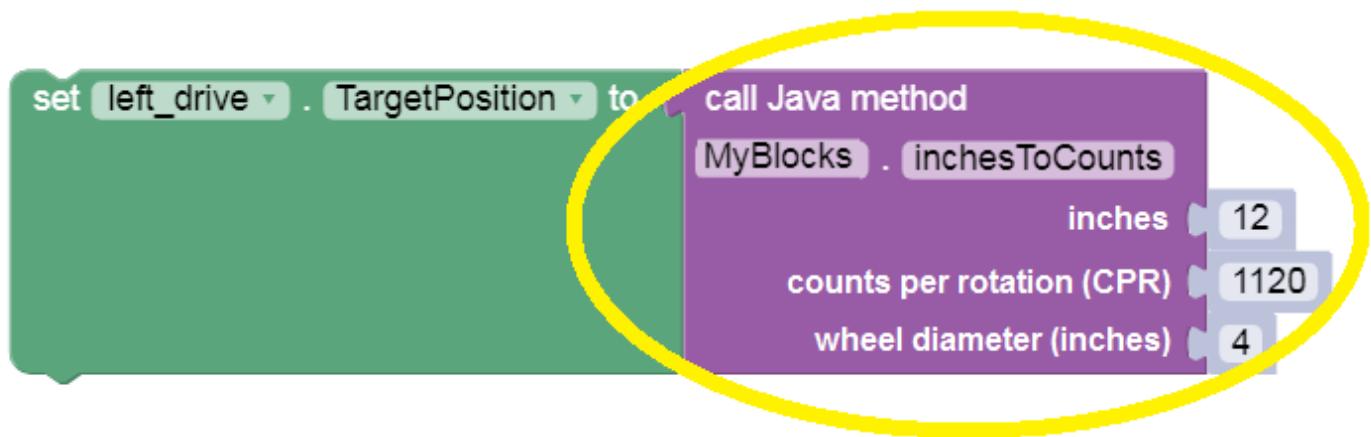


Fig. 7: sample myBlock: return encoder target value based on inputs

## 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.

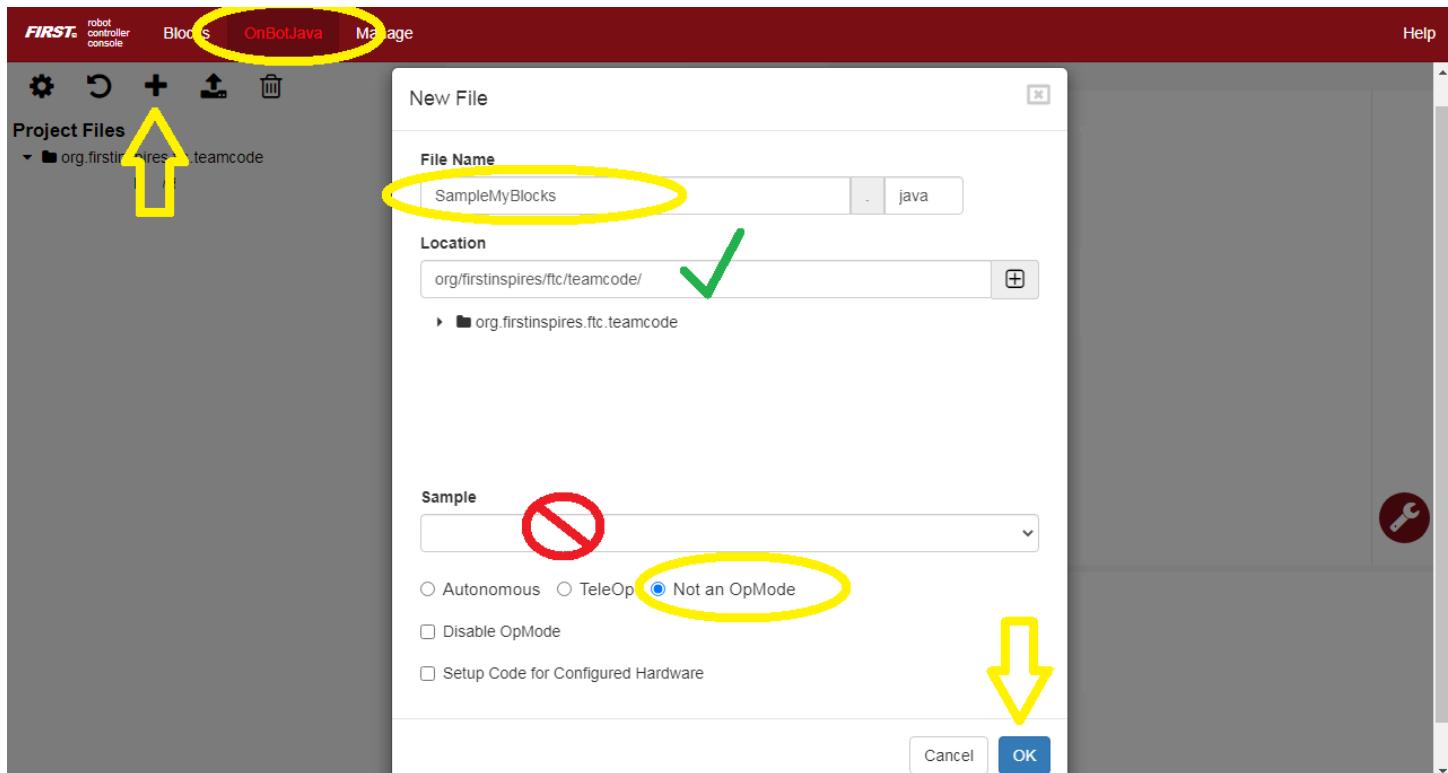
### 6.4.2 Simple Example: create myGreeting

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.

The screenshot shows the FTC Java code editor interface. At the top, there are tabs for "FIRST. robot controller console", "Blocks", "OnBotJava", and "Manage". Below the tabs is a toolbar with icons for settings, refresh, add, up, down, and delete. The left sidebar is titled "Project Files" and shows a single file: "SampleMyBlocks.java". The main area displays the following Java code:

```
</> SampleMyBlocks.java x  Welcome x
1 package org.firstinspires.ftc.teamcode;
2
3
4 public class SampleMyBlocks {
5
6     // todo: write your code here
7 }
```

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.

The screenshot shows the FTC Java code editor interface. The "SampleMyBlocks.java" file is open. The code now includes the `extends` keyword:

```
</> SampleMyBlocks.java x  Welcome x
1 package org.firstinspires.ftc.teamcode;
2
3 import org.firstinspires.ftc.robotcore.external.BlocksOpModeCompanion;
4
5
6 public class SampleMyBlocks extends BlocksOpModeCompanion {
7
8     // todo: write your code here
9 }
```

A yellow oval highlights the word "extends" in the line "public class SampleMyBlocks extends BlocksOpModeCompanion {".

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`.

```
<> SampleMyBlocks.java x  Welcome x
1 package org.firstinspires.ftc.teamcode;
2
3 import org.firstinspires.ftc.robotcore.external.BlocksOpModeCompanion;
4 import org.firstinspires.ftc.robotcore.external.ExportToBlocks;
5
6
7 public class SampleMyBlocks extends BlocksOpModeCompanion {
8
9     @ExportToBlocks (
10         comment = "Here is a greeting for you.",
11         tooltip = "Greet a person or group.",
12         parameterLabels = {"Recipient"}
13     )
14     public static String myGreeting (String greetingRecipient) {
15         return ("Hello " + greetingRecipient + "!");
16     }
17
18 }
```

Build started at Thu Nov 05 2020 17:05:17 GMT-0800 (Pacific Standard Time)

Build SUCCESSFUL!

Build finished in 0.8 seconds

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)

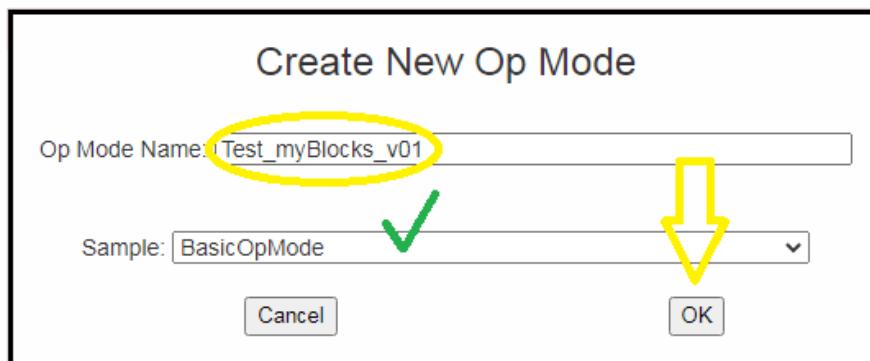
```

    tooltip = "Greet a person or group.",
    parameterLabels = {"Recipient"}
)
public static String myGreeting (String greetingRecipient) {
    return ("Hello " + greetingRecipient + "!");
}
}

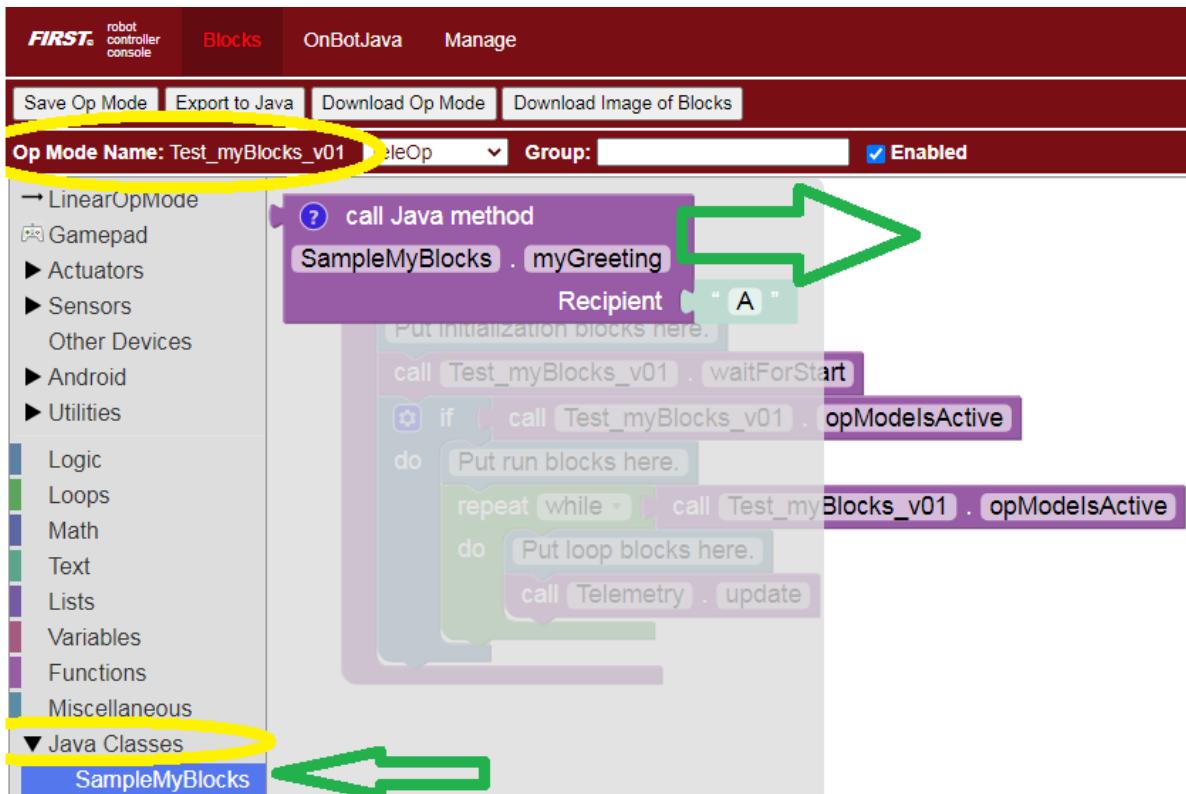
```

### 6.4.3 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.

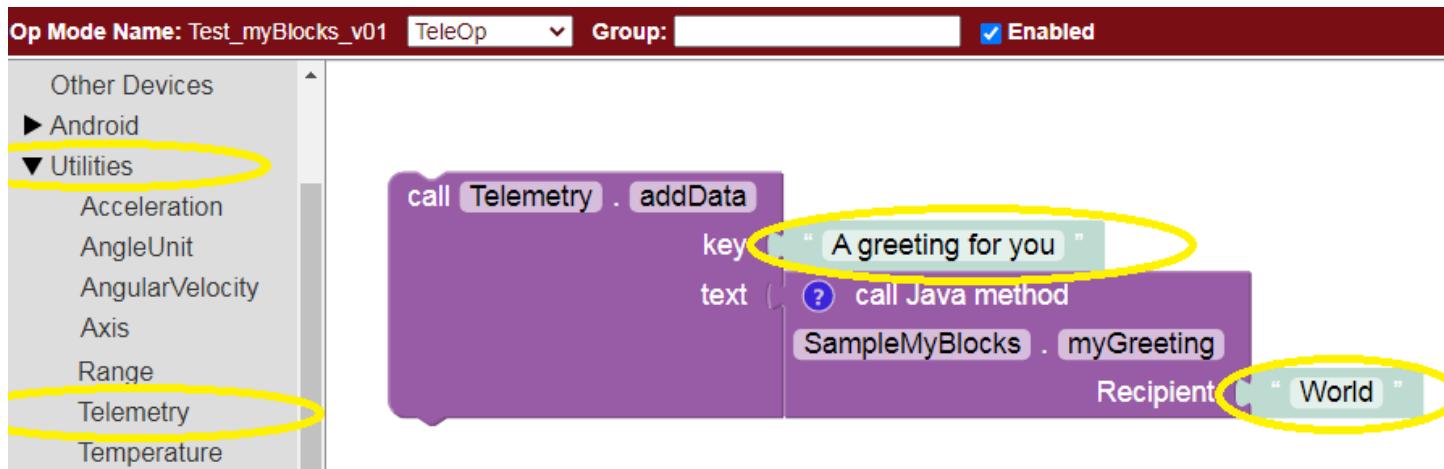


## FTC Docs

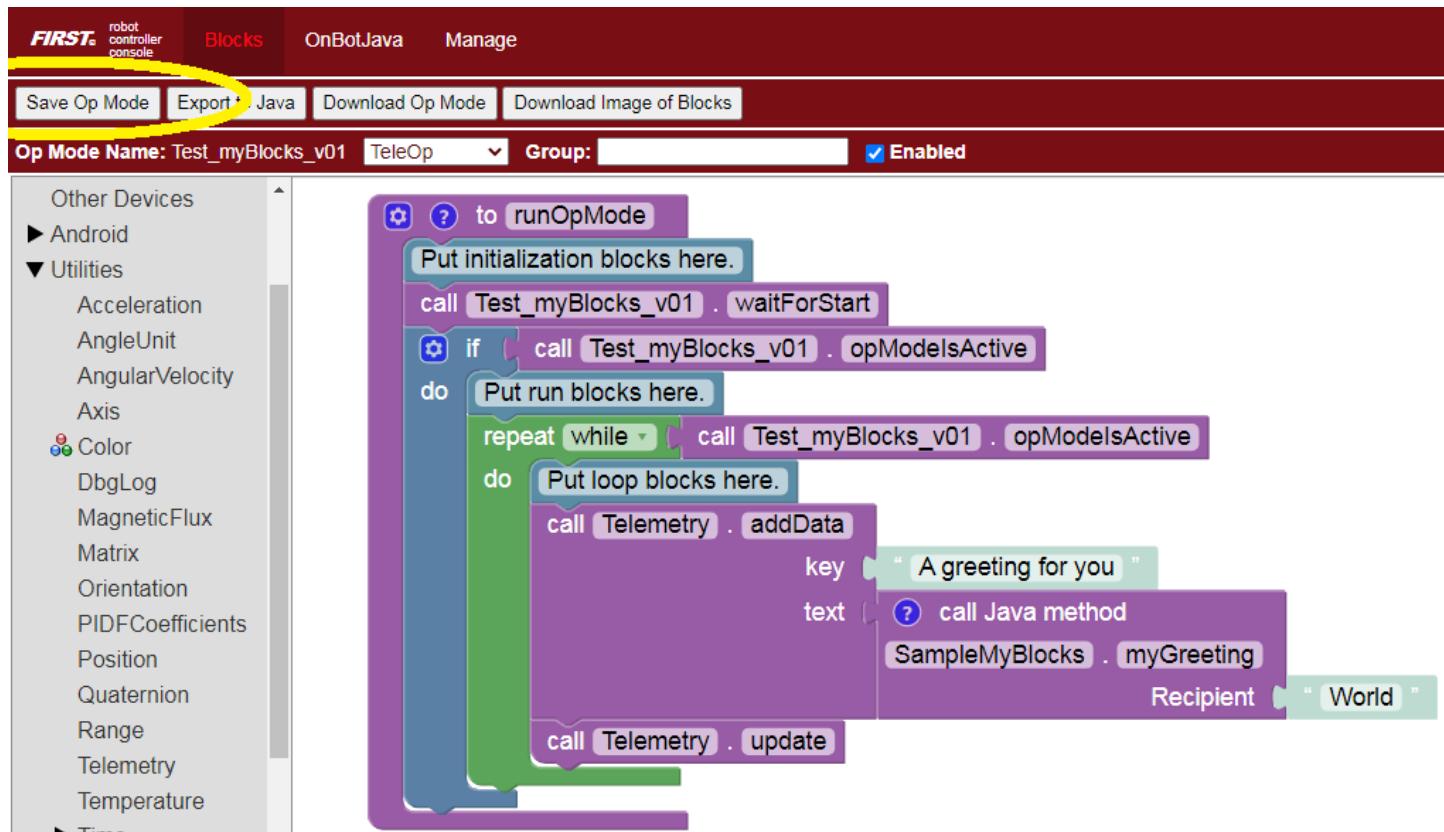
This myBlock has one grey input field or **socket**, containing the letter A to indicate a String or text input. Type the greeting recipient, **World**.

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).

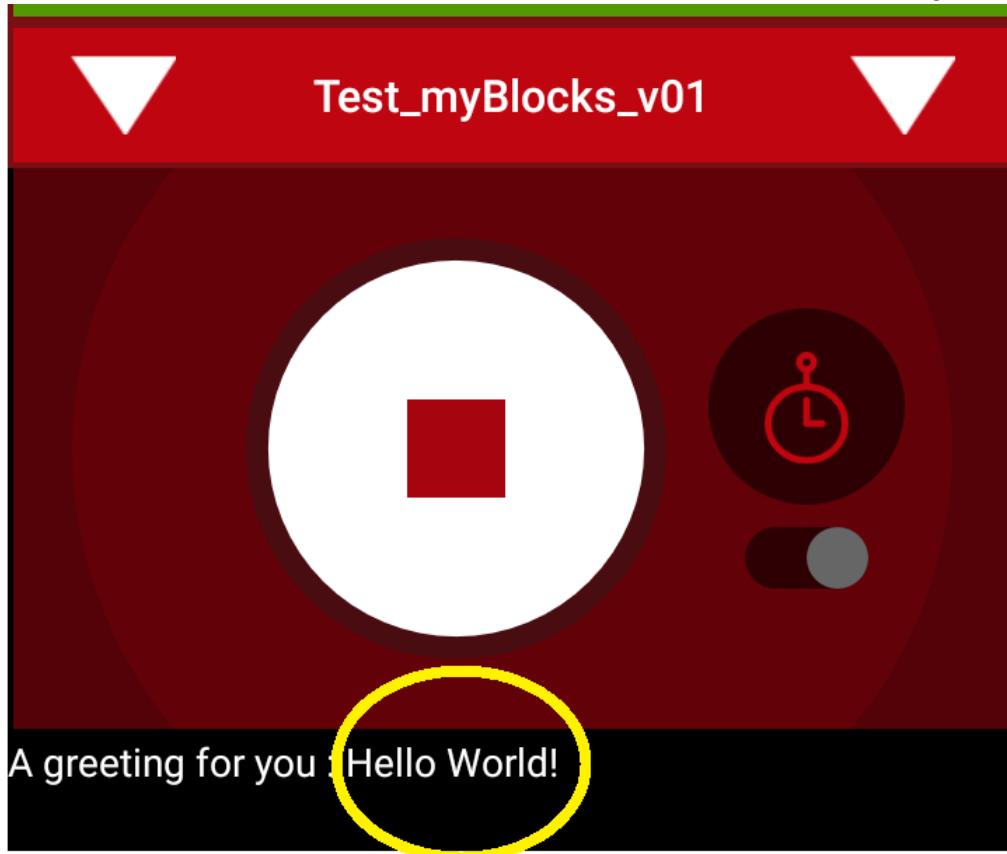
In the key socket, type A greeting for you. At the text socket, drag and connect your new myBlock. The myBlock's **text output** will be read and displayed by the **text** version of the **Telemetry.addData** Block.



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.



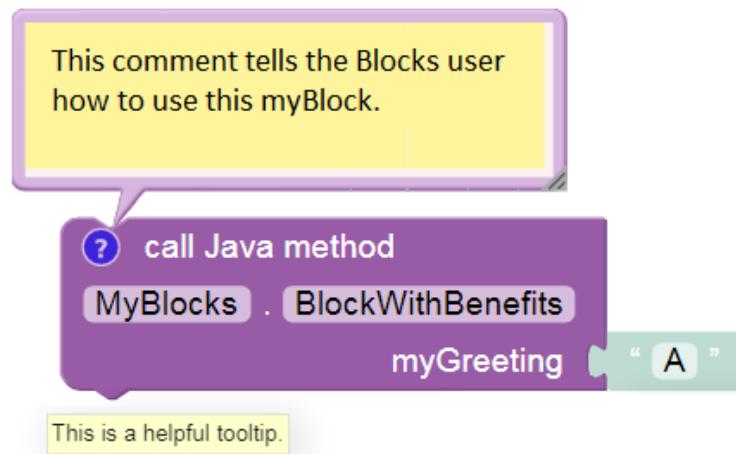
**Congratulations!** You are now an OnBot Java programmer and myBlocks creator.

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!

#### 6.4.4 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!

## 6.4.5 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 {

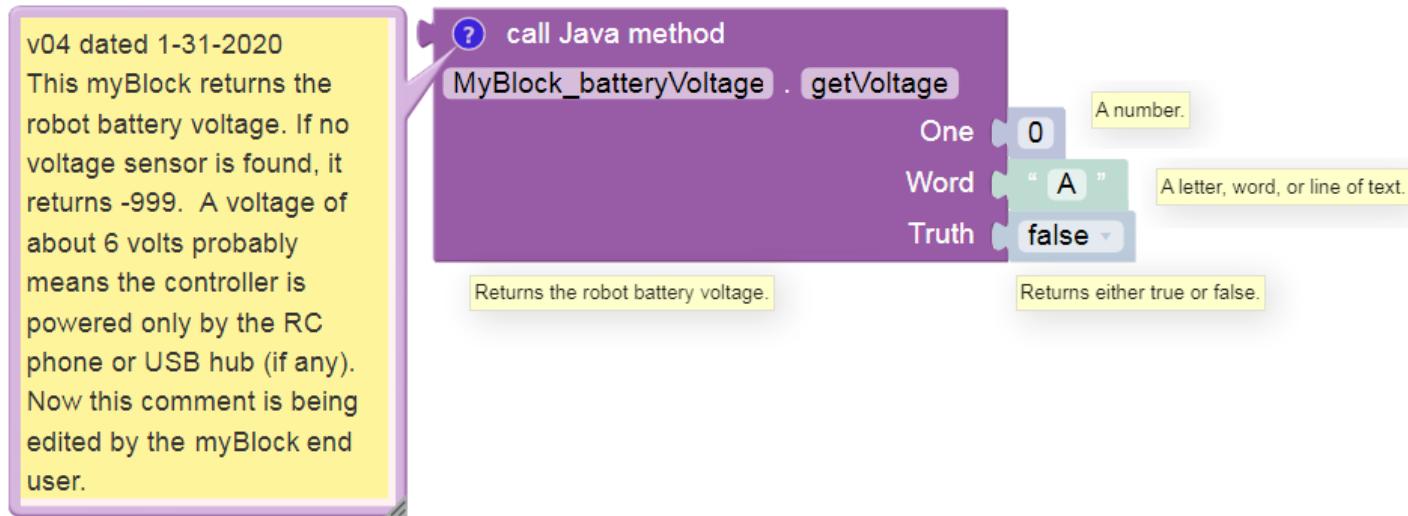
    static double batteryVoltage = -999;

    @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.

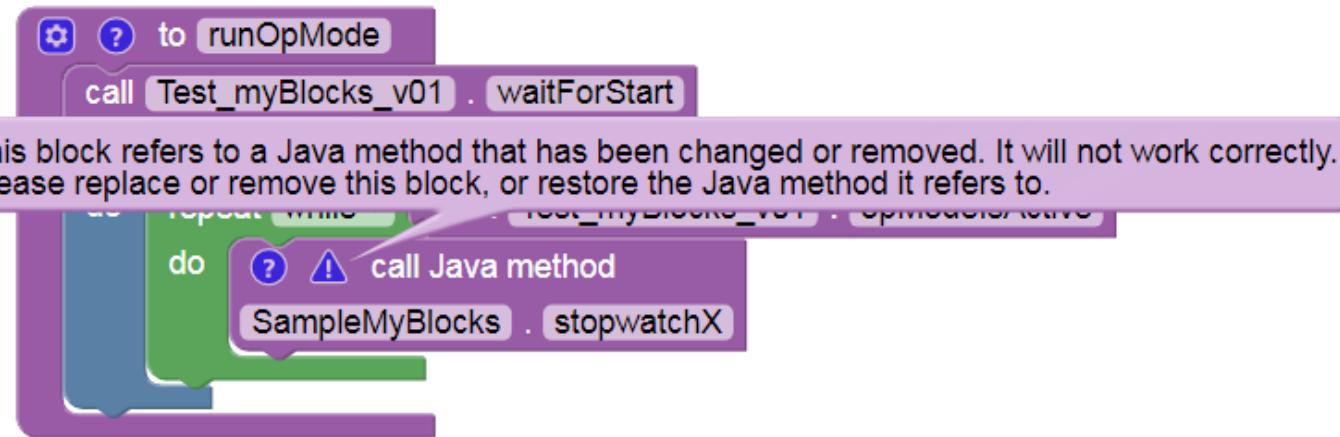
Programming tip: In this example, the variable batteryVoltage is declared and initialized **outside** the method, and thus could be used by other methods in this class.

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.

#### 6.4.6 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!

#### 6.4.7 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 ✘
1 package org.firstinspires.ftc.teamcode;
2
3 import org.firstinspires.ftc.robotcore.external.BlocksOpModeCompanion;
4 import org.firstinspires.ftc.robotcore.external.ExportToBlocks;
5 import com.qualcomm.robotcore.hardware.Servo;
6
7 public class SampleMyBlocks extends BlocksOpModeCompanion {
8
9     @ExportToBlocks (
10         comment = "Move a conventional servo back and forth. Assumes servo starts" +
11         " from position 0. Servo name must be in the active configuration.",
12         tooltip = "Wiggle a user-designated servo.",
13         parameterLabels = {"Servo name", "Duration (milliseconds)", "Number of cycles"}
14     )
15     public static void wiggleServo (String servoName, int duration, int cycles) {
16
17         Servo myServo = hardwareMap.get(Servo.class, servoName);
18
19         // count up to 'cycles' AND while opMode was not stopped
20         for (int i = 0; i < cycles && linearOpMode.opModeIsActive(); i++) {
21
22             myServo.setPosition(0.5);                      // move servo clockwise
23             linearOpMode.sleep(duration);                // wait for 'duration'
24             myServo.setPosition(0);                       // move servo counterclockwise
25             linearOpMode.sleep(duration);                // wait for 'duration'
26         }
27     } // end method wiggleServo()
28 } // end class SampleMyBlocks
```



#### Example Code

SampleMyBlocks\_v01.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)

```

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++) {

            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_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**).

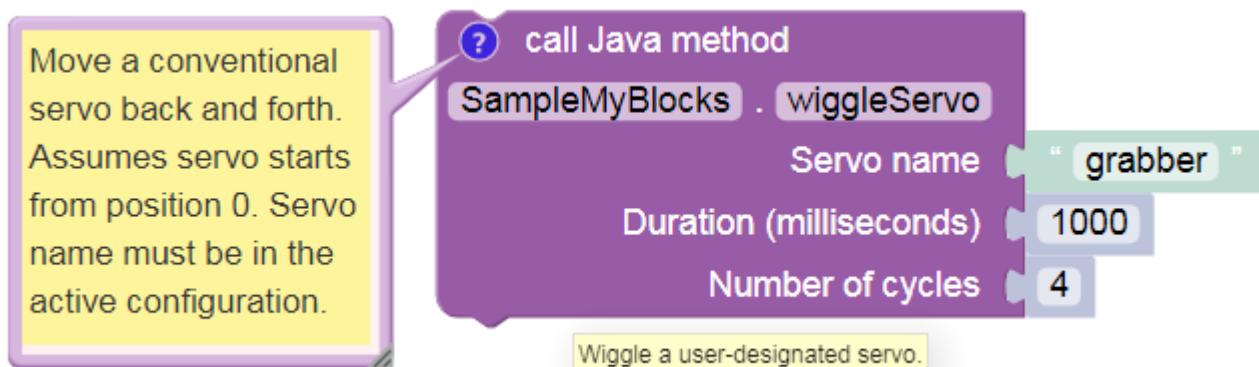
Line 17 shows how to access **hardwareMap**, the configured devices list provided from **BlocksOpModeCompanion**. That single line of Java does this: - declare a new variable called **myServo**, of type (class) **Servo** - **get** the properties (methods and variables) of the named servo from **hardwareMap** - 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.



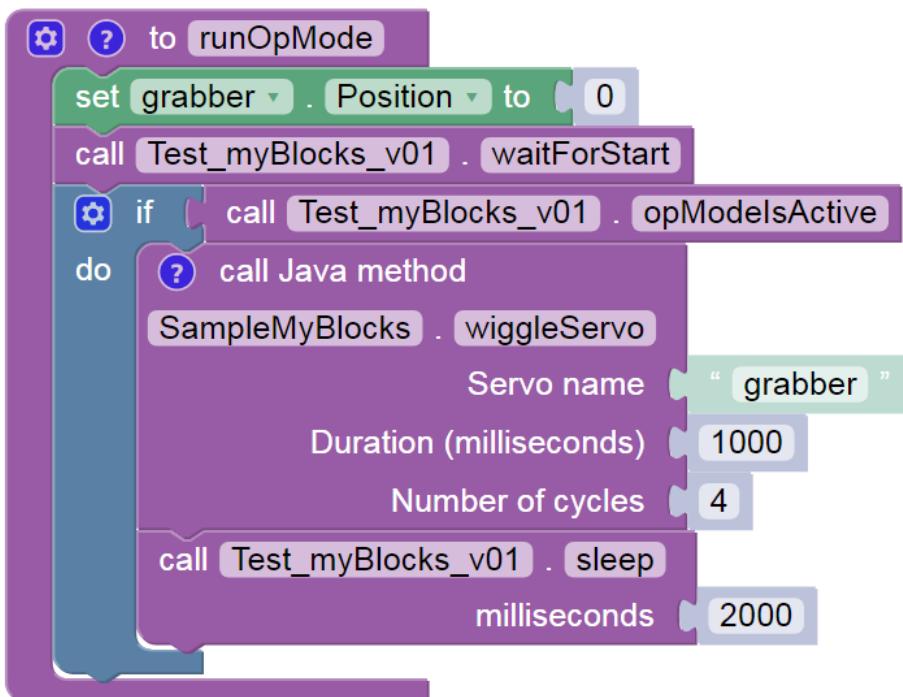
As an alternate, you could 'hard-code' the servo's name directly into the Java method, instead of the Blocks user entering the servo name as a parameter.

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)

```

// 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-designed 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)

```
}
```

```
// end class SampleMyBlocks_v02
```

### 6.4.8 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;
}
```

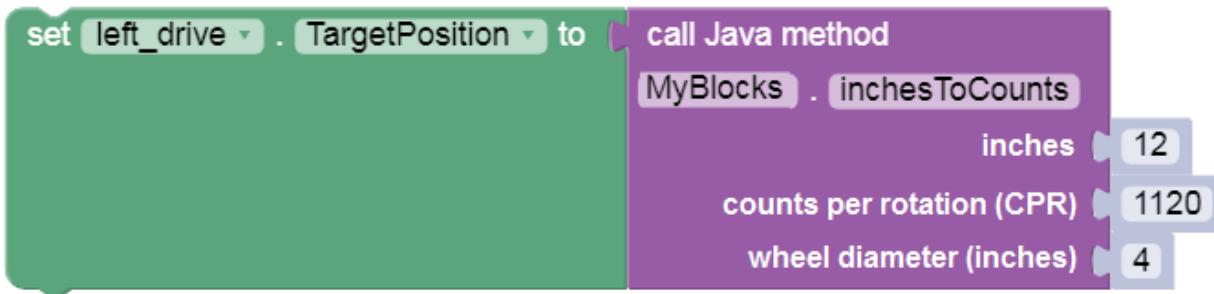
This method takes three inputs from the Blocks user, and **returns** one output (of type **int** or **integer**) to the regular Block that **calls** the myBlock.

---

**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

## 6.4.9 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

```
/*
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)

```

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

```

#### 6.4.10 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.

```

7 public class SampleMyBlocks extends BlocksOpModeCompanion {
8
9     @ExportToBlocks (
10         comment = "FOR ROBOT A ONLY. Enter inches to drive.",
11         tooltip = "Robot A convert inches to encoder counts",
12         parameterLabels = "Drive Distance (inches)"
13     )
14     public static int inchesToCountsRobotA (double inchesToDriveA) {
15         final double WHEEL_DIAMETER_A = 4.0;
16         final double COUNTS_PER_ROTATION_A = 1120;
17         int countsToDriveA = calculateCounts (inchesToDriveA, COUNTS_PER_ROTATION_A, WHEEL_DIAMETER_A);
18         return countsToDriveA;
19     }
20
21     @ExportToBlocks (
22         comment = "FOR ROBOT B ONLY. Enter inches to drive.",
23         tooltip = "Robot B convert inches to encoder counts",
24         parameterLabels = "Drive Distance (inches)"
25     )
26     public static int inchesToCountsRobotB (double inchesToDriveB) {
27         final double WHEEL_DIAMETER_B = 3.0;
28         final double COUNTS_PER_ROTATION_B = 537.6;
29         int countsToDriveB = calculateCounts (inchesToDriveB, COUNTS_PER_ROTATION_B, WHEEL_DIAMETER_B);
30         return countsToDriveB;
31     }
32
33     // This method is NOT a myBlock; it is called by other methods.
34     private static int calculateCounts (double inchesToDrive, double countsPerWheelRotation, double wheelDiameter) {
35         double circumference = wheelDiameter * Math.PI;
36         double rotations = inchesToDrive / circumference;
37         double encoderCounts = rotations * countsPerWheelRotation;
38         return (int) encoderCounts;
39     }
40 }
41 } // end of class SampleMyBlocks

```



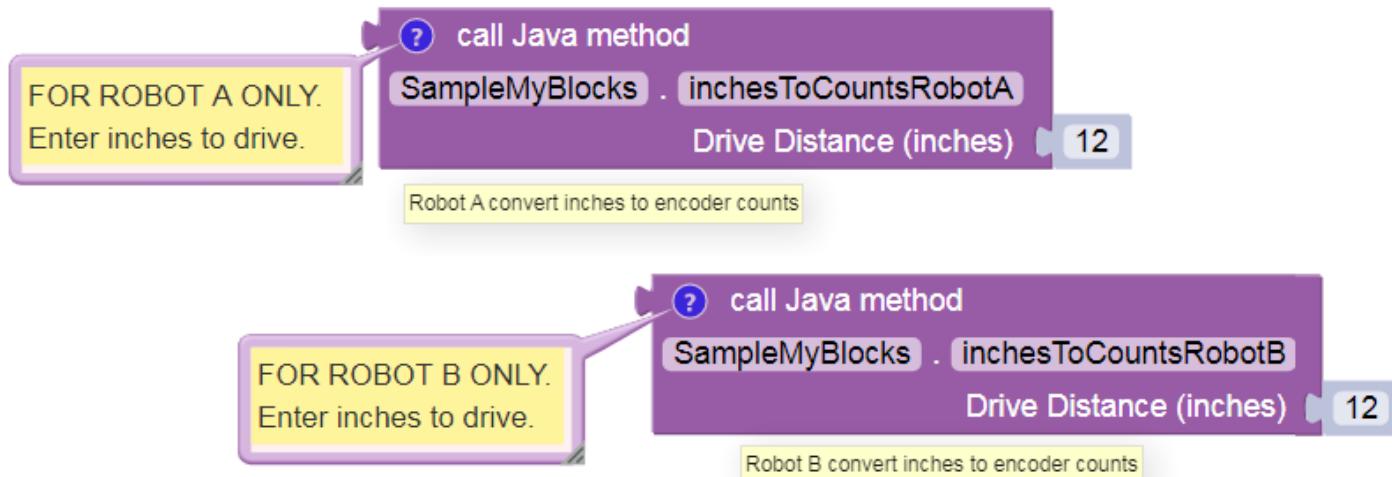
Line 34 shows the shared method that is **not** a myBlock. Simply omit the annotation @ExportToBlocks. The keyword **private** means the method can be called only from inside the same class. Use this whenever possible.

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)

```

// 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
}

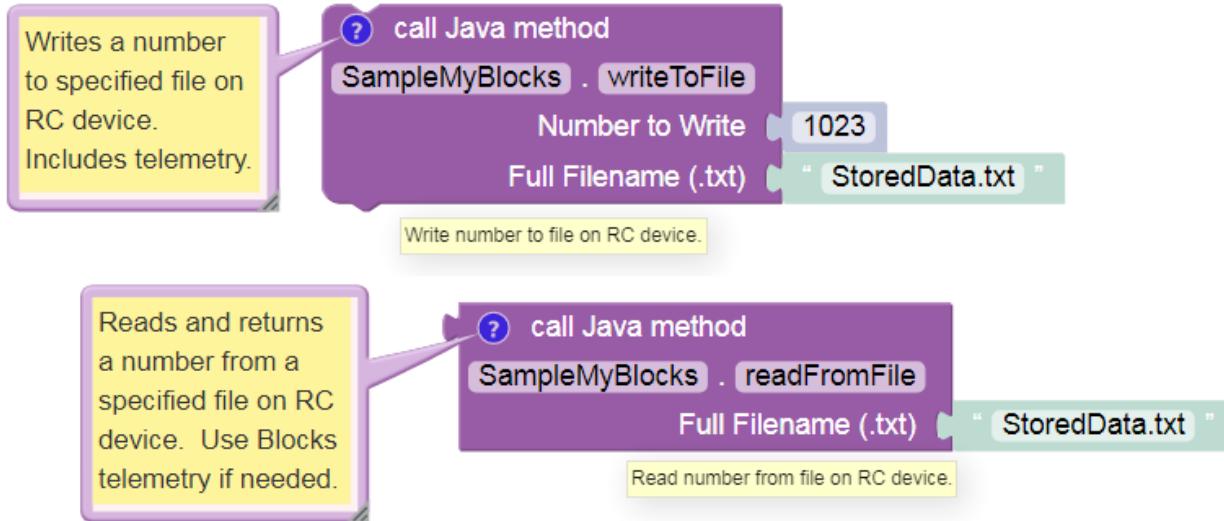
// end of class SampleMyBlocks_v04

```

### 6.4.11 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:

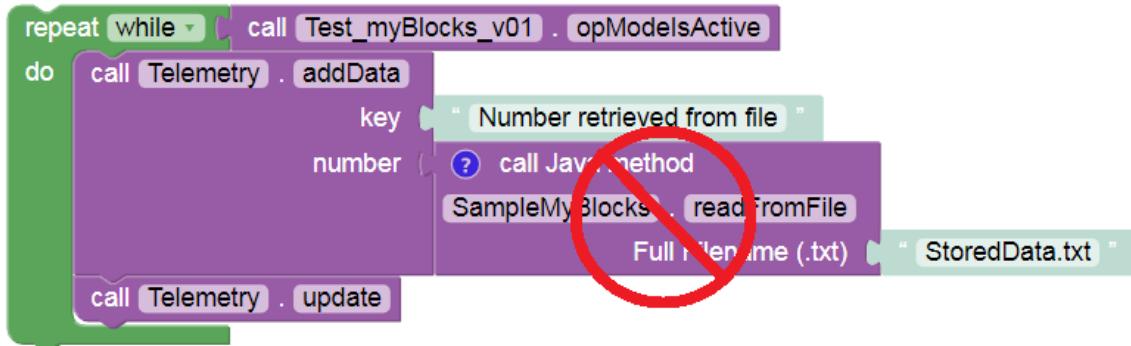
- Autonomous passes information to TeleOp. For example, what was the latest value of a sensor or encoder?
- A special **set-up OpMode** allows gamepad input to choose an autonomous strategy and adjust key parameters. The robot could then be idle for a long time, even turned off. When the match begins, 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.

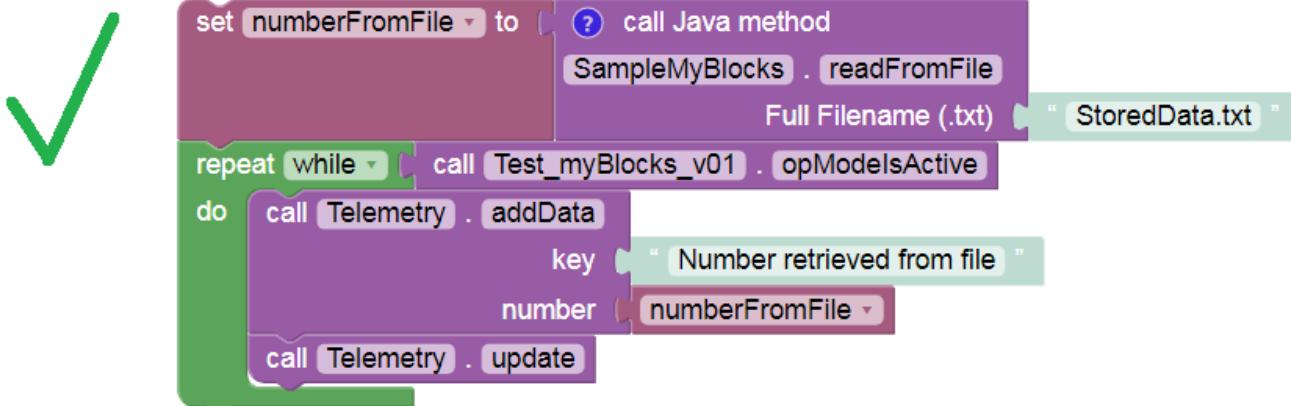
Programming tip: Instead of memorizing every possible Java command, programmers often study and modify existing code for a similar task. Unfamiliar commands are explored with an internet search, reference book, at the [Javadoc reference](#), or at the official [Oracle Javadoc](#).

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)

*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)

```

@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()

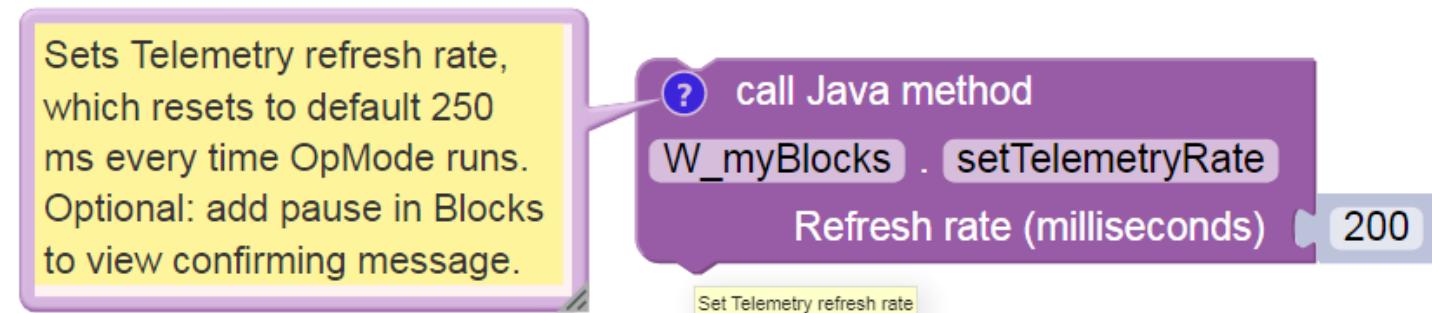
} // end of class SampleMyBlocks_v05

```

#### 6.4.12 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:

```

36     // This method has 1 input and no outputs (keyword void).
37     public static void setTelemetryRate (int myRate) {
38
39         // Get and store the existing (default) minimum interval between
40         // Telemetry transmissions from Robot Controller to Driver Station.
41         int oldRate = telemetry.getMsTransmissionInterval();
42
43         // Set the minimum interval, provided by the Blocks user.
44         telemetry.setMsTransmissionInterval(myRate);
45
46         // For confirmation, get and store the updated interval.
47         int newRate = telemetry.getMsTransmissionInterval();
48
49         telemetry.addData("TELEMETRY REFRESH RATE in milliseconds", null);
50         telemetry.addData("Default/previous rate", oldRate);
51         telemetry.addData("Requested rate", myRate);
52         telemetry.addData("Confirmed new rate", newRate);
53         telemetry.update();           // display info on Driver Station screen
54
55     } // 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

```
/*
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)

```

// 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 attached 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)

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.firstrainspires.ftc.teamcode;

**import** org.firstrainspires.ftc.robotcore.external.BlocksOpModeCompanion;  
**import** org.firstrainspires.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)

```

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

```

#### 6.4.13 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](mailto: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.

## 6.4.14 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](mailto:westsiderobotics@verizon.net)

## 6.5 External Libraries in OnBot Java and Blocks

### 6.5.1 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.*

### 6.5.2 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.

### 6.5.3 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.

### 6.5.4 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. 8: 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.

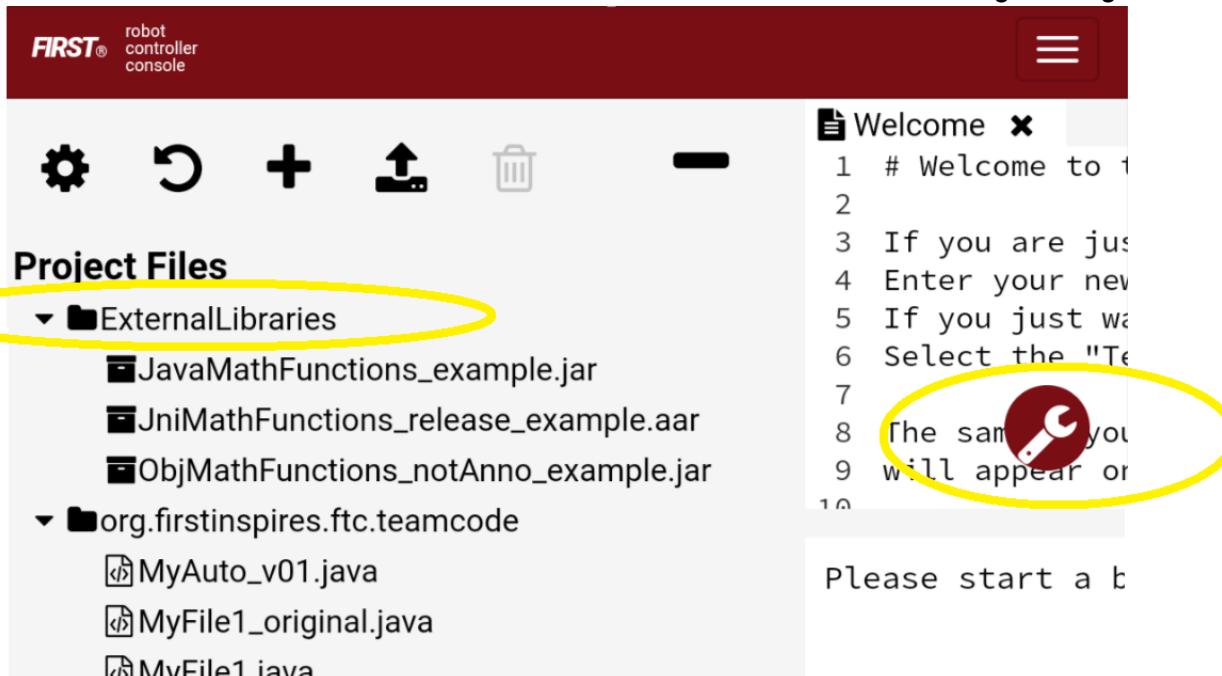


Fig. 9: ExternalLibraries folder and Build Everything icon

### 6.5.5 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.

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.

### 6.5.6 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.

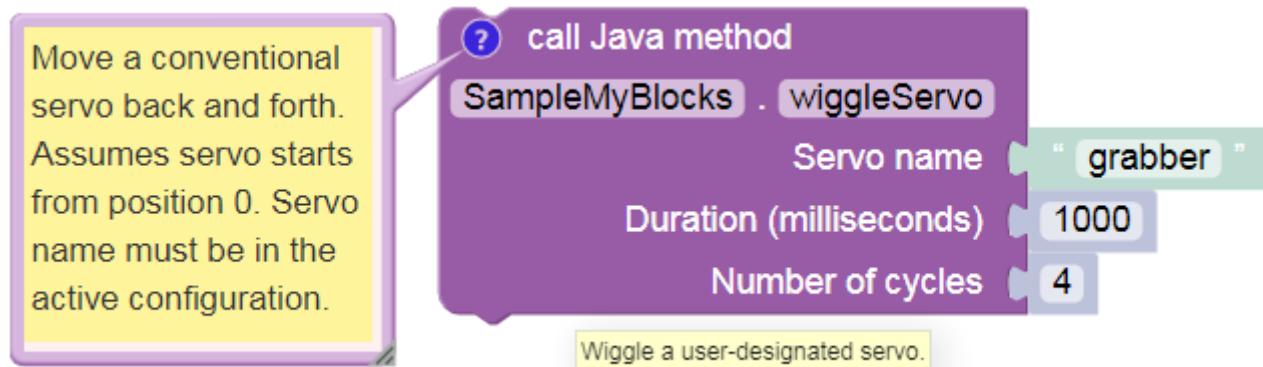


Fig. 10: myBlock documented using comment, tooltip and input labels

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!

### 6.5.7 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.

### 6.5.8 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.

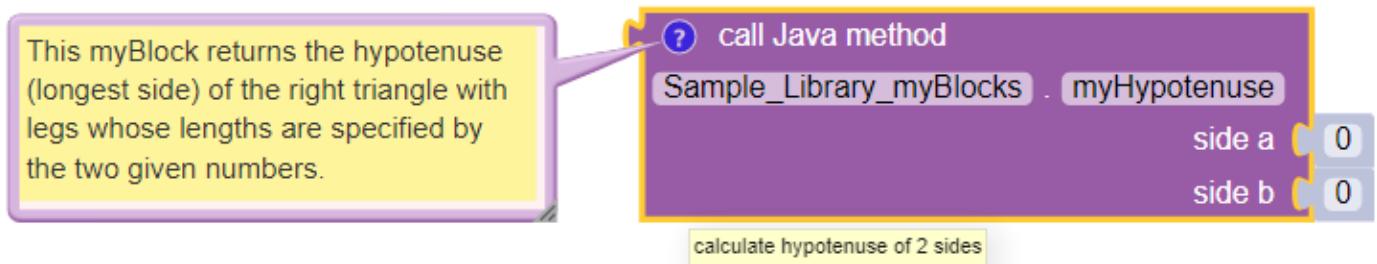


Fig. 11: myBlock using library method Geometry.hypot()

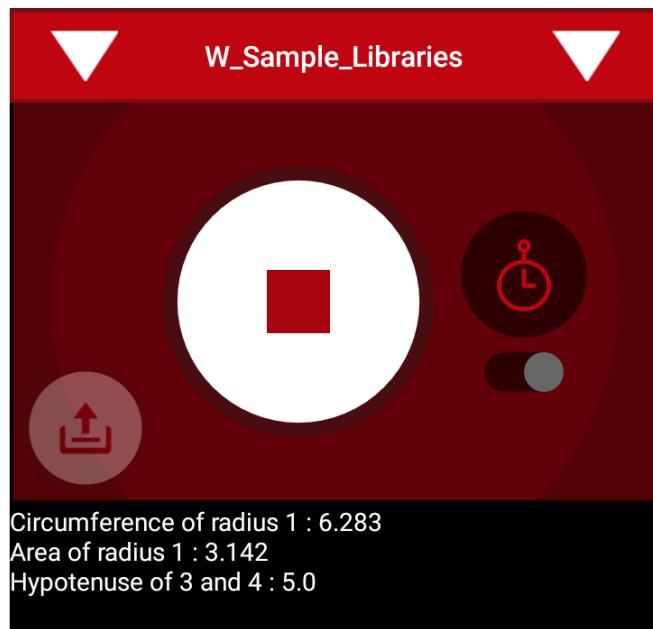


Fig. 12: Telemetry of myBlocks using Geometry library

## 6.5.9 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:

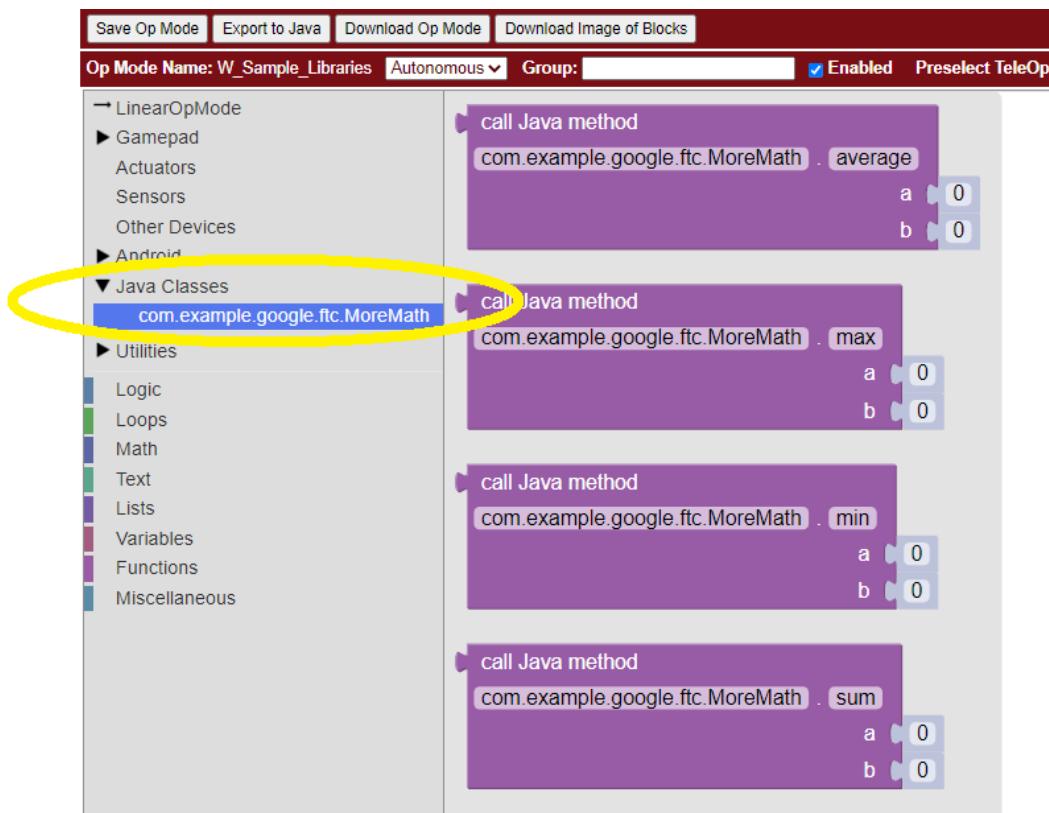


Fig. 13: Pass-through methods from class MoreMath in annotated library Arithmetic For Blocks

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`.

## 6.5.10 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.

## 6.5.11 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
```

## 6.5.12 Summary

FTC Docs Blocks and OnBot Java programmers can benefit and learn from this new capability with external libraries [FTC Programming Resources, 553](#)

You are encouraged to submit other examples and suggestions that worked for you.

Questions, comments and corrections to [westsiderobotics@verizon.net](mailto:westsiderobotics@verizon.net)

## 6.6 Universal IMU Interface

### 6.6.1 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)

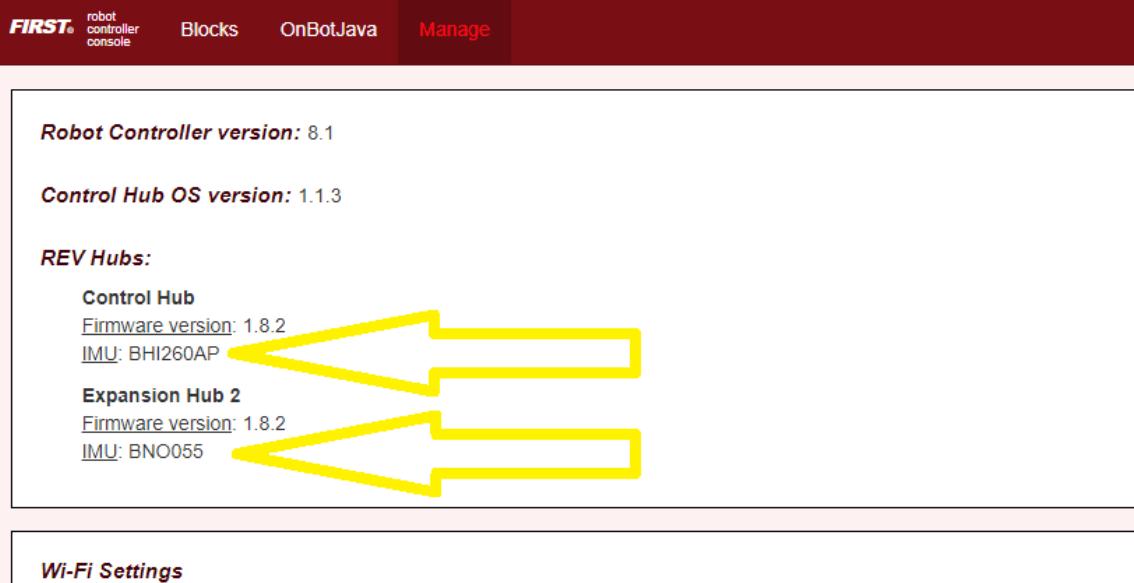
Each Hub's IMU type is listed there, as of SDK 8.0.

**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,



#### Wi-Fi Settings

Fig. 14: Sample Control Hub and Expansion Hub display

which will result in much more reliable orientation angle values for most mounting orientations.

...  
If you have calibrated your BNO055, you can provide that calibration data to the new IMU interface by passing a BN0055IMUNew.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 BN0055IMU interface. When you are migrating your code, pay careful attention to the documentation.

### 6.6.2 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 (!)

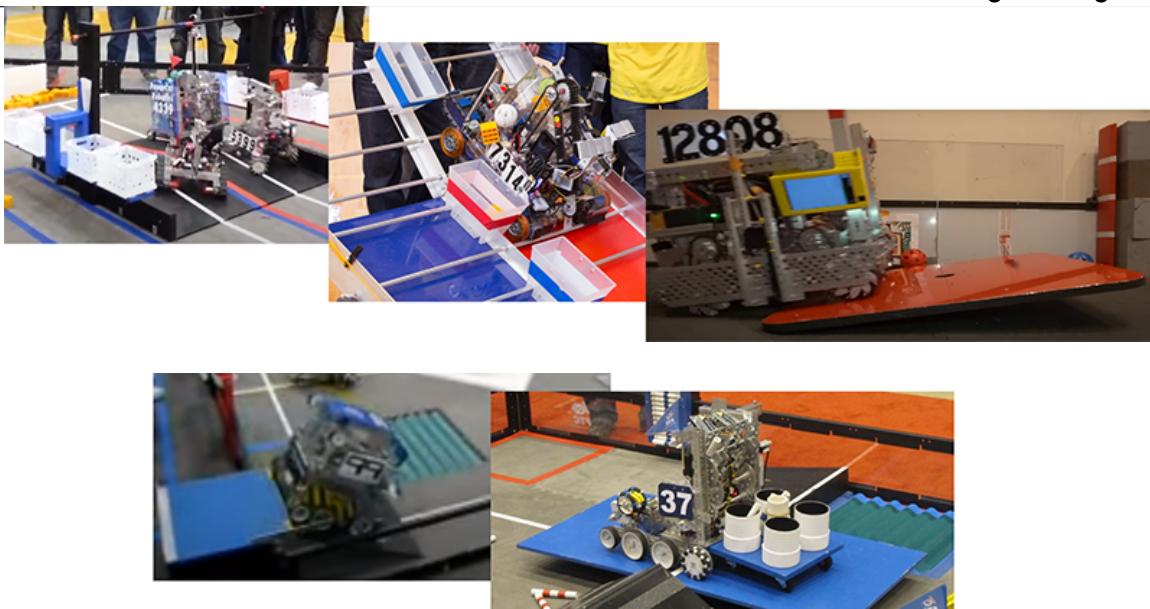


Fig. 15: Sample images from previous games utilizing tilted surfaces  
(Block Party!, FIRST RES-Q, Relic Recovery, Face Off!, Get Over It!)

- robot's secondary Expansion Hub (with IMU) is mounted on a tilting mechanism

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!

### 6.6.3 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.*

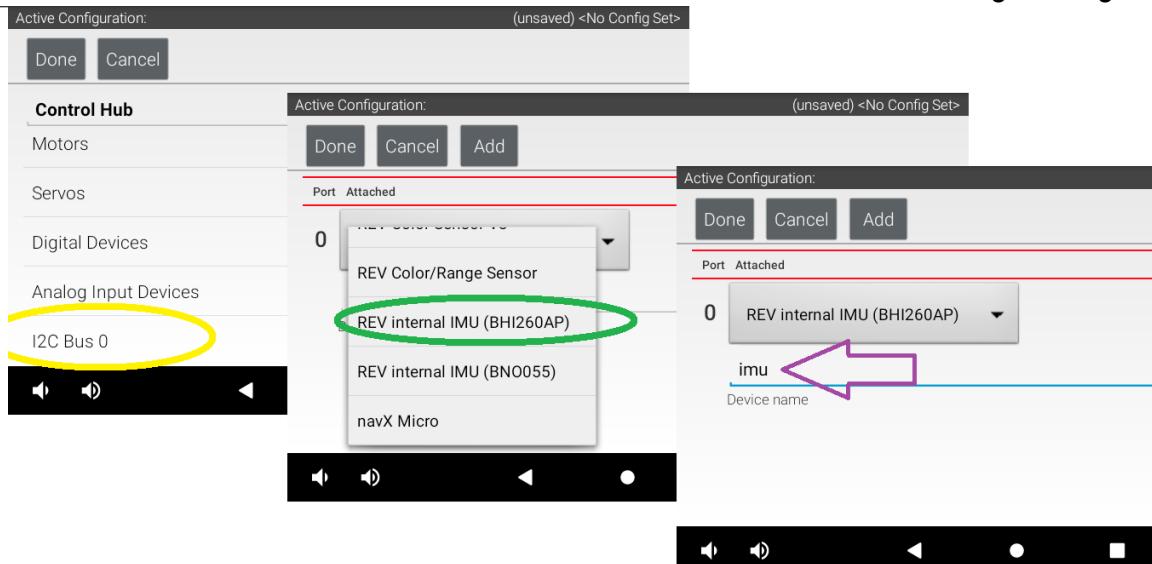


Fig. 16: REV IMU Robot Configuration Validation

#### 6.6.4 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 BN0055IMU 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.

## 6.6.5 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**.

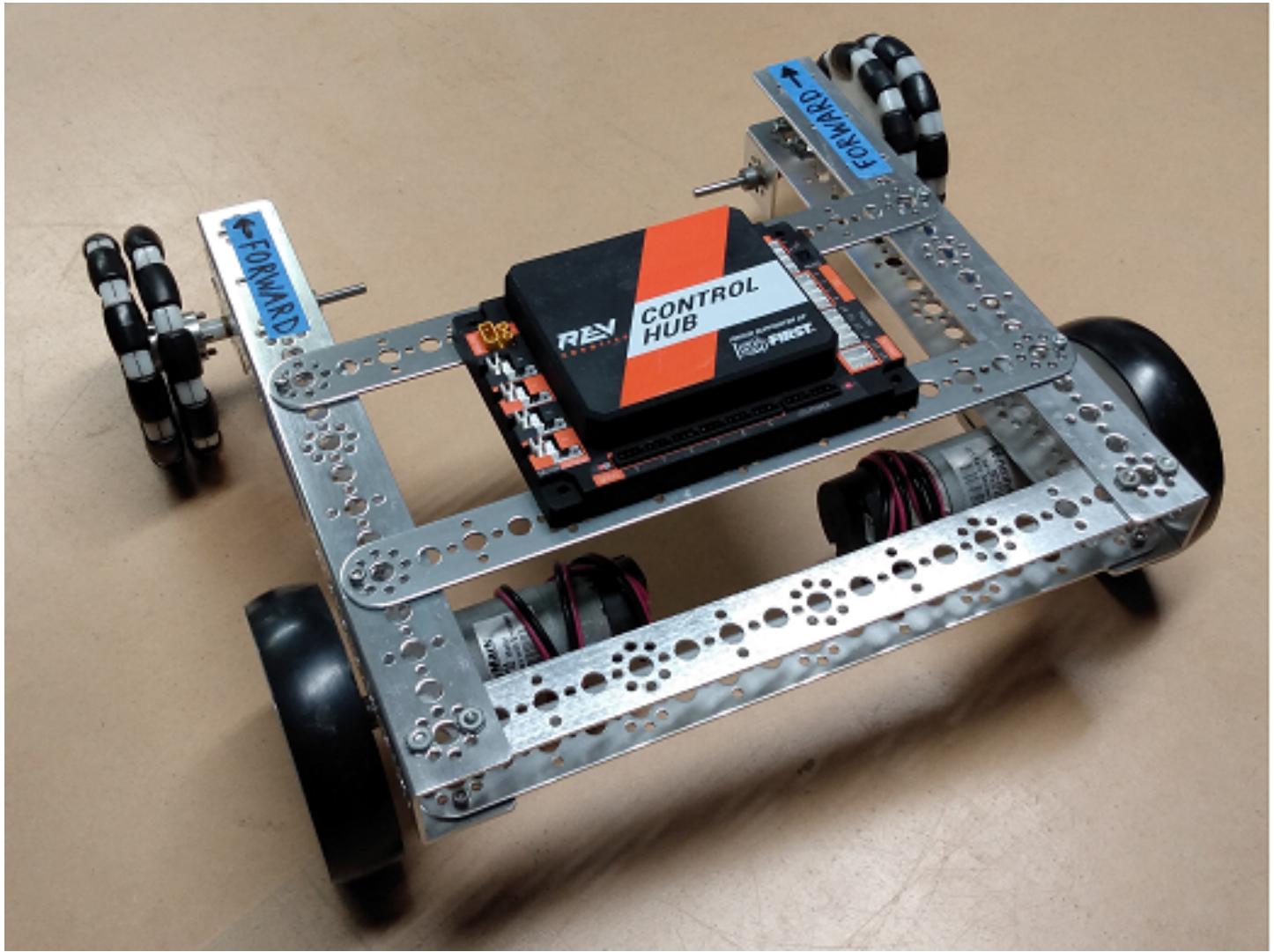
### Orthogonal Mounting

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.

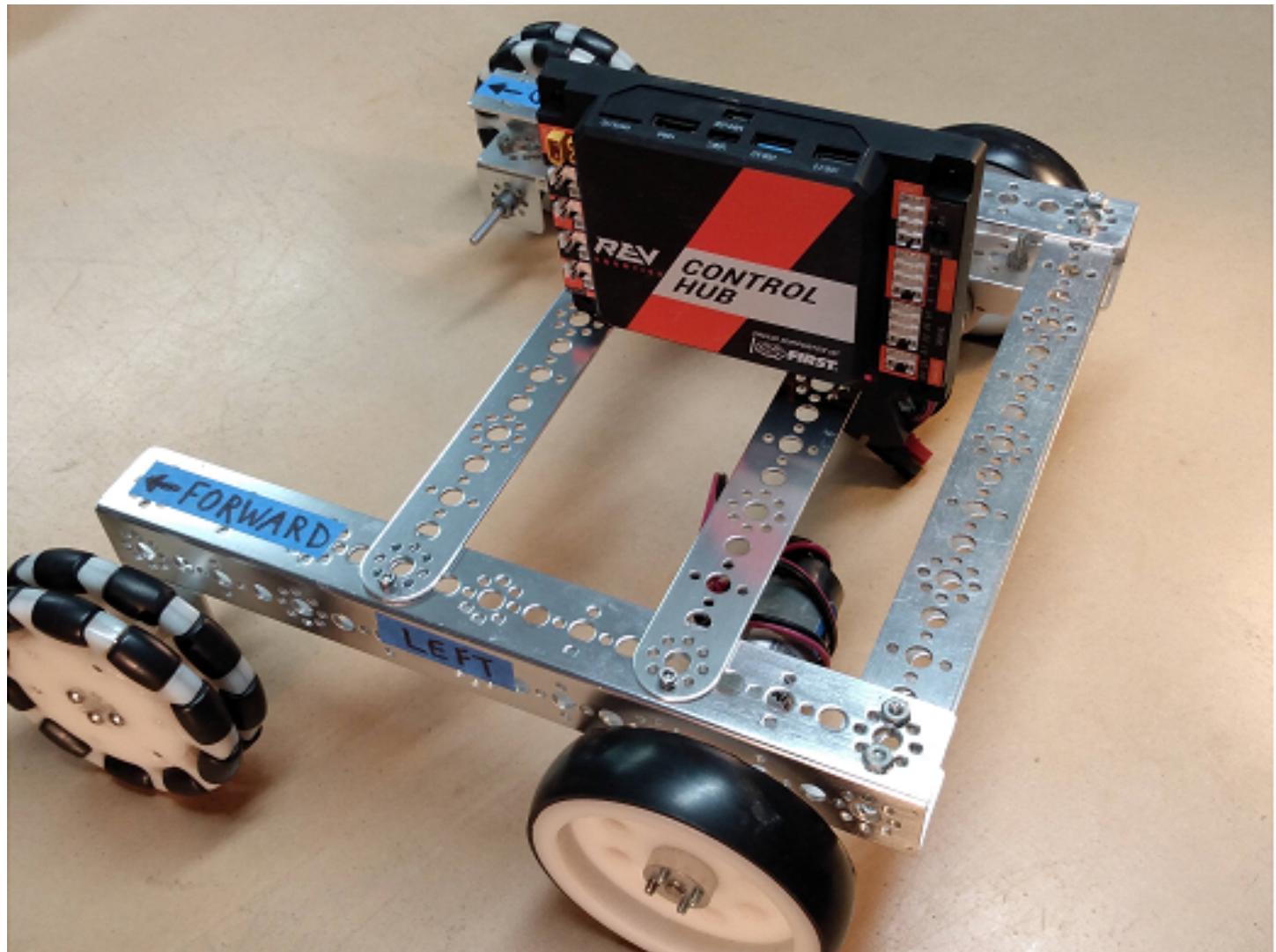
Here are some common examples:

Orthogonal #1

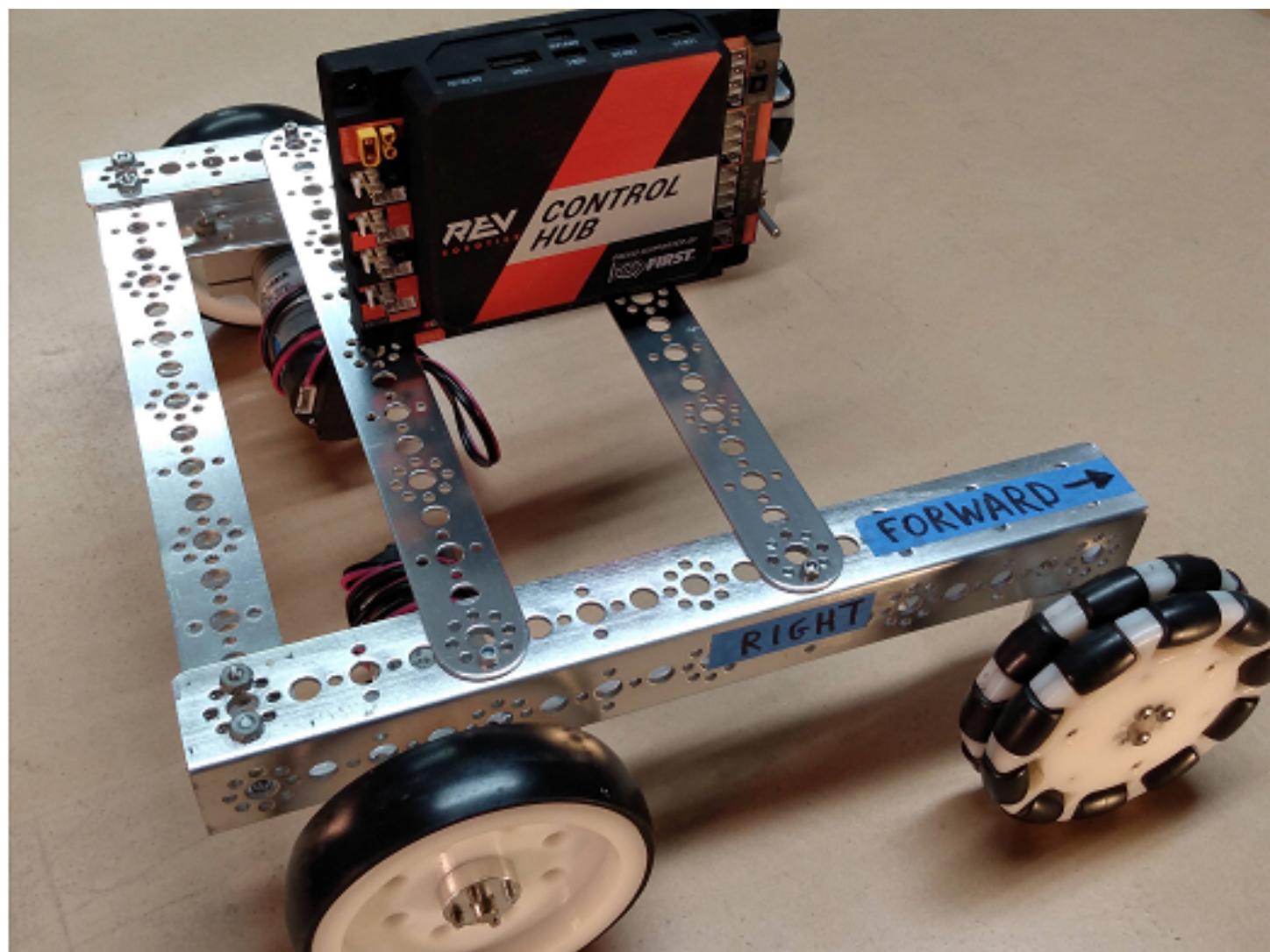


Logo UP, USB FORWARD

Orthogonal #2



Orthogonal #3



Logo RIGHT, USB UP

Orthogonal #4

Logo FORWARD, USB UP

Orthogonal #5

Logo BACKWARD, USB UP

Orthogonal #6

Logo DOWN, USB FORWARD

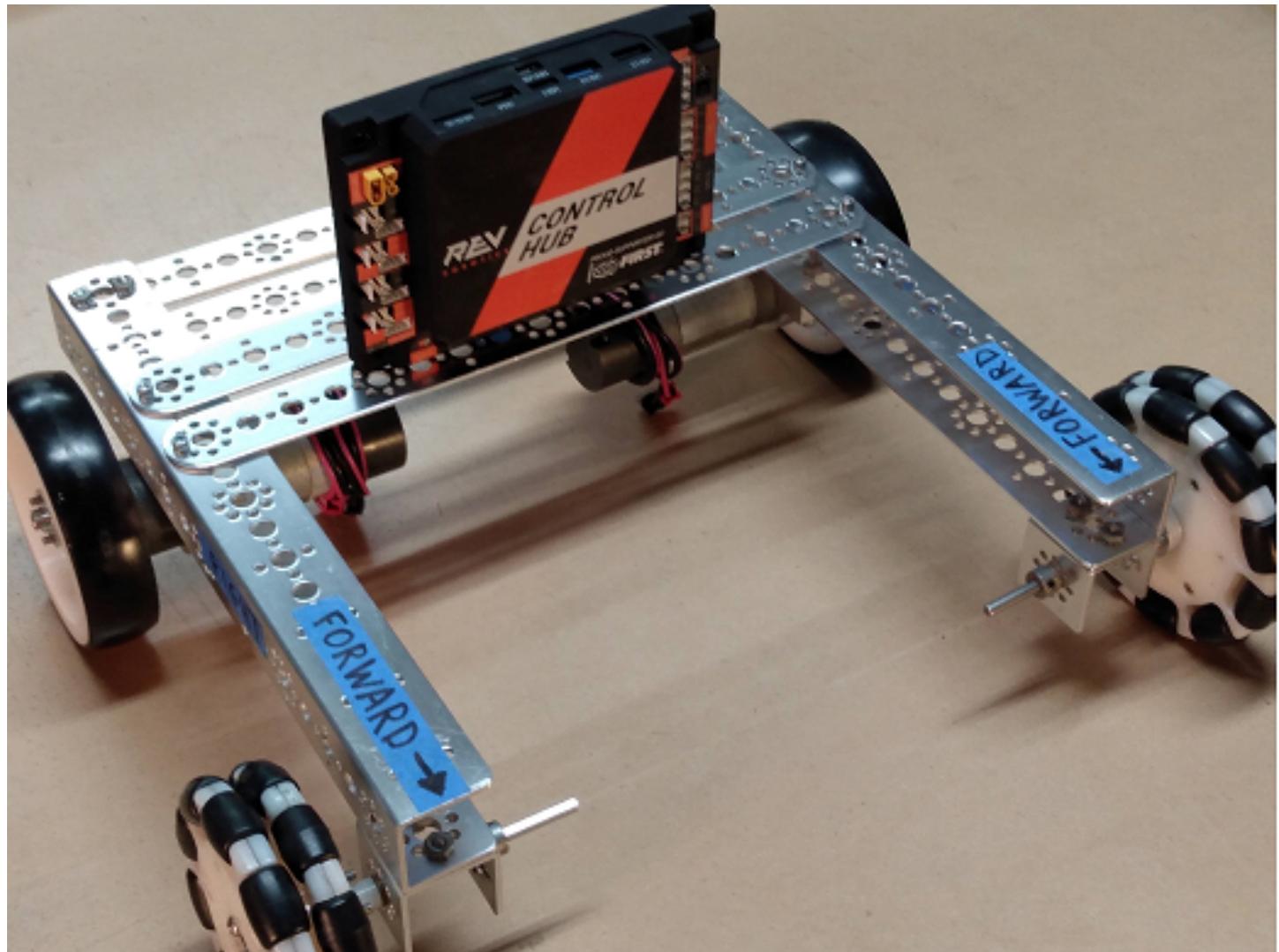
Orthogonal #7

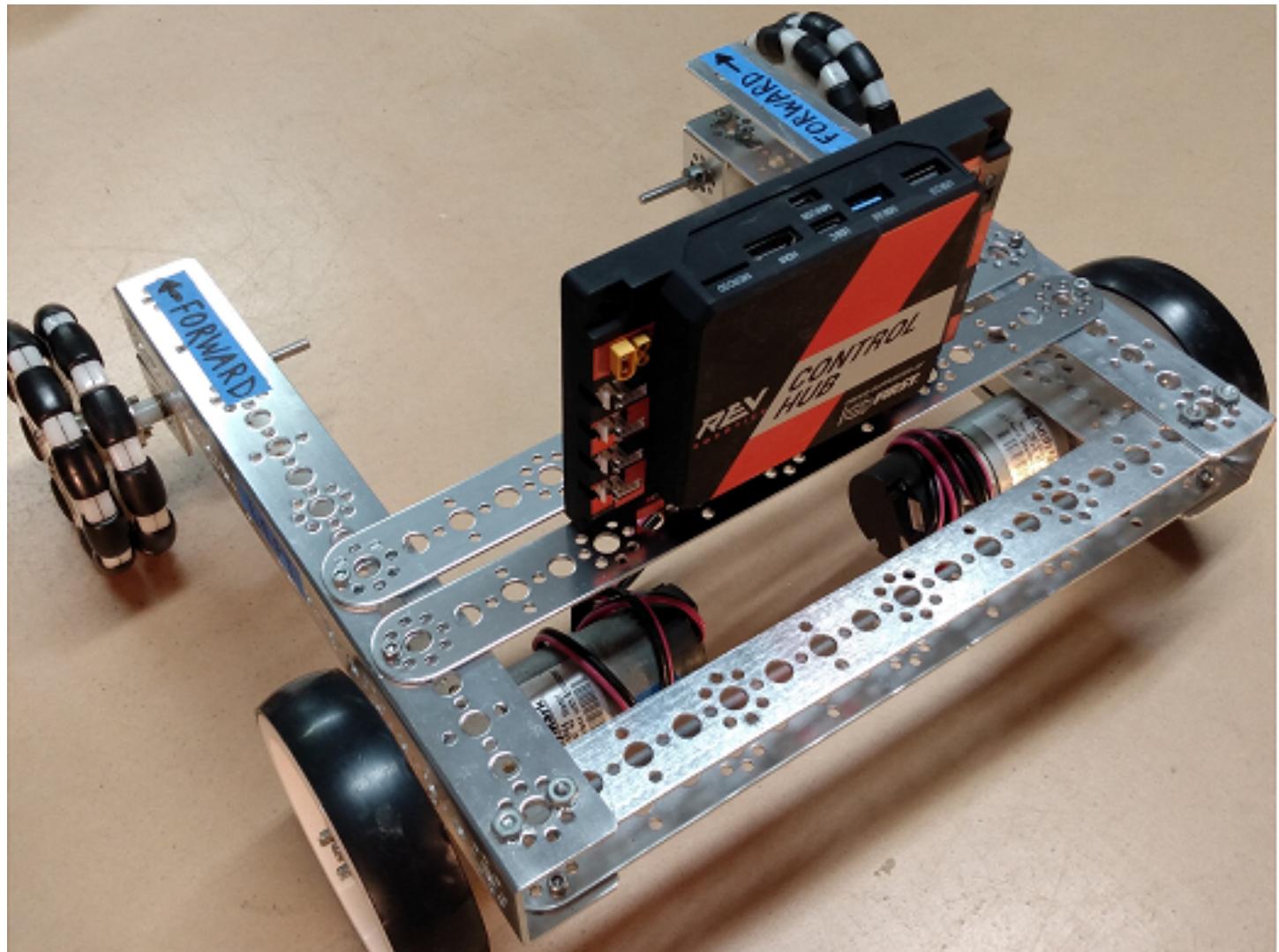
Logo FORWARD, USB LEFT

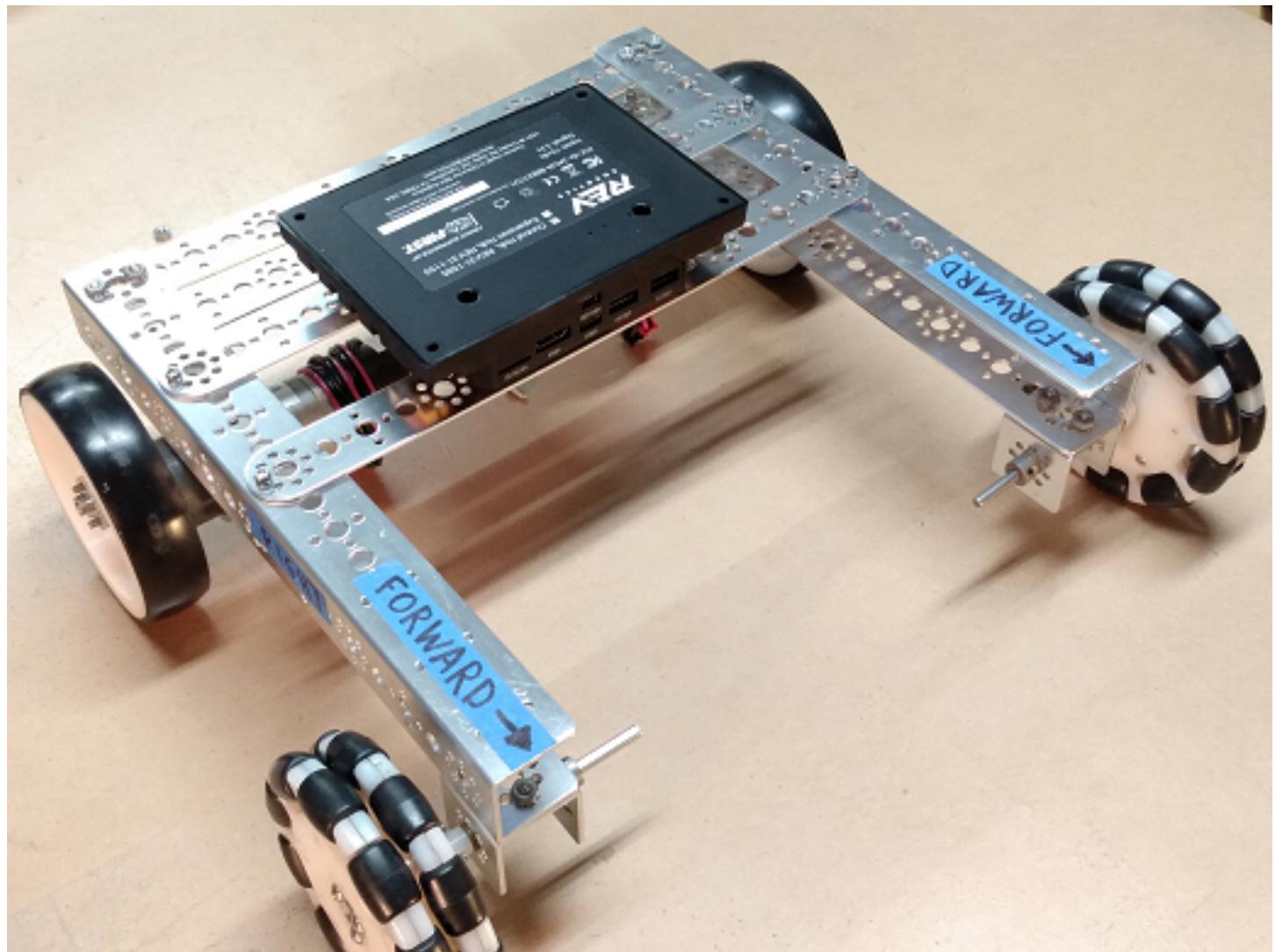
Orthogonal #8

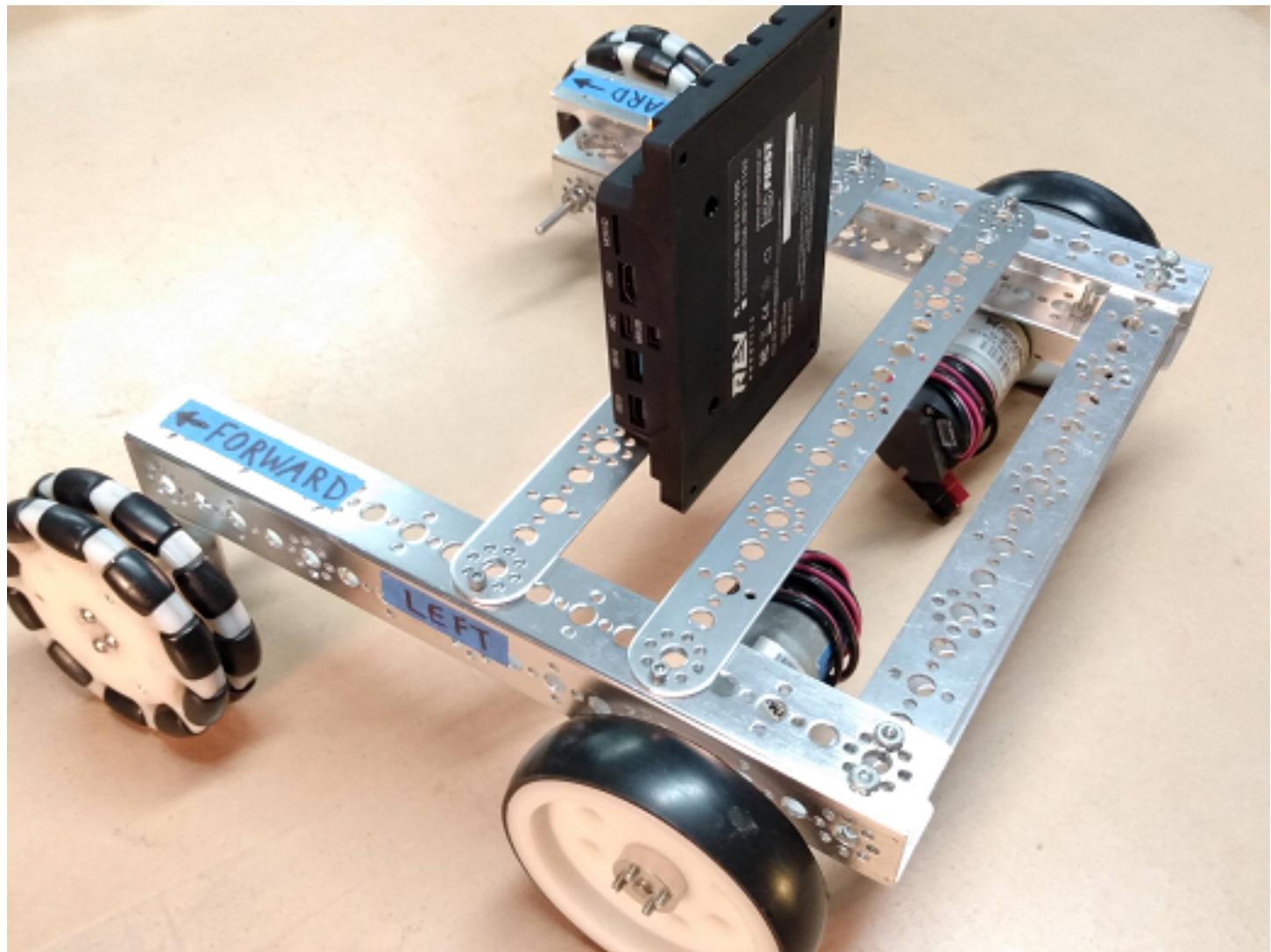
Logo FORWARD, USB RIGHT

Orthogonal #9

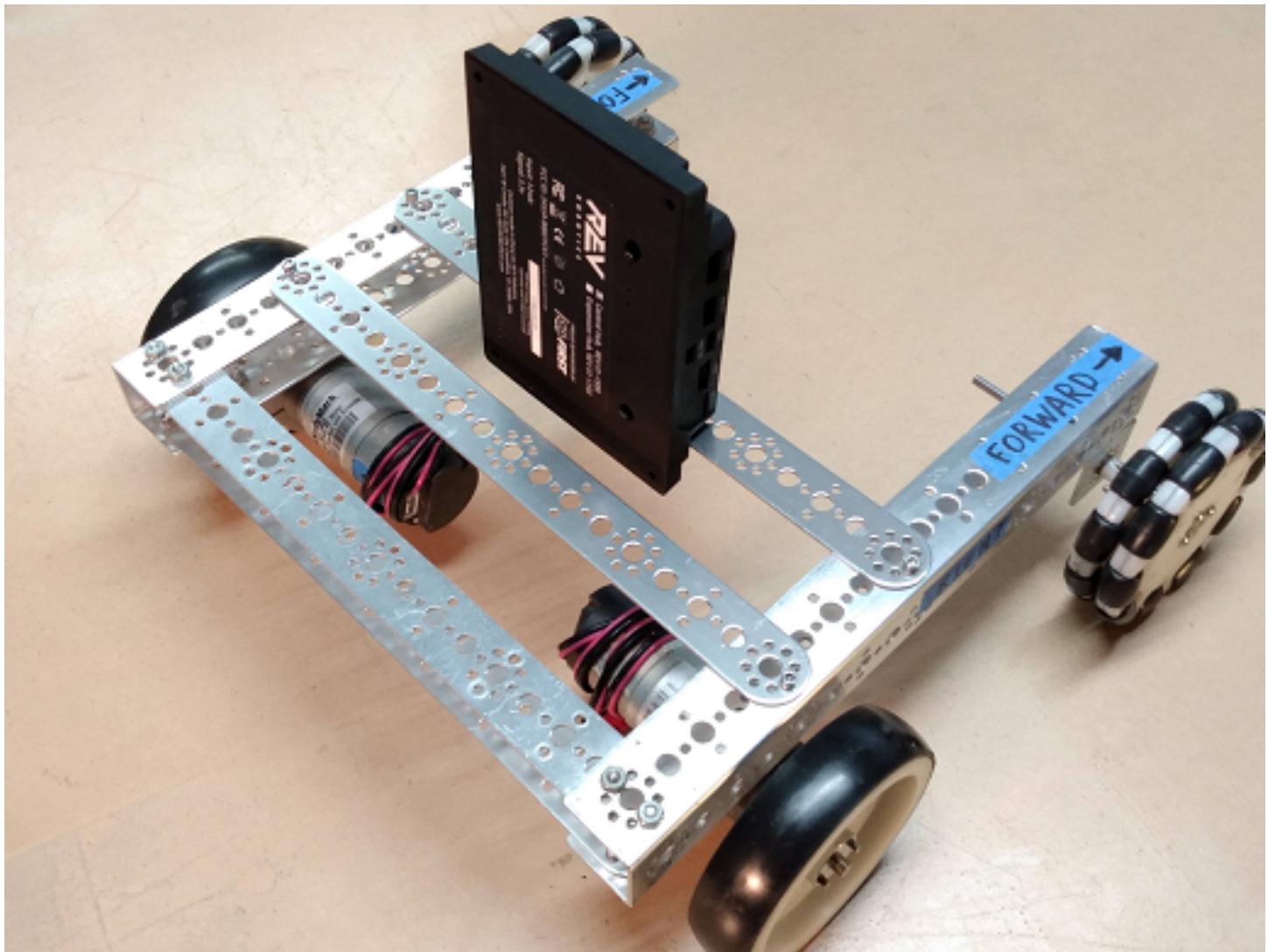


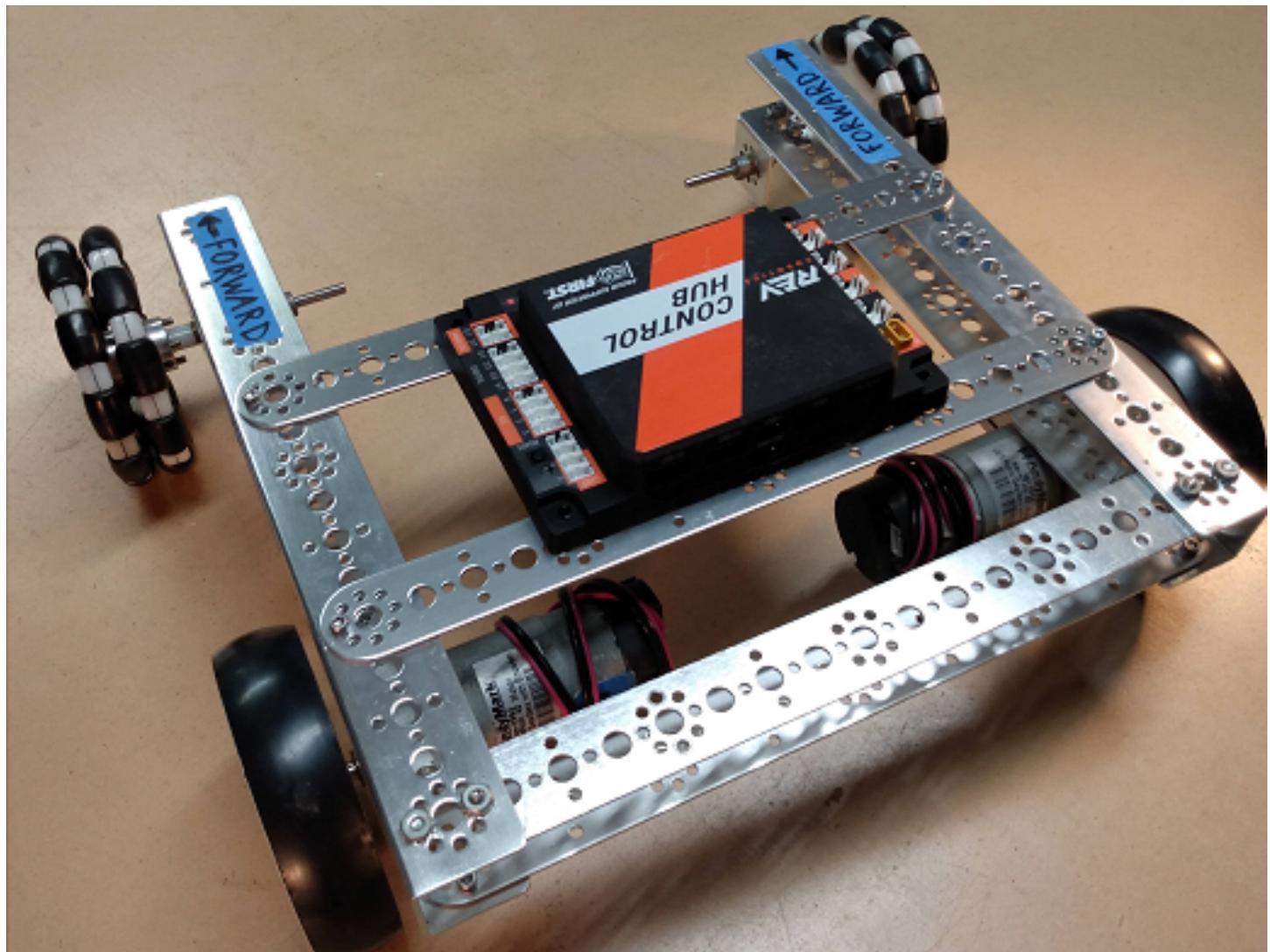






**Gracious Professionalism® - "Doing your best work while treating others with respect and kindness - It's what makes FIRST, first."**





**Gracious Professionalism® - "Doing your best work while treating others with respect and kindness - It's what makes FIRST, first."**

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.

### 6.6.6 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).

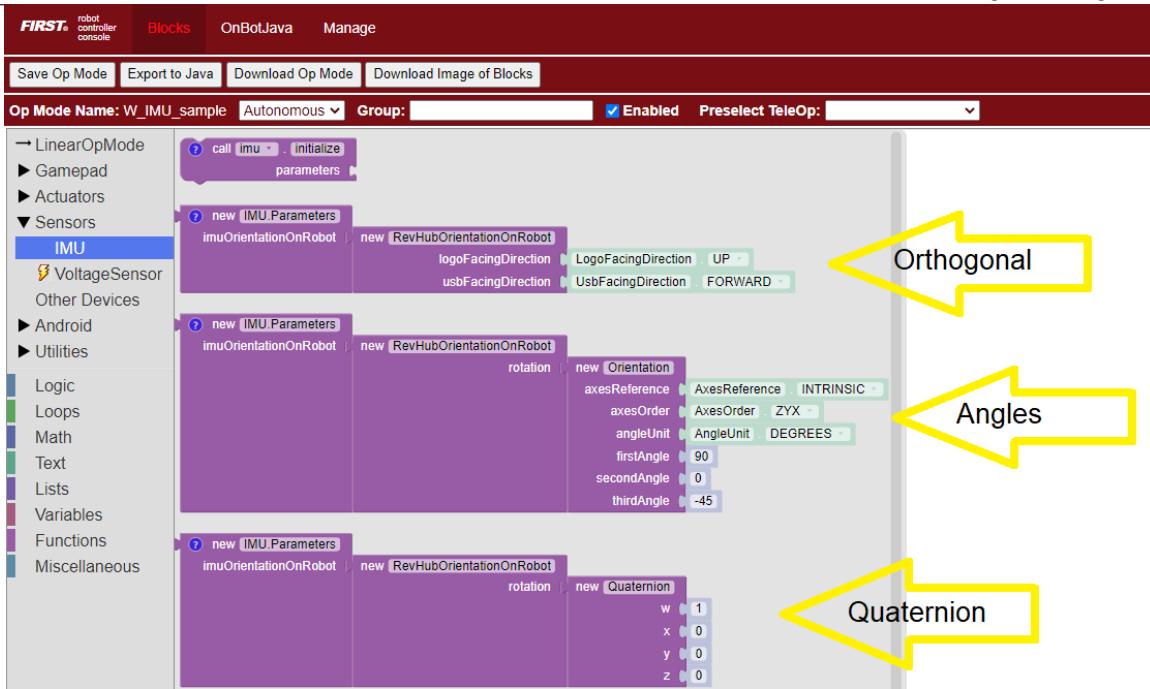


Fig. 17: Sample Blocks screenshot, demonstrating the three parameter methods

## Blocks

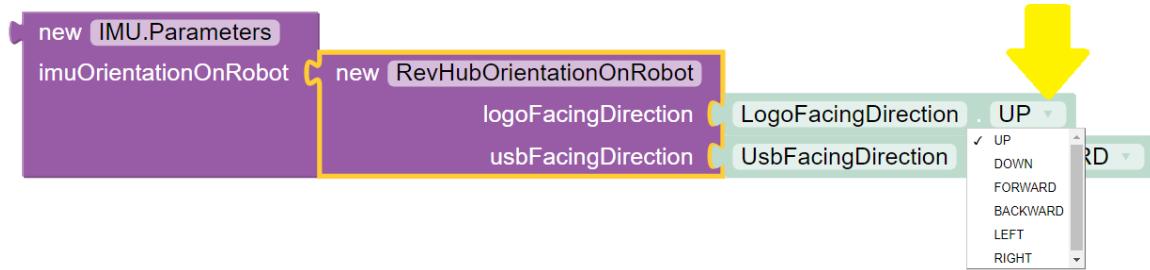


Fig. 18: Specifying Logo Facing Direction

## Java

```
IMU.Parameters myIMUparameters;

myIMUparameters = new IMU.Parameters(
    new RevHubOrientationOnRobot(
        RevHubOrientationOnRobot.LogoFacingDirection.UP,
        RevHubOrientationOnRobot.UsbFacingDirection.FORWARD
    )
);
```

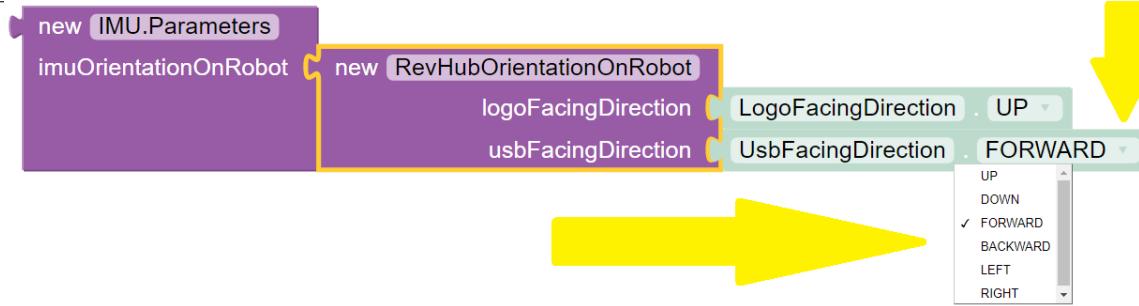


Fig. 19: Specifying USB Facing Direction



Fig. 20: 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 BN0055IMU 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 BN0055 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.

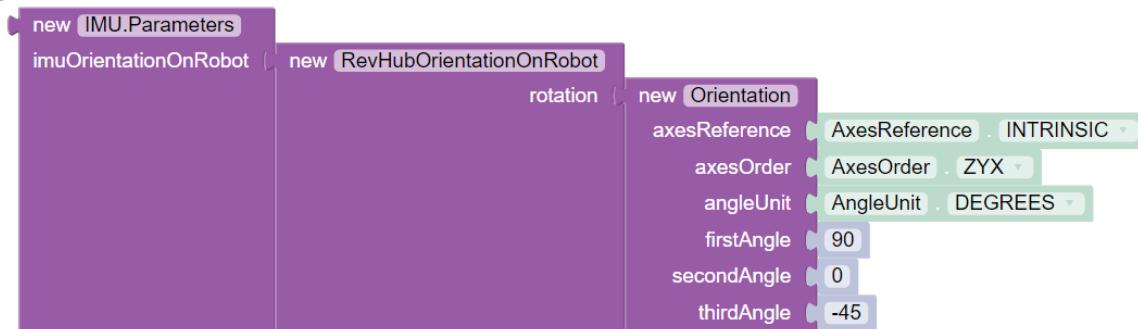


Fig. 21: Sample Block demonstrating angles method

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).

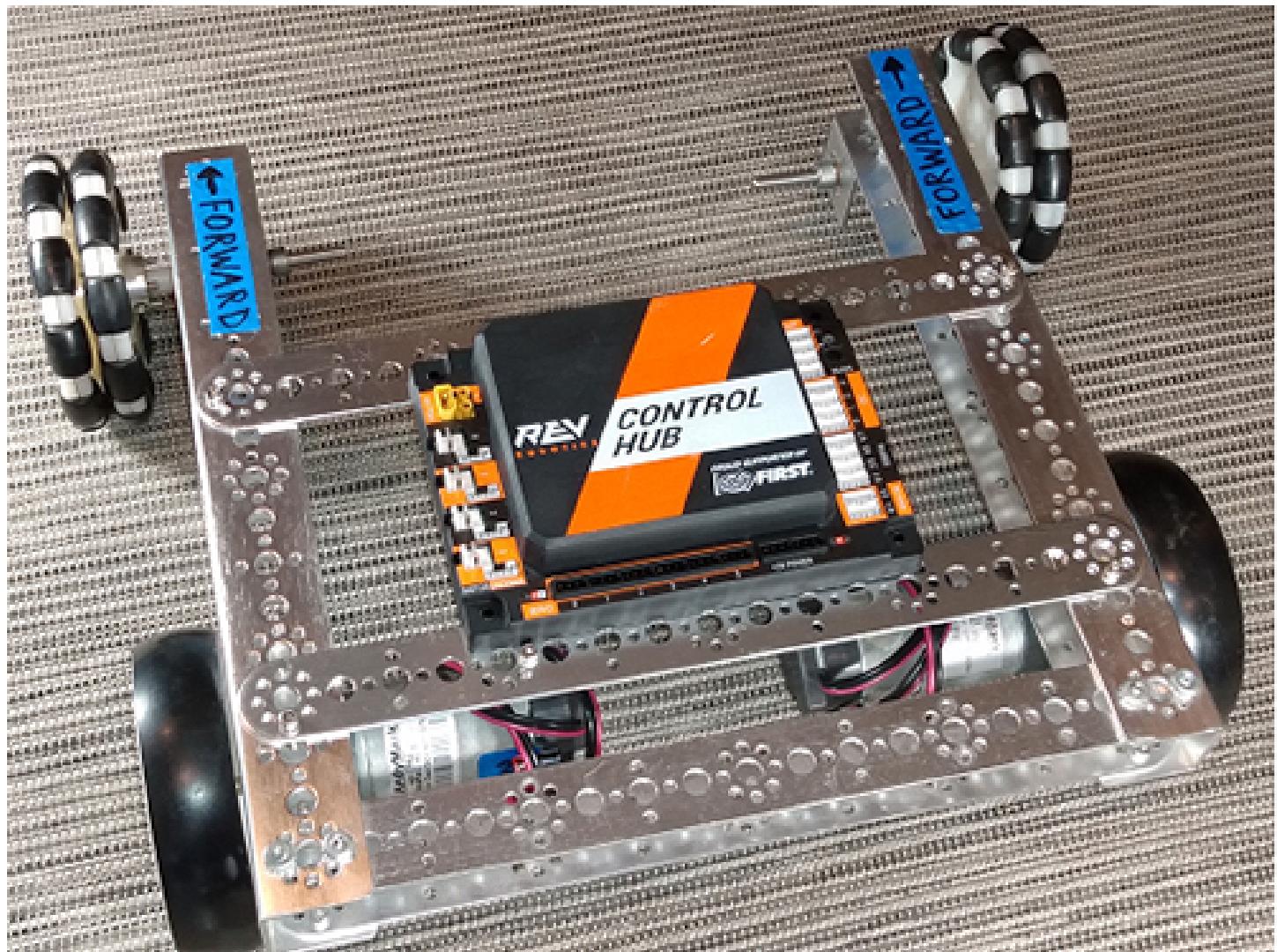
## Blocks

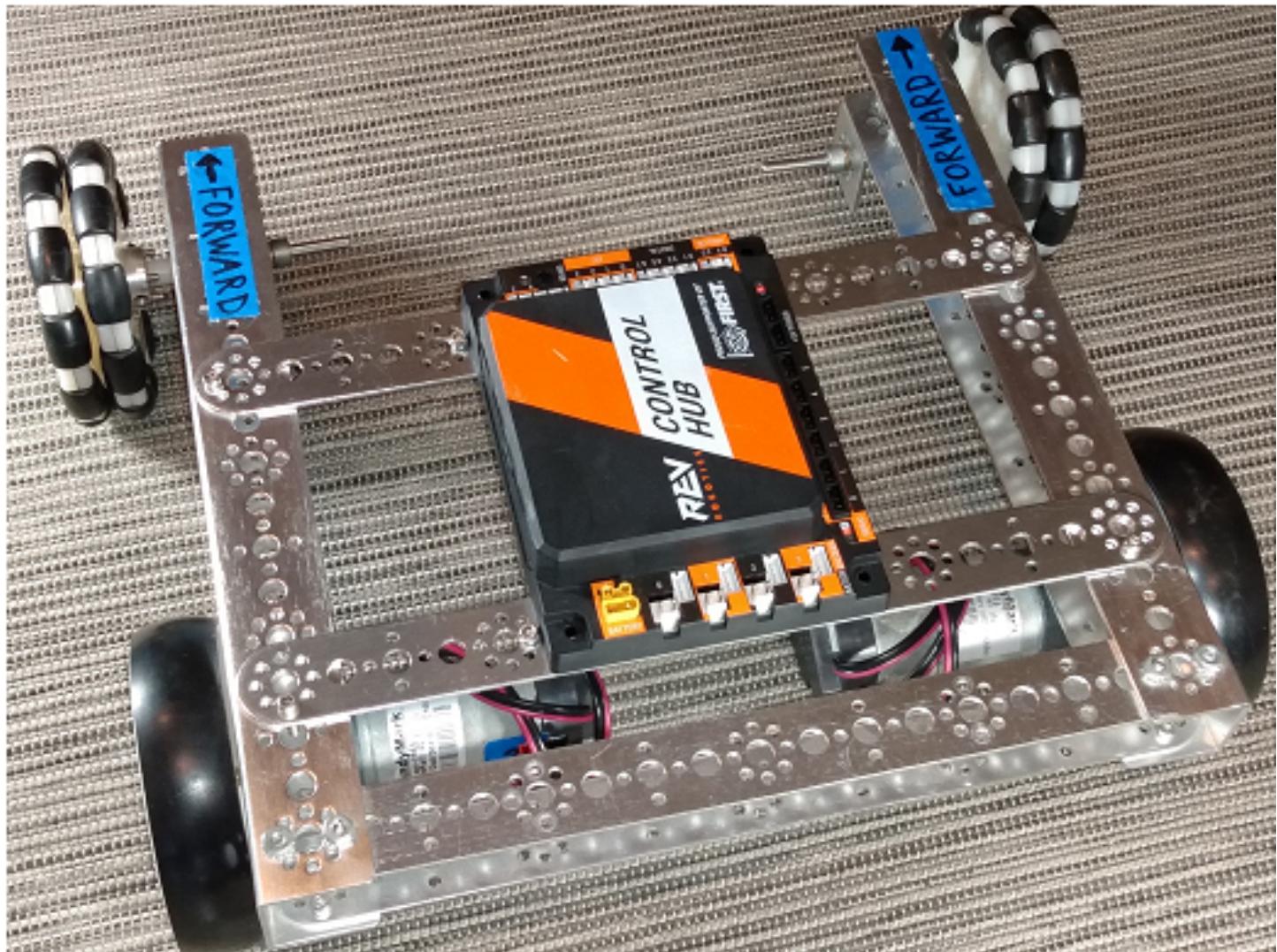
### Java

```
IMU.Parameters myIMUparameters;

myIMUparameters = new IMU.Parameters(
    new RevHubOrientationOnRobot(
        new Orientation(
            AxesReference.INTRINSIC,
            AxesOrder.ZYX,
            AngleUnit.DEGREES,
            90,
            0,
            -45,
```

(continues on next page)





**Gracious Professionalism® - "Doing your best work while treating others with respect and kindness - It's what makes FIRST, first."**

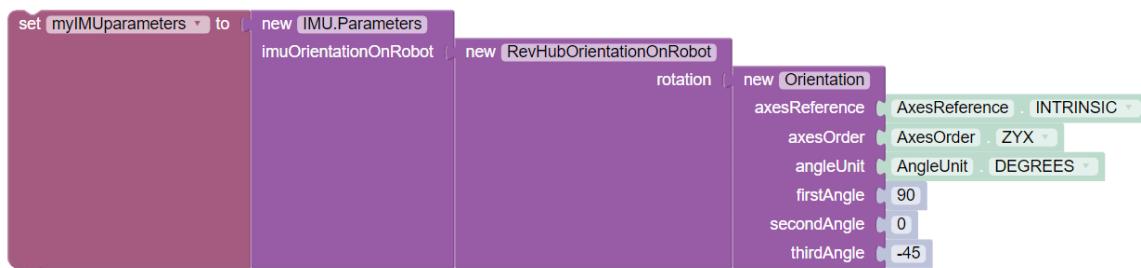
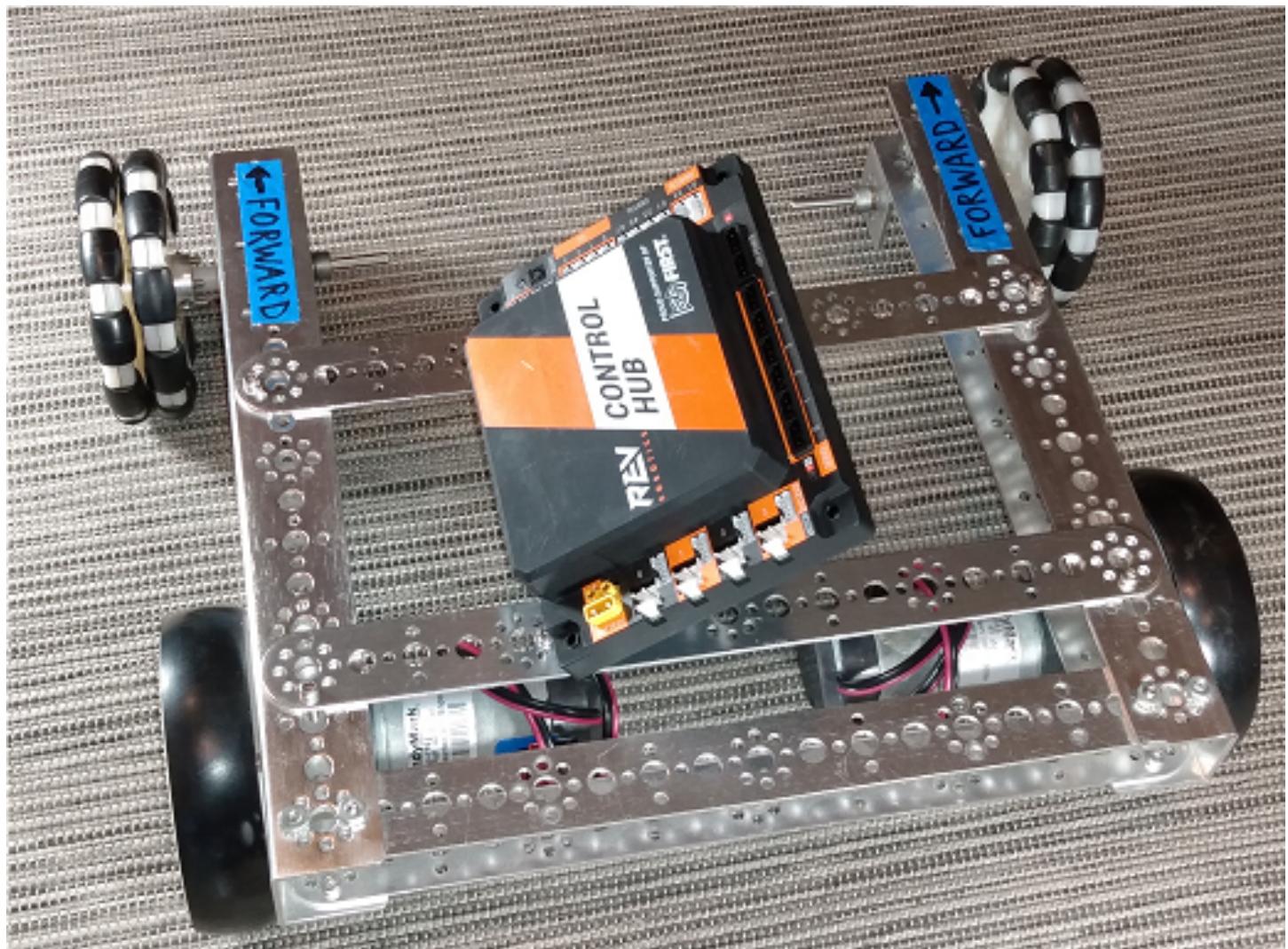


Fig. 22: Setting Angles in Blocks

```

        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

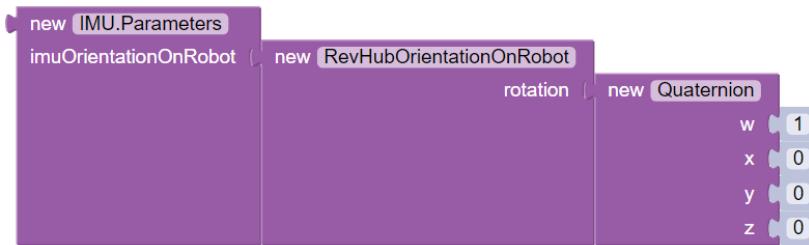


Fig. 23: Default Quaternion (no rotation)

### 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)

```

        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 BN0055 IMU interface, initialization 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. 24: Initializing the IMU directly

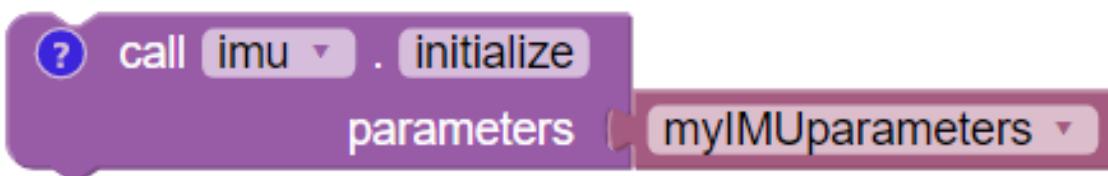


Fig. 25: Initializing the IMU using Parameters

```
// 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. 26: 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**.

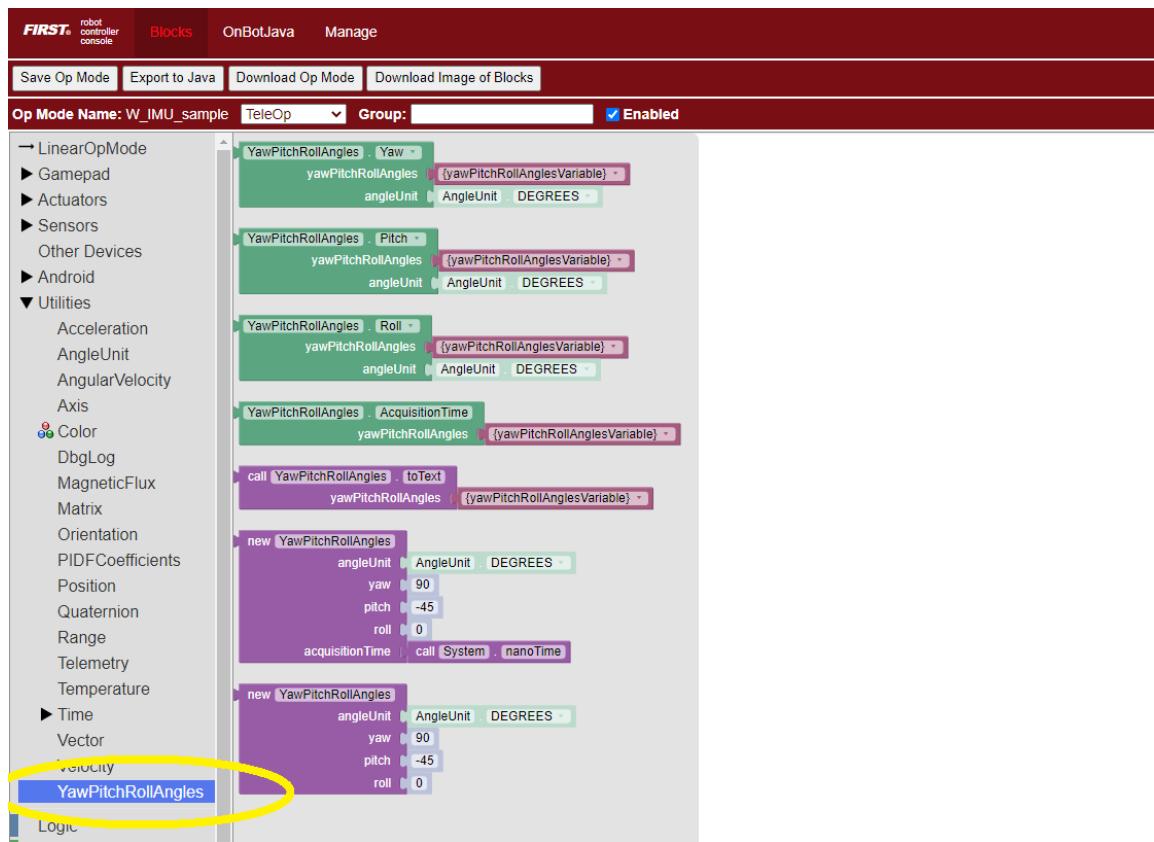


Fig. 27: Extract Angles

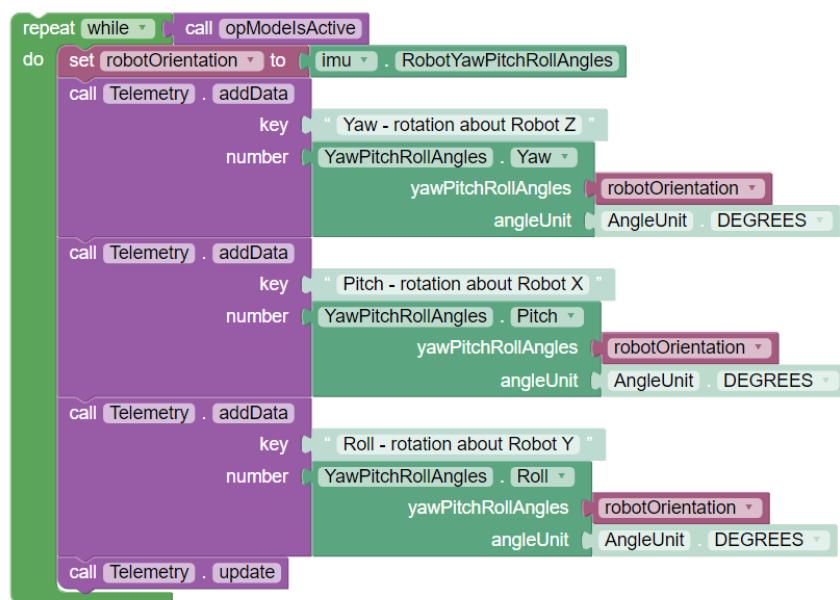


Fig. 28: Displaying Yaw-Pitch-Roll using Telemetry

```
// 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):

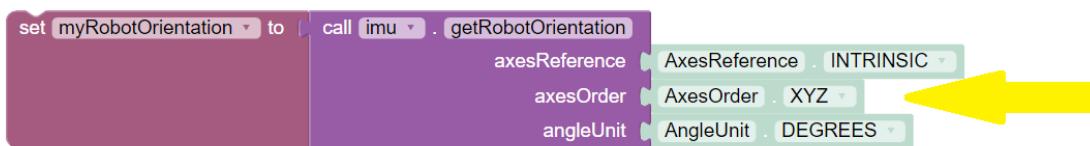


Fig. 29: Get Robot Orientation

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:

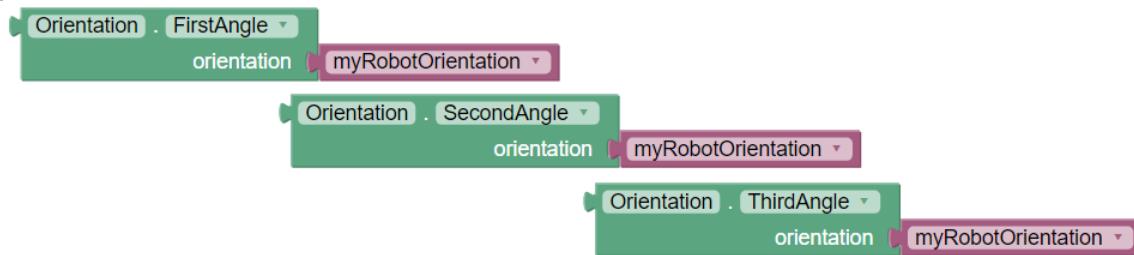


Fig. 30: Extract Orientation Angles

## Java

```
// Create Orientation variable
Orientation myRobotOrientation;

// Get Robot Orientation
myRobotOrientation = imu.getRobotOrientation(
    AxesReferenceINTRINSIC,
    AxesOrderXYZ,
    AngleUnitDEGREES
);

// 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):

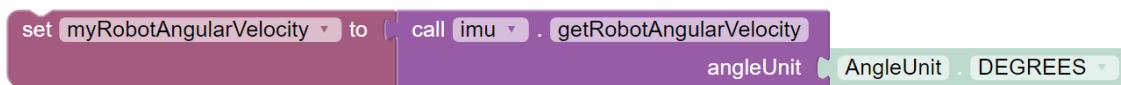


Fig. 31: Get Robot Angular Velocity

Then extract the specific axis rotations you want, from the Blocks Utilities / AngularVelocity palette:  
These Blocks are shown in the Sample OpMode called SensorIMU.

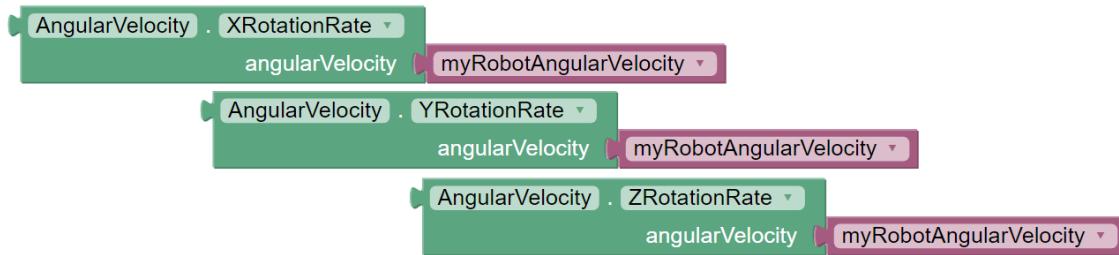


Fig. 32: Extract Rotation Rates

**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. 33: Reset Yaw

```
// 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.

---

## 6.6.7 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.

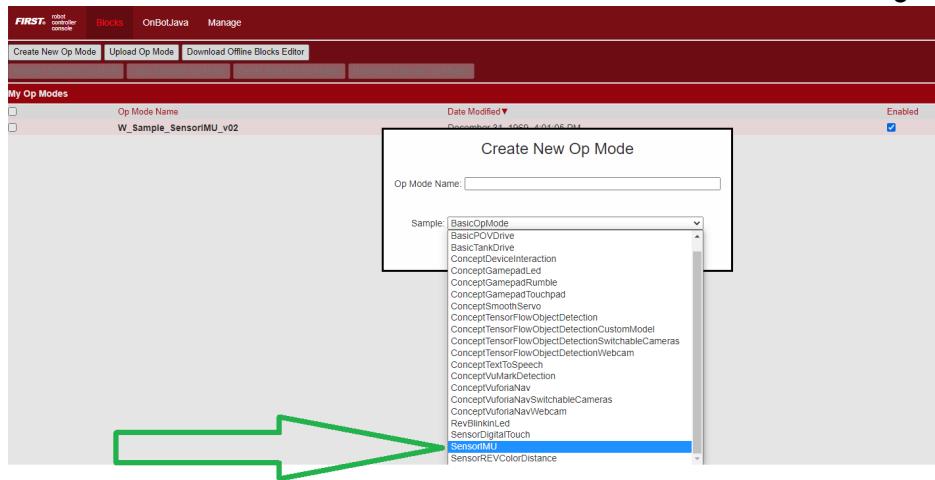


Fig. 34: Blocks IMU Sample

## 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)

```

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)

```

        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 BN0055 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)

```

* <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)

```

* 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.xzOrientation;

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 BN0055 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)

```

* <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_nonorthe 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)

```

/*
* -----
* 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)

```

        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.

## 6.6.8 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.

## 6.6.9 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](mailto:westsiderobotics@verizon.net)

## 6.7 Using the Kotlin Programming Language

### 6.7.1 What Is Kotlin?

The Kotlin programming language is a modern alternative to the Java programming 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.
- Dynamic typing; Kotlin allows you to use dynamic typing 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.

### 6.7.2 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.

### 6.7.3 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 = '1.8.20' <----- ADD THIS LINE, UPDATE VERSION TO LATEST IF NEEDED
    repositories {
        mavenCentral()
        google()
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:7.2.0'
```

(continues on next page)

```

    }
}

classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:$kotlin_version" <----- ADD THIS LINE
}
}

```

**Note:** This file is located in the base folder of your project, not the one in the TeamCode module nor the one in the FtcRobotController 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](#)).

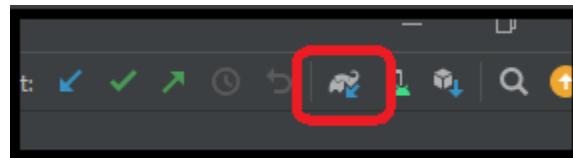
Next you need to add the Kotlin plugin to the build.gradle file in the TeamCode module. Open the file and find the following section near the top of the file. Change it to look like this:

```

// 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.

## 6.8 HuskyLens Intro for FIRST Tech Challenge

### 6.8.1 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.

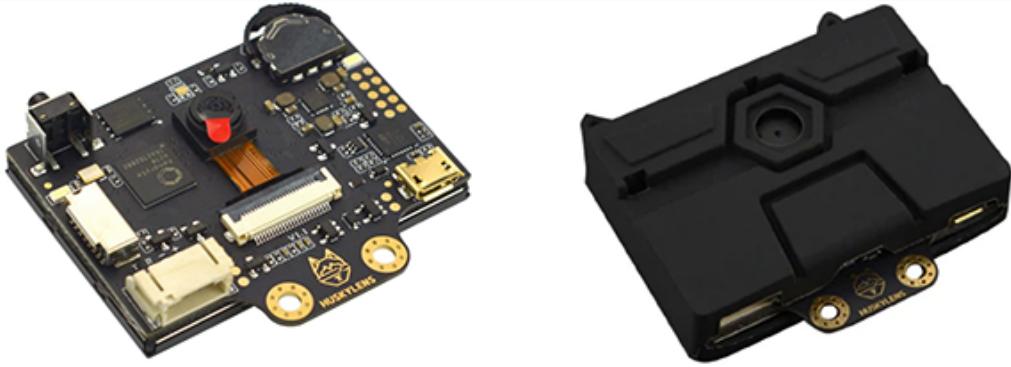


Fig. 35: DFRobot HuskyLens

HuskyLens is **not a USB webcam**, and **does not use** the FTC *VisionPortal* software.

### 6.8.2 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.

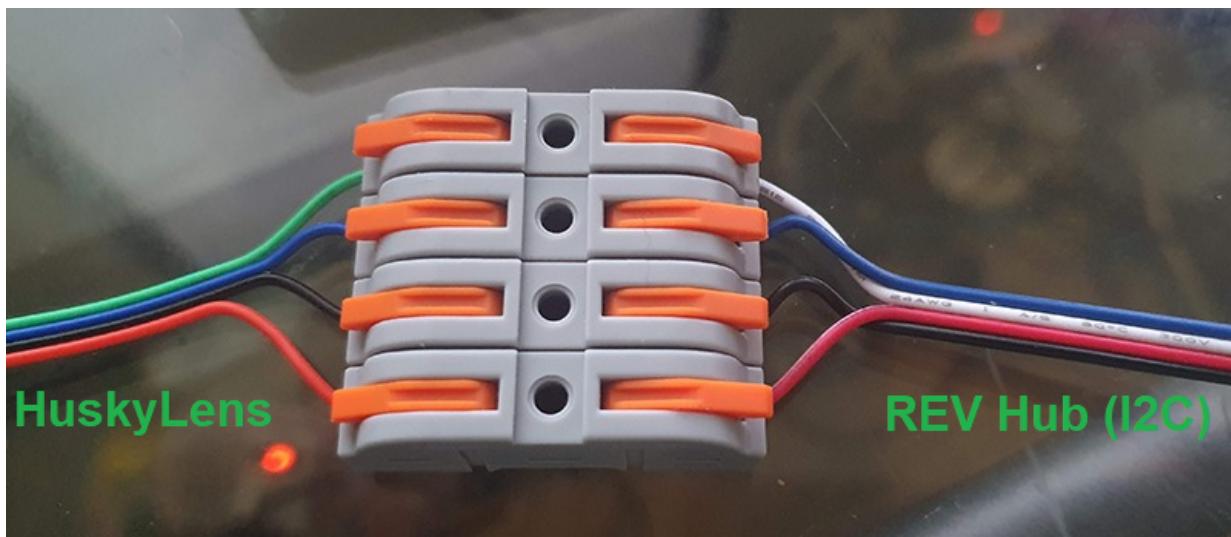


Fig. 36: image credit: @texasdiaz

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

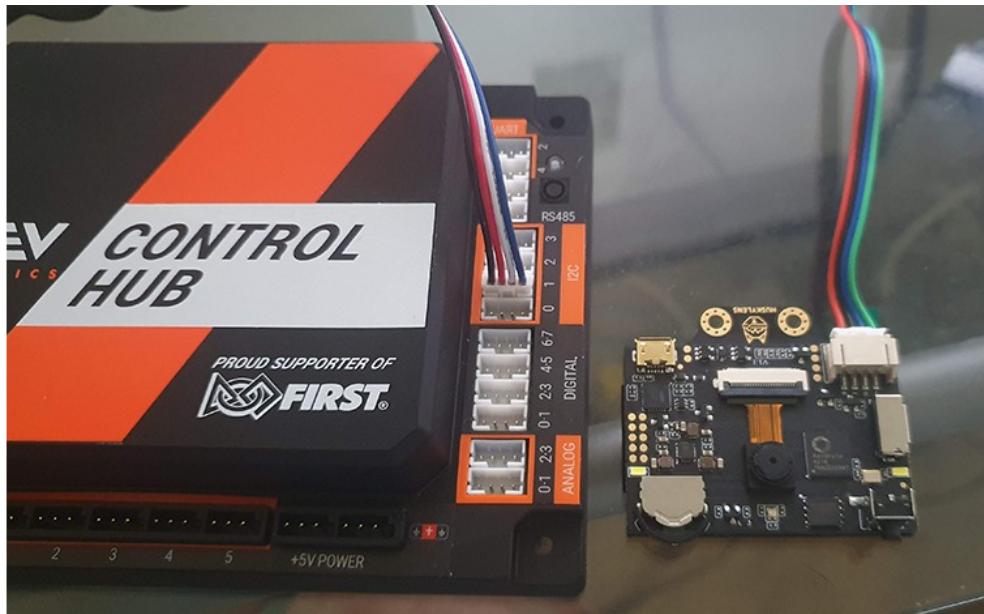


Fig. 37: image credit: @texasdiaz

### 6.8.3 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.

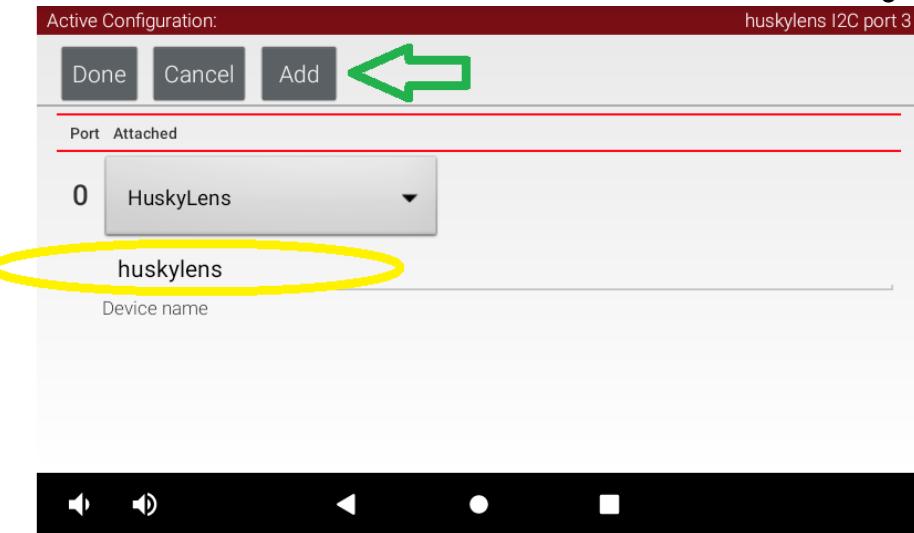


Fig. 38: Driver Station Config

#### 6.8.4 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":

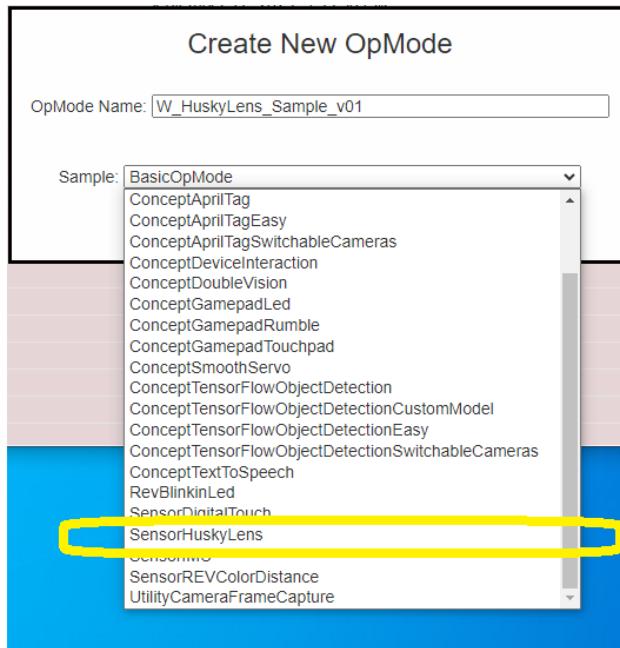


Fig. 39: HuskyLens Blocks Sample

Change the OpMode type from TeleOp to Autonomous, since this sample does not use the gamepads.

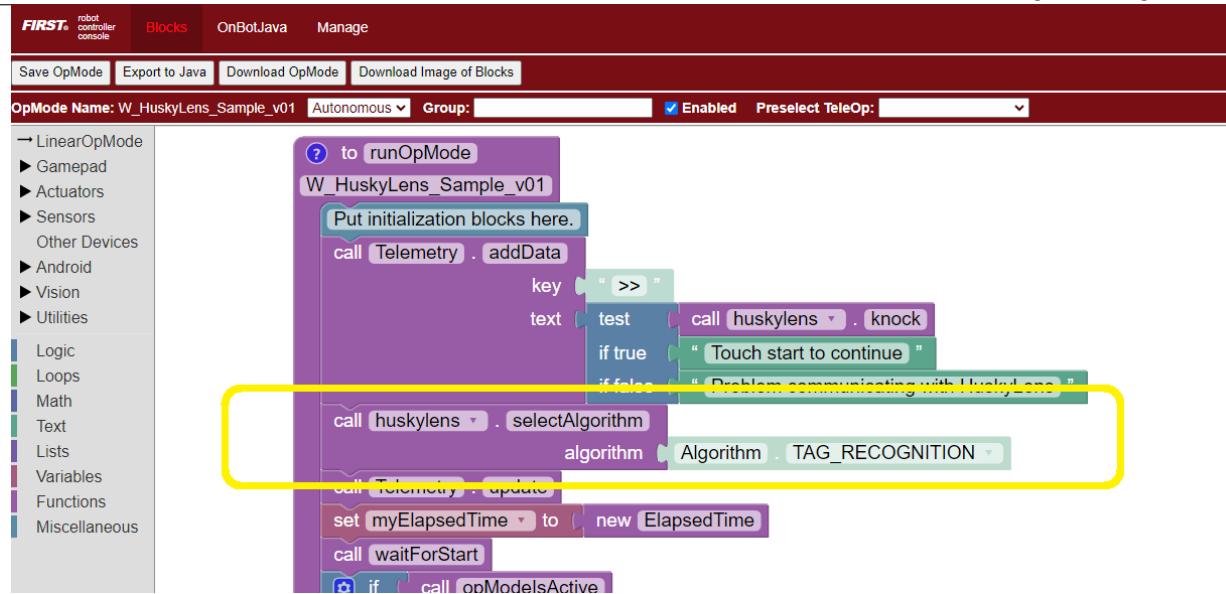


Fig. 40: HuskyLens Blocks Algorithm

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:

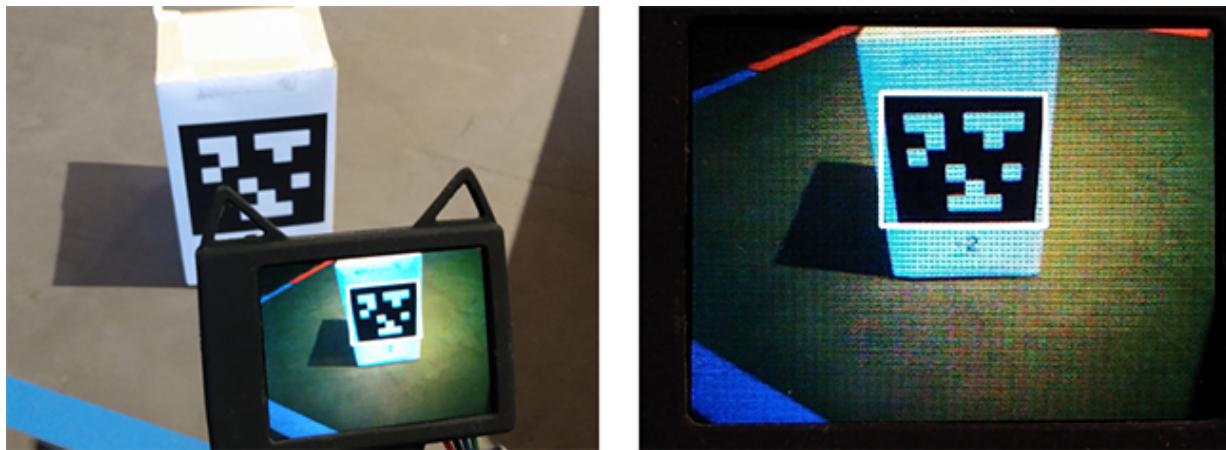


Fig. 41: Uncategorized AprilTag Detected

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

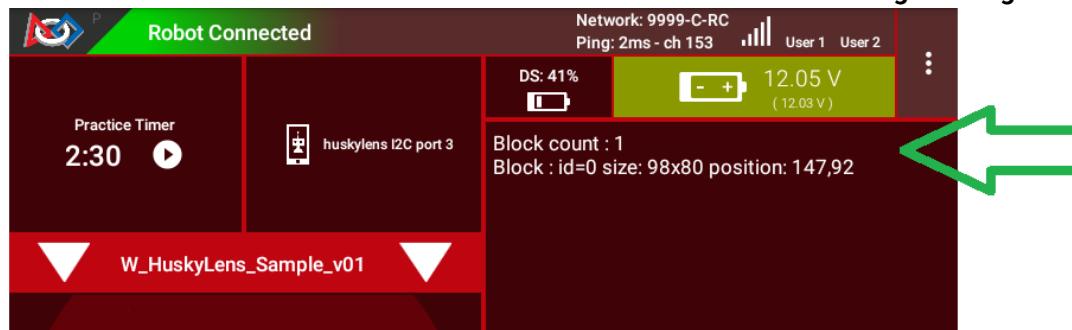


Fig. 42: AprilTag Telemetry

- 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.

### 6.8.5 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.

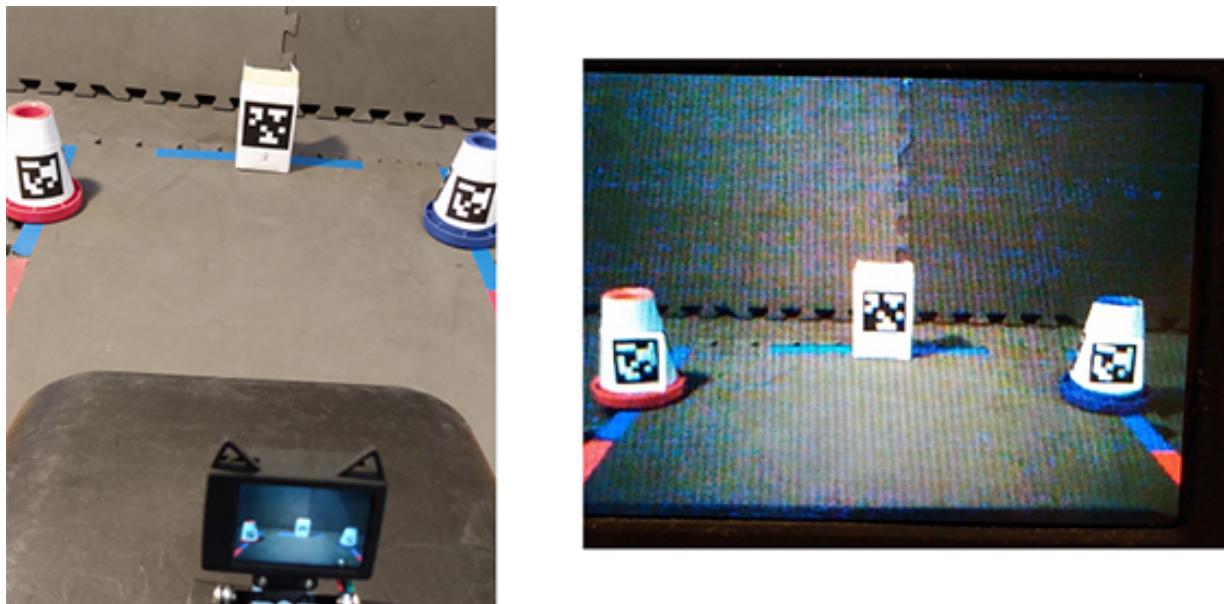


Fig. 43: HuskyLens Viewing 3 Uncategorized Tags

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.

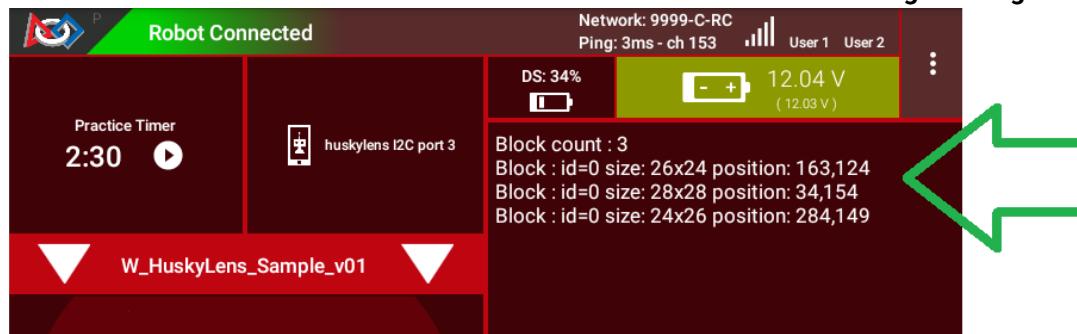


Fig. 44: Telemetry Showing 3 Blocks

### 6.8.6 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.

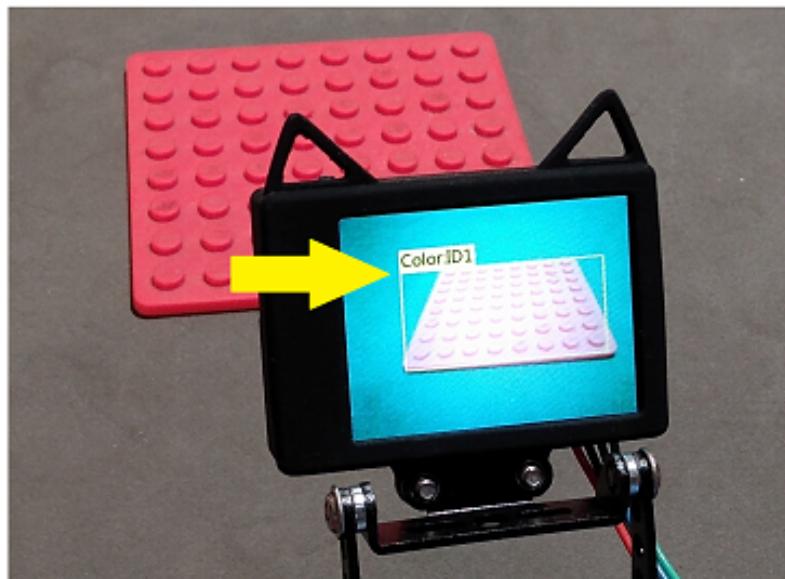


Fig. 45: Red Color ID

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.”

## 6.8.7 Single Color Detection

Aim the HuskyLens at one or more of your color-trained objects.

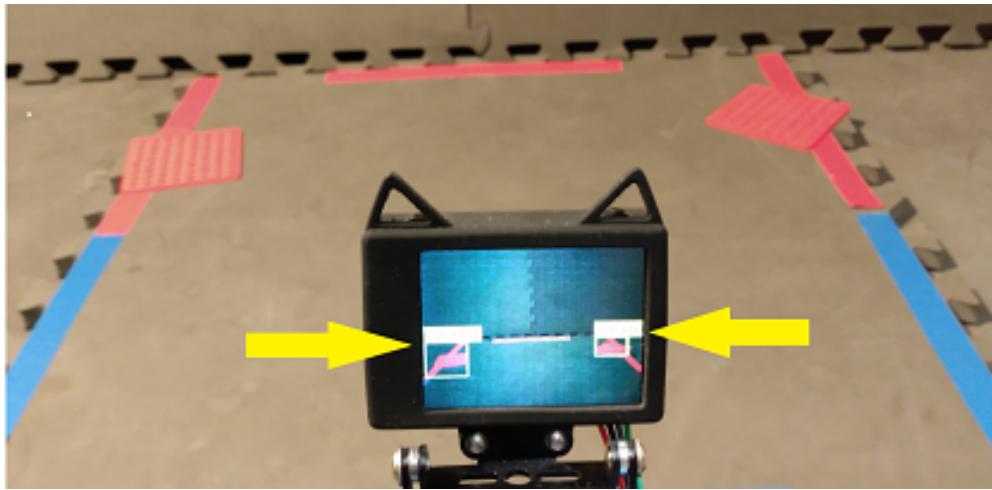


Fig. 46: HuskyLens Detecting Two Red Objects

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:

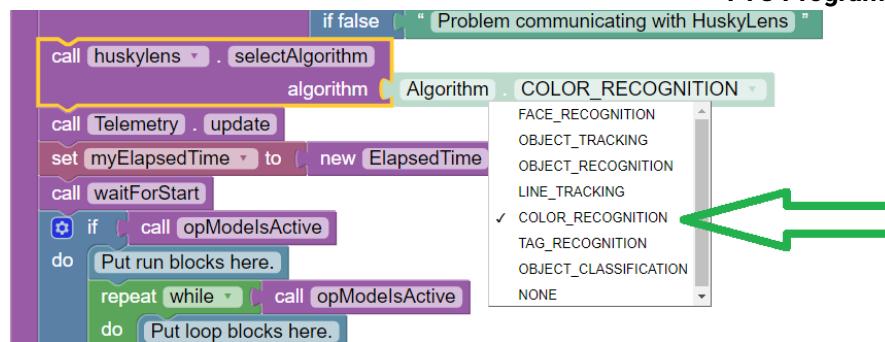


Fig. 47: Selecting COLOR\_RECOGNITION algorithm

```
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.

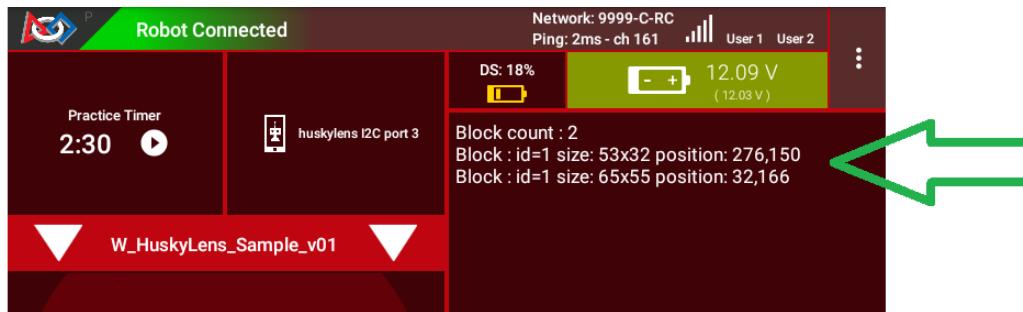


Fig. 48: DS Telemetry Two Objects

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.

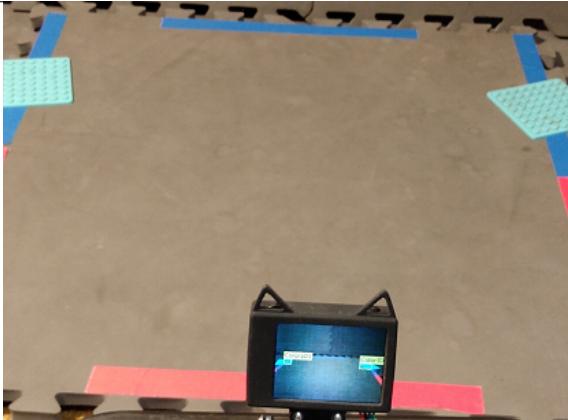


Fig. 49: HuskyLens Two Blue Objects

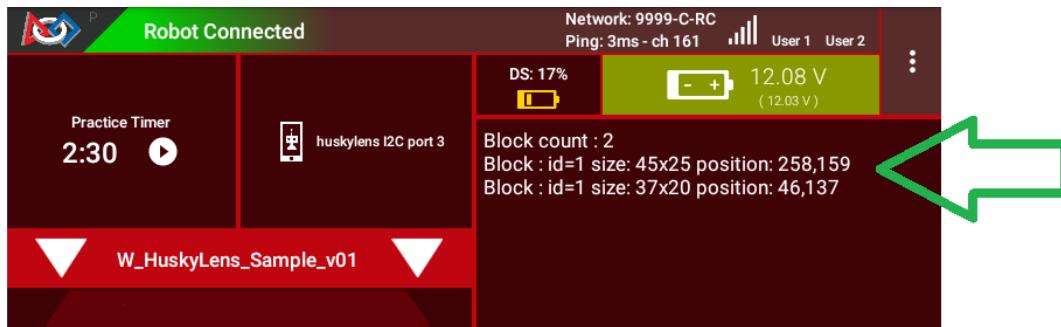


Fig. 50: Telemetry for Two Blue Objects

## 6.8.8 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.

### 6.8.9 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”.

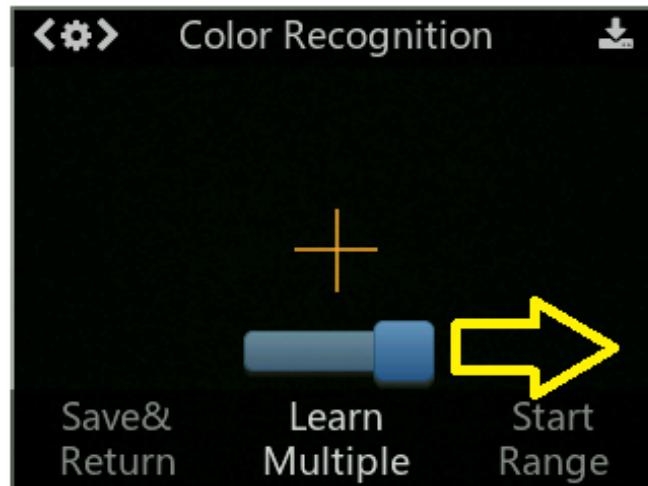


Fig. 51: HuskyLens - Learn Multiple

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!

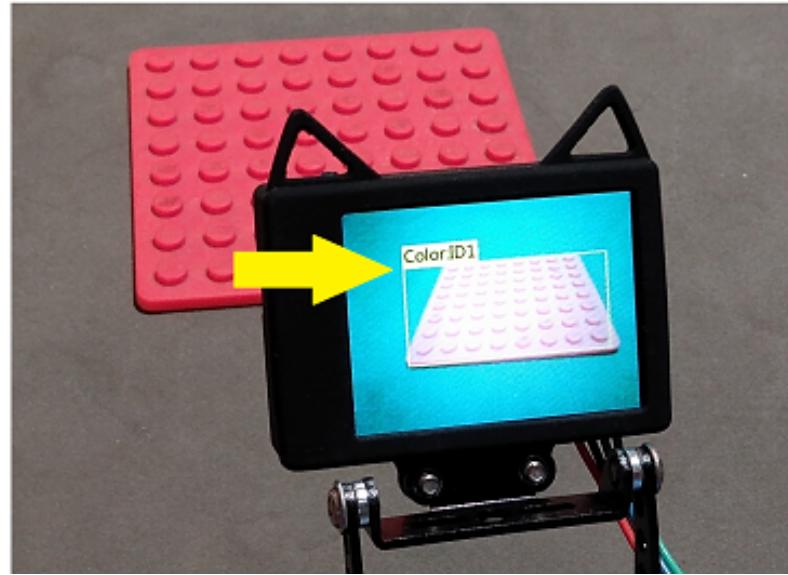


Fig. 52: HuskyLens - RED (Color 1) Trained

**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!

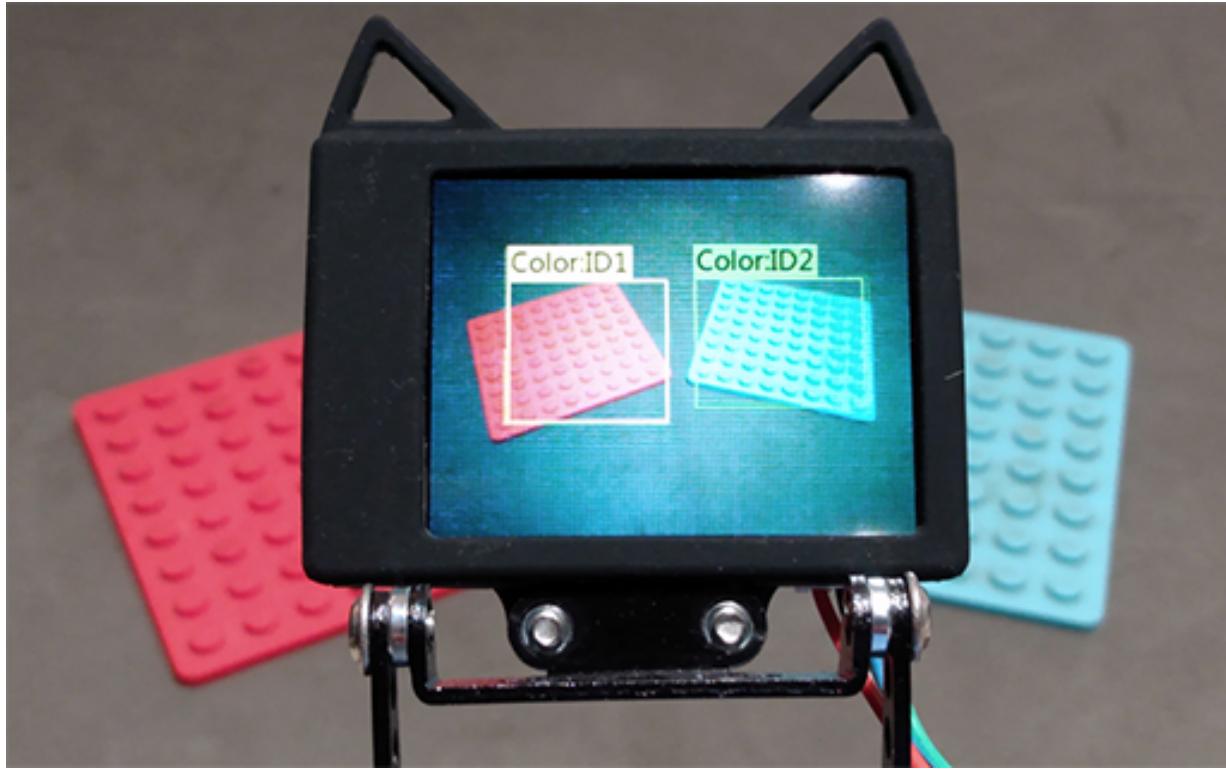


Fig. 53: HuskyLens - Two Colors Trained (ID1 and ID2)

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.

### 6.8.10 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.

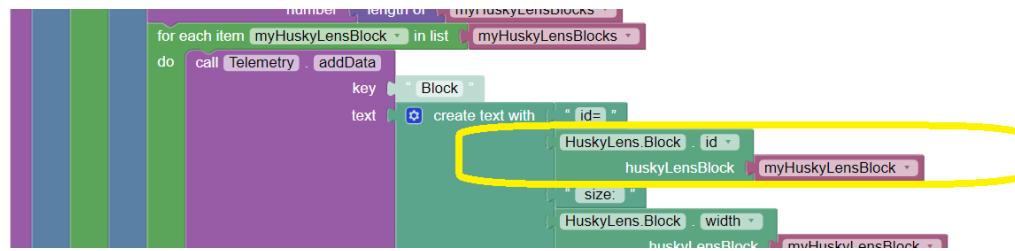


Fig. 54: Adding Telemetry for Colors

Here's the DS Telemetry from the Sample OpMode used above for single color, **with no coding changes**:

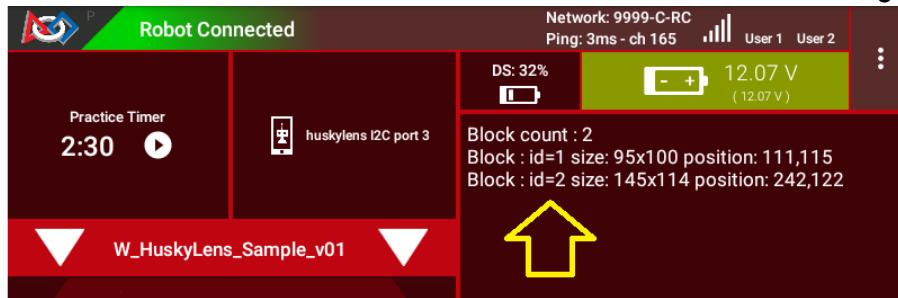


Fig. 55: Example Telemetry showing Both Colors

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.

---

## 6.8.11 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](mailto:westsiderobotics@verizon.net)

## Chapter 7

---

### Additional FIRST Website Resources

- FIRST Website Programming Resources Link

## Chapter 8

---

### Version Information

#### 8.1 Document Information

**Author:** FIRST Tech Challenge

**Version:** 0.2

**Release Date:** 21/08/2024

**Generation Time:** 06:24

#### 8.2 Git Information

**Git Hash:** 88be42263d2a052e1ffc619f3462f07181241886

**Git Branch:** main

**Git Commit Date:** 2024-08-21 06:17:23

**Git Commit Author:** Danny Diaz <texasdiaz@gmail.com>

#### 8.3 Document License

**License:** BSD 3-Clause