FTC Kickoff Programming Presentation

1.0 - Introduction to FTC Programming

Presentation available at http://team2486.org/ftc

Programming Workshop 1.0 Overview

Progamming Workshop 1.0

- Intended audience: rookie or less-experienced teams
- General introduction and overview
- Hardware overview & best practices for a stable robot
- Android Phones Setup & Configuration
- Android Studio Programming Environment
- Simple Java programming examples for using gamepads, motors, and servos
- Time permitting, help with getting Android Studio installed on your laptop
- Programming Workshop 2.0 covers sensors.

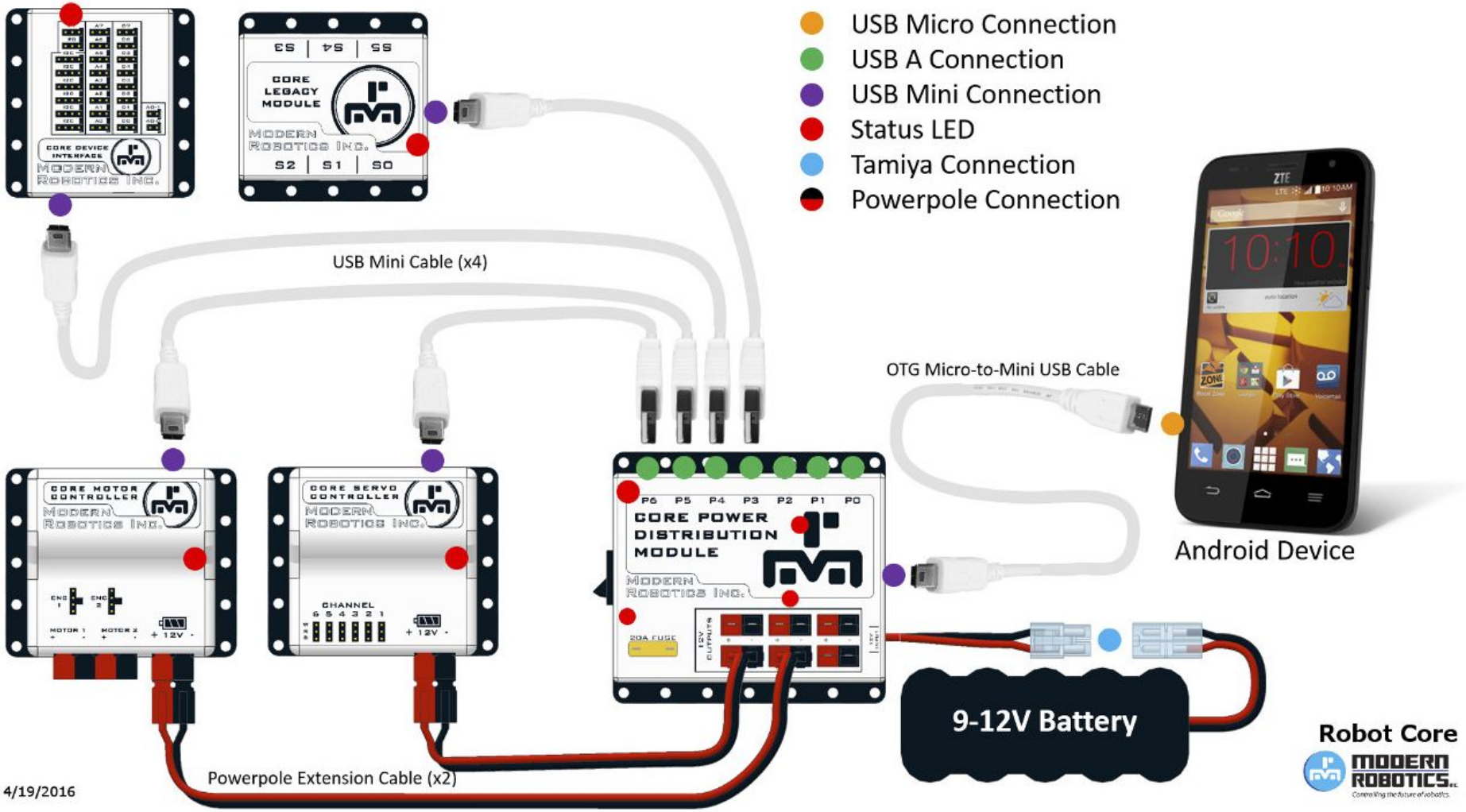
Introduction to FTC Control System

- Control System Hardware
- Supported Sensors
- Best Practices Programming Options
- Programming Resources

FTC Control System Resources

- **FTC Technology Forum:**
  http://ftcforum.usfirst.org/forumdisplay.php?156-FTC-Technology
  - Share thoughts, ideas, and questions with other FTC teams
  - Get latest information about control system software updates and bug reports
  - Provides a communication link to FTC Engineers
  - Experienced teams offer tremendous support for rookie teams and coaches.
  - Must sign up for an account to use the forum.
- **Online FTC Control System Training for Beginners:**
  http://ftc.edu.intelitek.com
  - Overview tutorial intended for beginners.
- **2016-2017 Game Manuals Parts 1 & 2:**
  http://www.firstinspires.org/resource-library/ftc/game-and-season-info
  - Provides game rules on what is and is not allowed as part of the control system.
- **Robot Building Resources:**
  http://www.firstinspires.org/resource-library/ftc/robot-building-resources

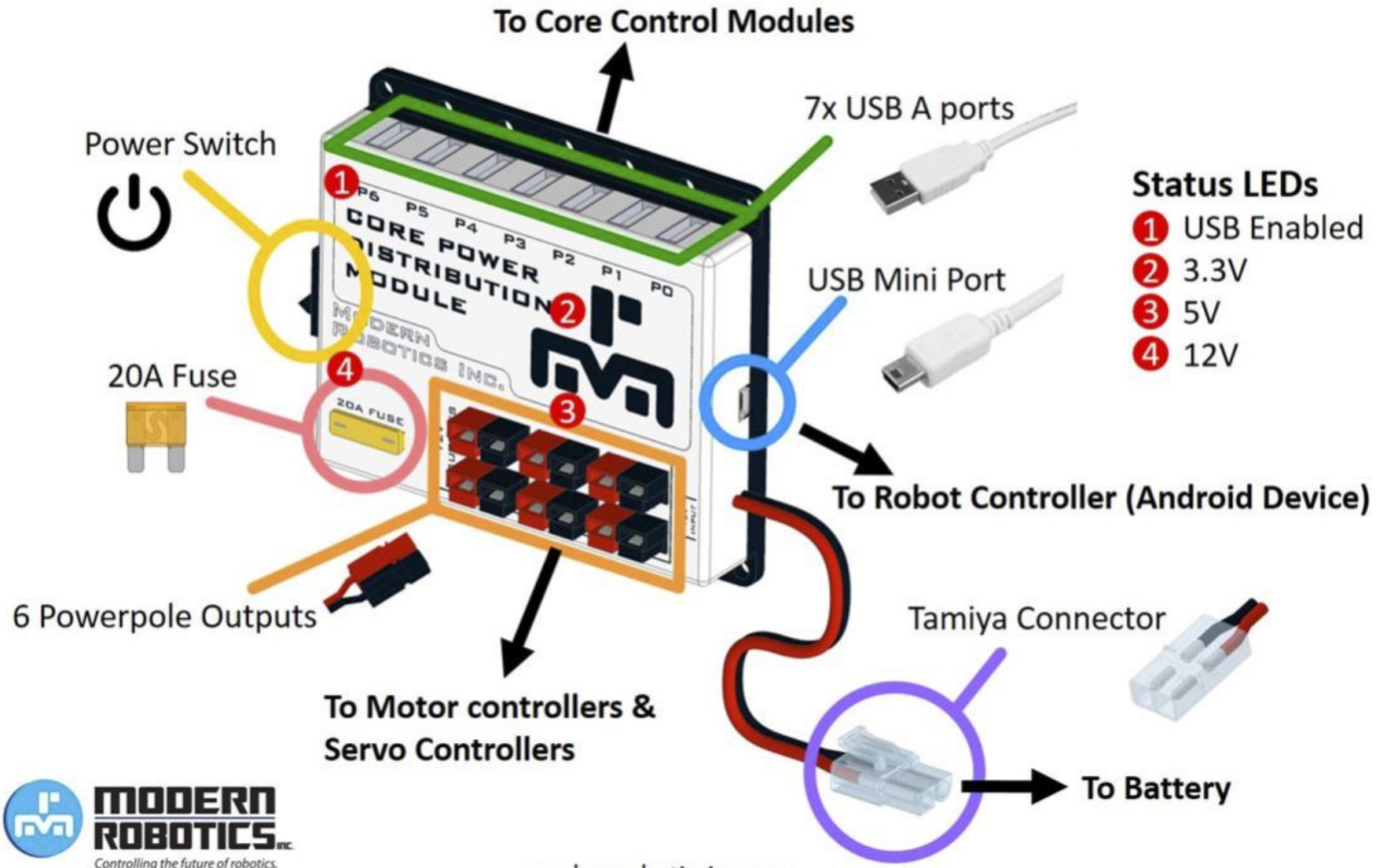
FTC Control System Hardware



4/19/2016

Core Power Distribution Module

45-2200

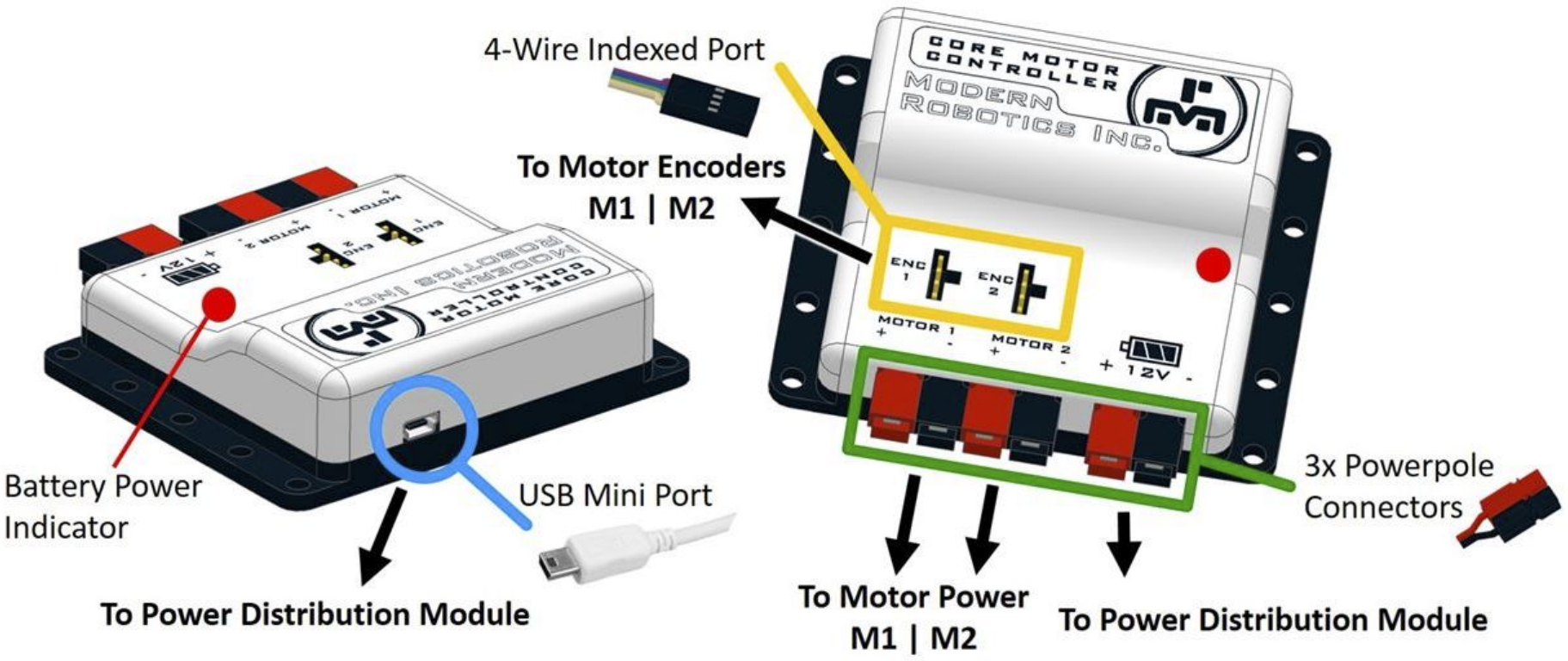


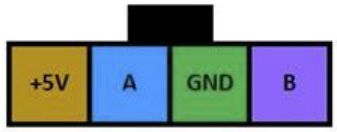
MODERN ROBOTICS INC.
Controlling the future of robotics.

modernroboticsinc.com

Core Motor Controller

45-2203





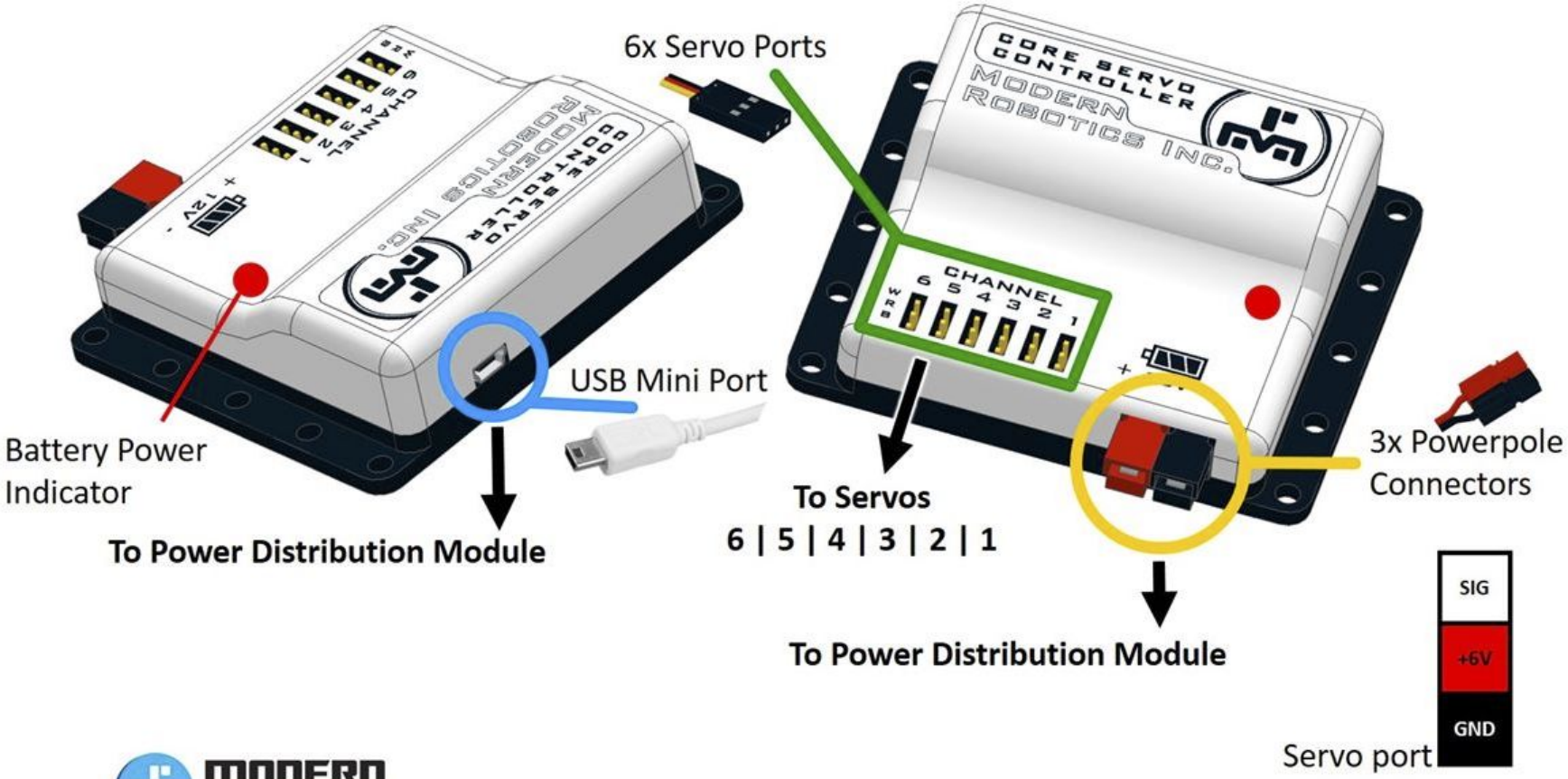
Encoder port

MODERN ROBOTICS INC.
Controlling the future of robotics.

modernroboticsinc.com

Core Servo Controller

45-2204

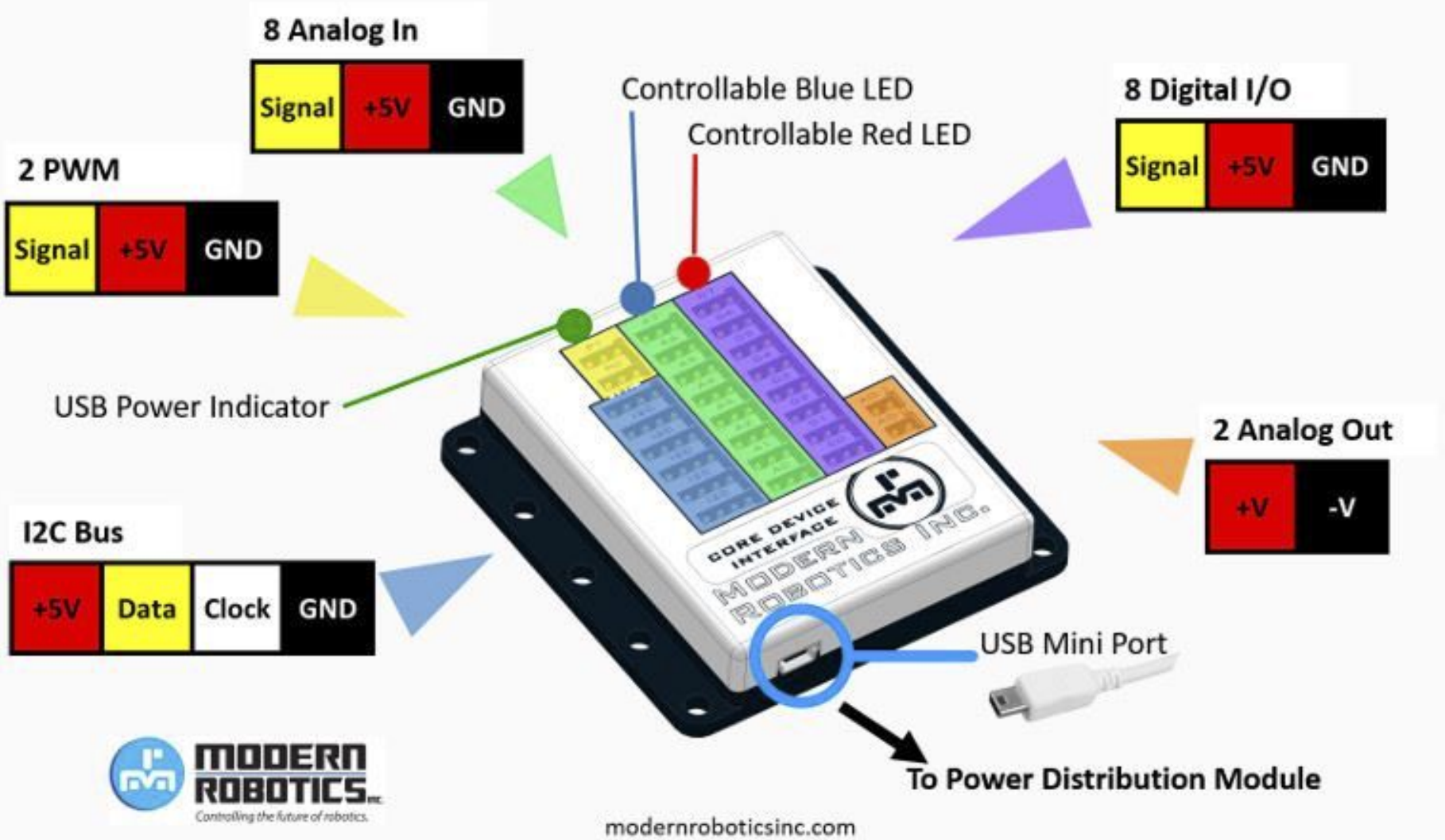


MODERN ROBOTICS inc.
Controlling the future of robotics.

modernroboticsinc.com

Core Device Interface

45-2201

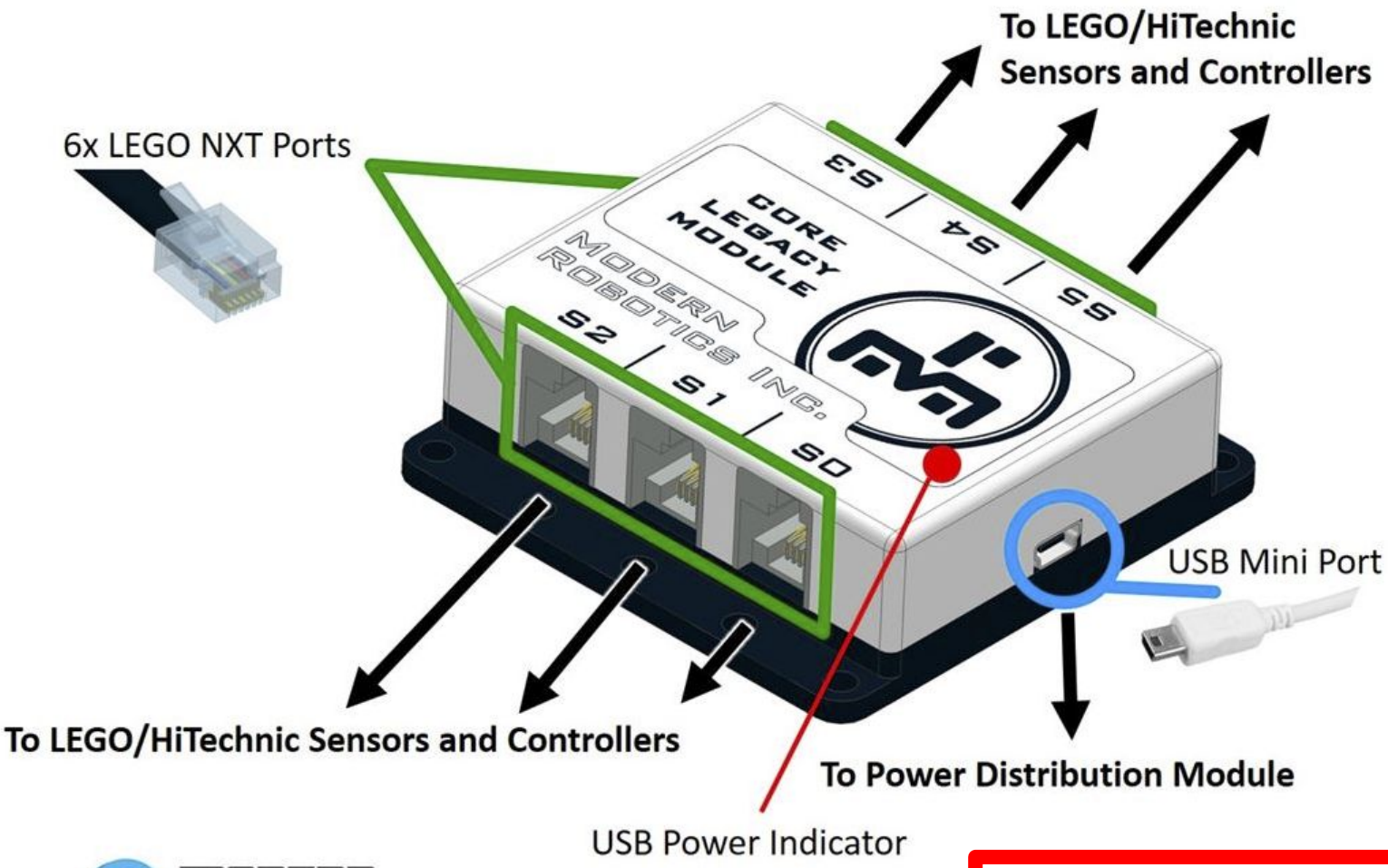


MODERN ROBOTICS INC.
Controlling the future of robotics.

modernroboticsinc.com

Core Legacy Module

45-2202



MODERN ROBOTICS INC.
Controlling the future of robotics.

modernroboticsinc.com

LEGO Ultrasonic Sensor may only be used on ports 4 & 5

2016-2017 FTC Supported Sensors

- DC Motor Encoders (see 2.0 Workshop)
- AdaFruit IMU (Inertial Measurement Unit) - provides 3-axis orientation data
  - http://www.adafruit.com/products/2472
- AdaFruit RGB Color Sensor - three-color sensor
  - http://www.firstinspires.org/sites/default/files/uploads/resource_library/ftc/adafruit-rgb-sensor-buildguide.pdf
- HiTechnic Color Sensor (connected to Legacy Core Module)
  - http://www.hitechnic.com/cgi-bin/commerce.cgi?preadd=action&key=nco1038
- LEGO Light Sensor (connected to Legacy Core Module)
  - https://shop.lego.com/en-US/Light-Sensor-9844
- LEGO Touch Sensor (connected to Legacy Core Module)
  - https://shop.lego.com/en-US/Touch-Sensor-9843
- LEGO Ultrasonic Sensor (discontinued, but still available at Amazon)
  - https://shop.lego.com/en-US/Ultrasonic-Sensor-9846

Supported Sensors (continued)

- Modern Robotics Color Sensor
  - http://www.modernroboticsinc.com/color-sensor
- Modern Robotics Gyro - z-axis orientation vector & 3-axis accelerometer
  - http://www.modernroboticsinc.com/integrating-3-axis-gyro
- Modern Robotics IR Seeker - provided direction from IR signals
  - http://www.modernroboticsinc.com/ir-seeker-v3-2
- Modern Robotics Optical Distance Sensor (short distances < 4 inches)
  - http://www.modernroboticsinc.com/optical-distance-sensor-2
- Modern Robotics Touch Sensor
  - http://www.modernroboticsinc.com/touch-sensor-2
- Want to use a different off-the-shelf sensor?
  - Good News! FTC Control System SDK now comes with open source code.
  - Use professionally developed code as a model for building interfaces to new sensors.
  - Source code available in “jniLibs” folder.
  - Experienced programmers only!

Resources:

➔ **Visit FTC Technical Form for suggestions and help on using sensors.**

➔ **See example programs in FTC Control System SDK.**

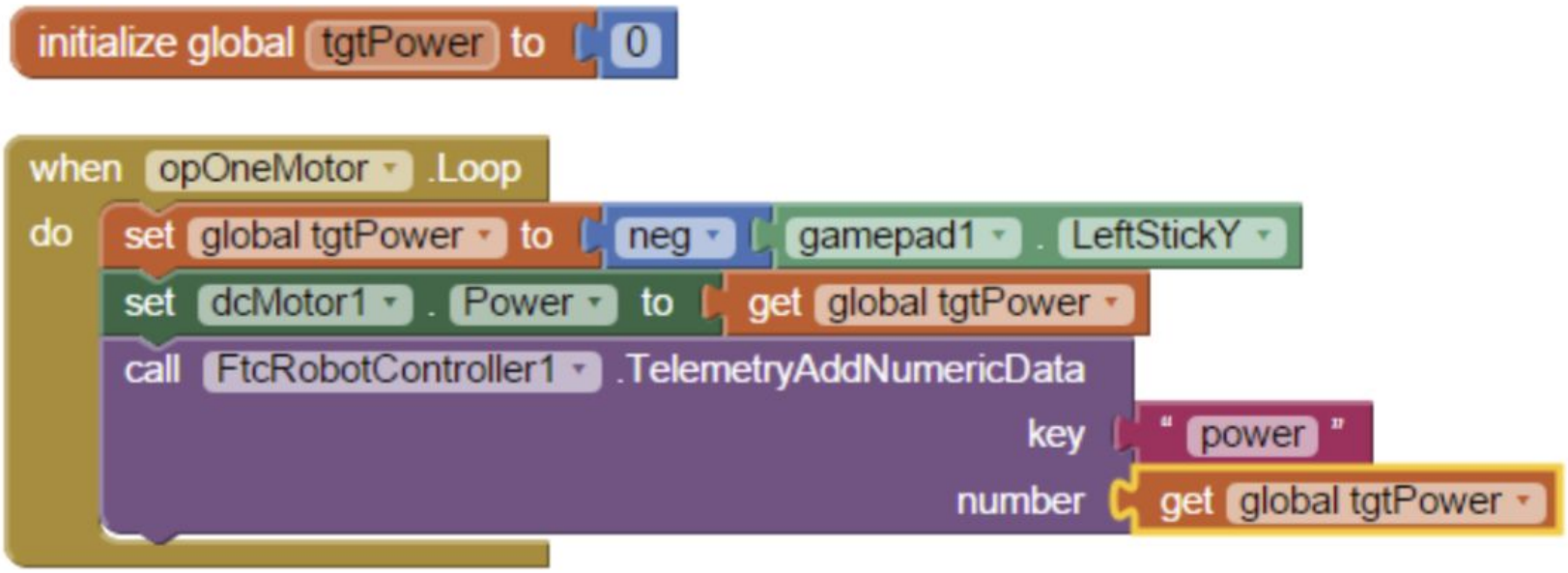
Programming with MIT App Builder

Programming design tool makes it easy to create your own robot Android App with a user-friendly drag-and-drop visual interface.

- Browser-based application available as a web site run by MIT.
- MIT App Builder may be a good option for younger students or rookie teams that lack programming experience.
- Get started quickly with programming but drag-and-drop interface may become tiresome for building more complex programs.
- App Builder Tutorial from Oregon Robotics: https://ortop.org/ftc/AppInventor/
- Training manuals, installation quides, and example programs found here: https://frc-events.firstinspires.org/ftcimages/2016
- Check out the MIT App Builder Support Forum: http://ftcforum.usfirst.org/forumdisplay.php?160-MIT-App-Inventor

MIT App Builder - Code Snippet.

Drag programming blocks to the viewer window:



Programming with Java & Android Studio

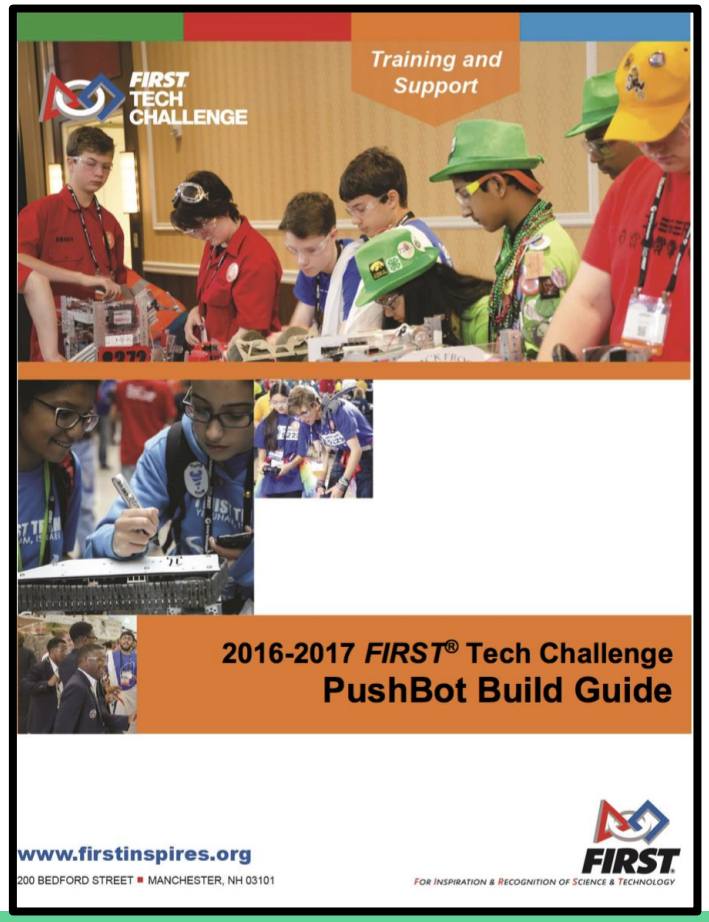
- Java -- a modern line-coding programming language offering efficient and rapid programming development.
- Android Studio -- professional-level integrated development environment (IDE) used worldwide.
- Programming in Java is a valuable skill for anyone interested in pursuing a technical career.
- More to learn to get started but once learned is an excellent tool for efficiently developing more complex and intricate robot algorithms.
- Recommended for older students on programming experience.
- Many excellent programming examples are provided with the FTC Control System Software Development Kit (SDK)
- The focus of this Kickoff presentation is on Java programming & Android Studio.

Are you a rookie team looking for a way to start?

Build the FTC-approved “PushBot” Robot!

2016-2017 PushBot Build Guide:

http://www.firstinspires.org/resource-library/ftc/robot-building-resources



- Detailed instructions for building, wiring, and configuring PushBot.
- Rookie team members learn good building practices by following build instructions.
- FTC Control System SDK comes with programs that operate the PushBot.
- Adapt PushBot robot to needs of tournament play
- Or build and design a new robot with the knowledge gained from building Pushbot

Best Practices for a Stable Robot

During tournament play keep robot batteries fully charged--including both android phones and 12V battery.

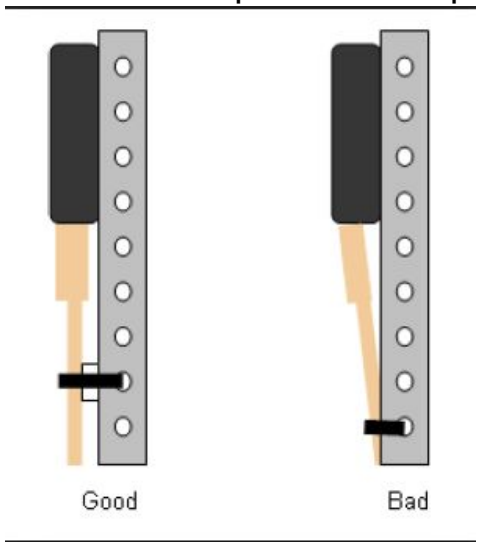
- Allowing battery charge to get critically low is the **biggest** source of control system instability at tournaments.
- Charge android phones and 12V batteries between matches.
- At tournaments assign a team member the task of keeping batteries fully charged.
- Consider having a duplicate set of fully-charged 12V batteries and android phones at tournaments.
- Bring chargers to Queuing Stations to charge android phones while waiting for your match.

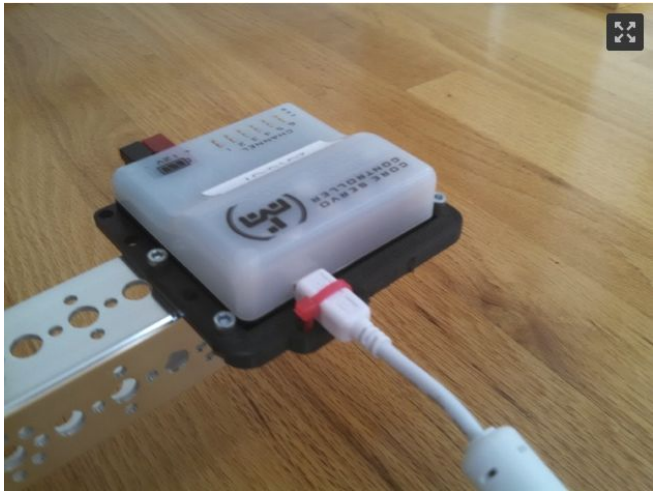
Best Practices for a Stable Robot

Minimize robot instability problems due to loose USB connections and robot vibration.

- Using zip ties secure phone and control-module USB cable connections to the frame.
- Avoid strain on connections by keeping cable connections aligned and straight
- For best results use 3D-printed strain-relief connectors

Example: Use of zip ties to secure connections

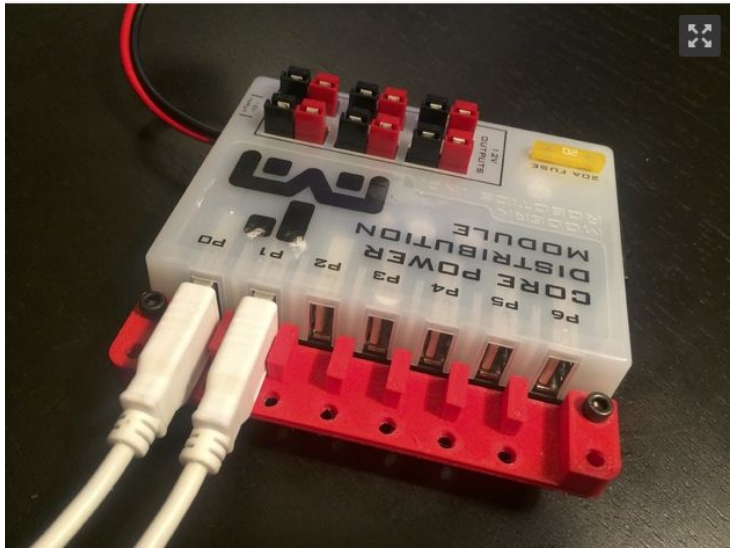


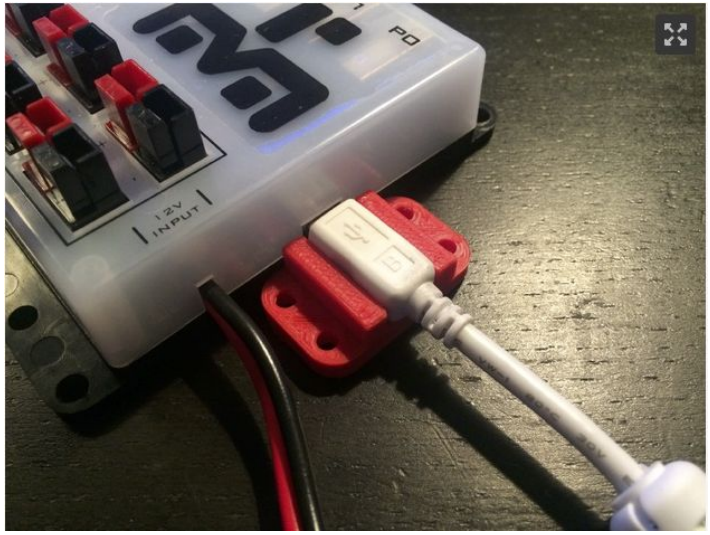


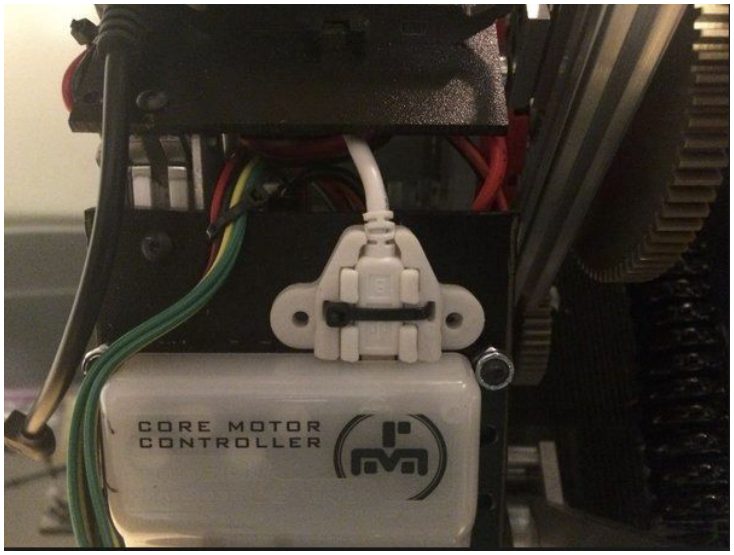
Best Practices for a Stable Robot

For best results use strain-relief connectors.
3D printer models are available here:

http://www.thingiverse.com/Techied/collections/ftc
http://www.yeggi.com/q/first+tech+challenge/?s=tt
http://www.thingiverse.com/thing:1145697







2016-2017 FTC Control System SDK is now available

https://github.com/ftctechnh/ftc_app

Android Phones

- Android phone models for the 2016 season
- Overview of the FTC Driver Station and Robot Controller apps
- Android phone setup and configuration

Overview of phone setup

- Allowed Phones
- Physical Setup
- Necessary Settings
- Necessary Software To Install

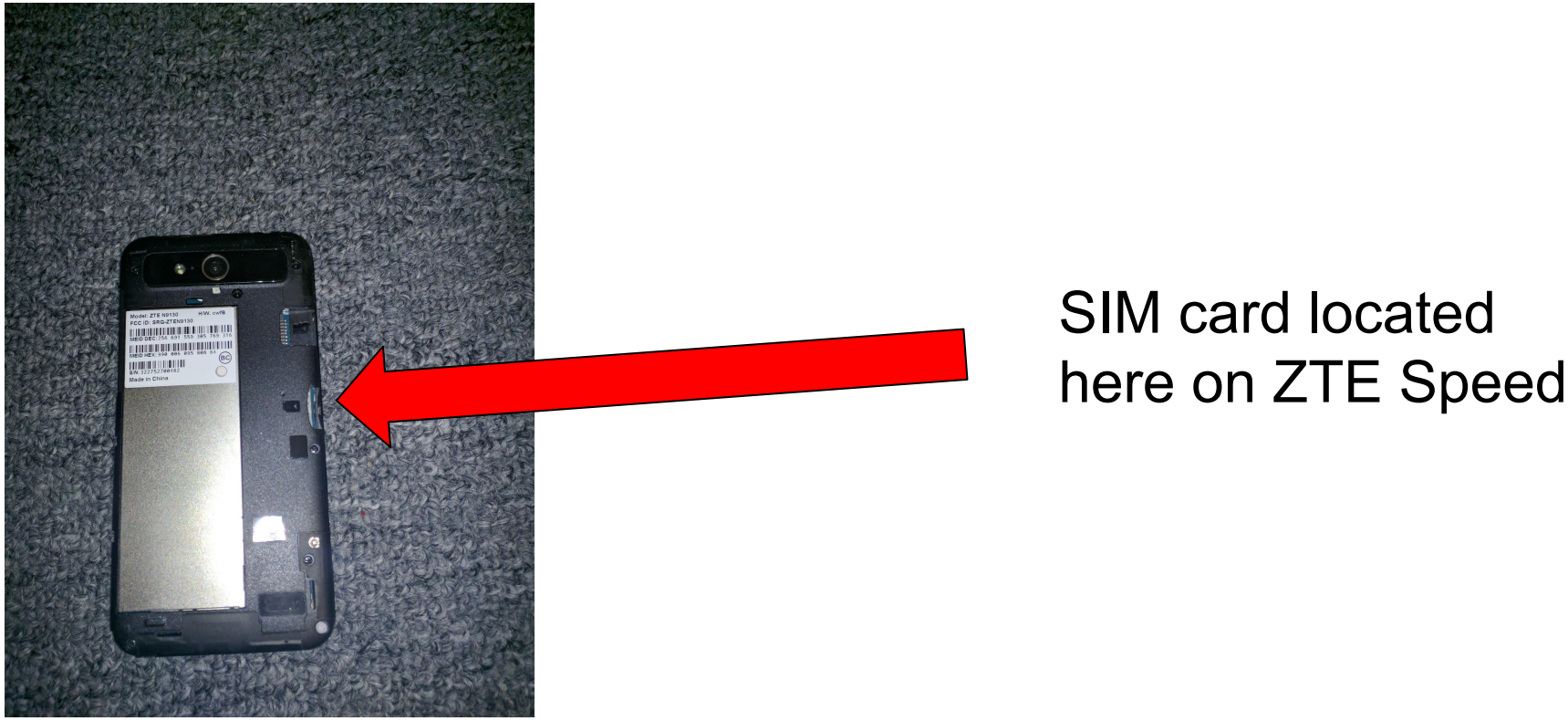
Allowed phones

The following phones are permitted for use with the FIRST Tech Challenge control system

- ZTE SPEED
- MOTO G (2nd & 3rd gen)
- NEXUS 5

Physical Preparation

Before turning on your phone for the first time, remove the SIM card from your phone. Refer to the operator's manual if necessary.



Software Setup

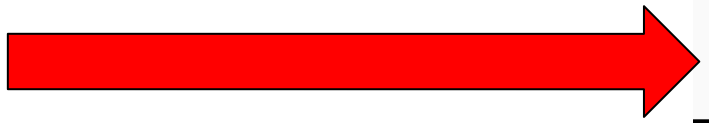
The following steps are required for proper function of the FIRST Tech Challenge control system

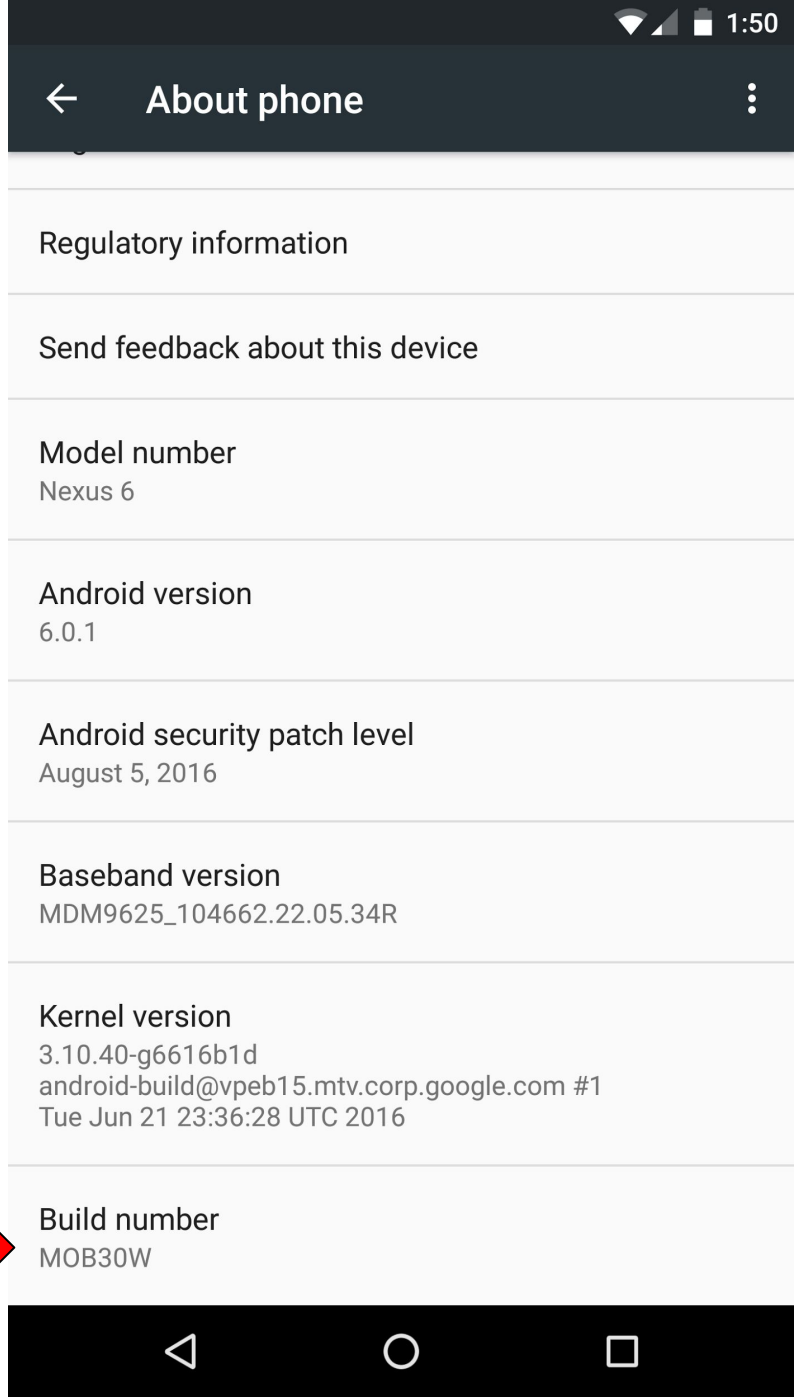
- Enabling developer mode
- Enabling USB debugging
- Disabling Google notifications
- Setting the sleep timer properly
- WIFI Direct
- Additional networking settings

Enabling Developer Mode

Open settings, and go to the “About Phone” section, and tap “Build number” (located at the bottom) seven times.

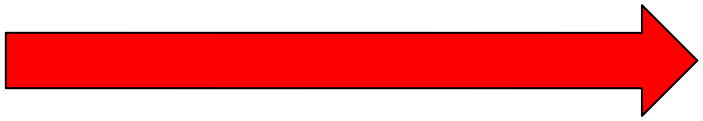
(NOTE: Screenshots are from Android 6.0.1, but in principle they are the same)

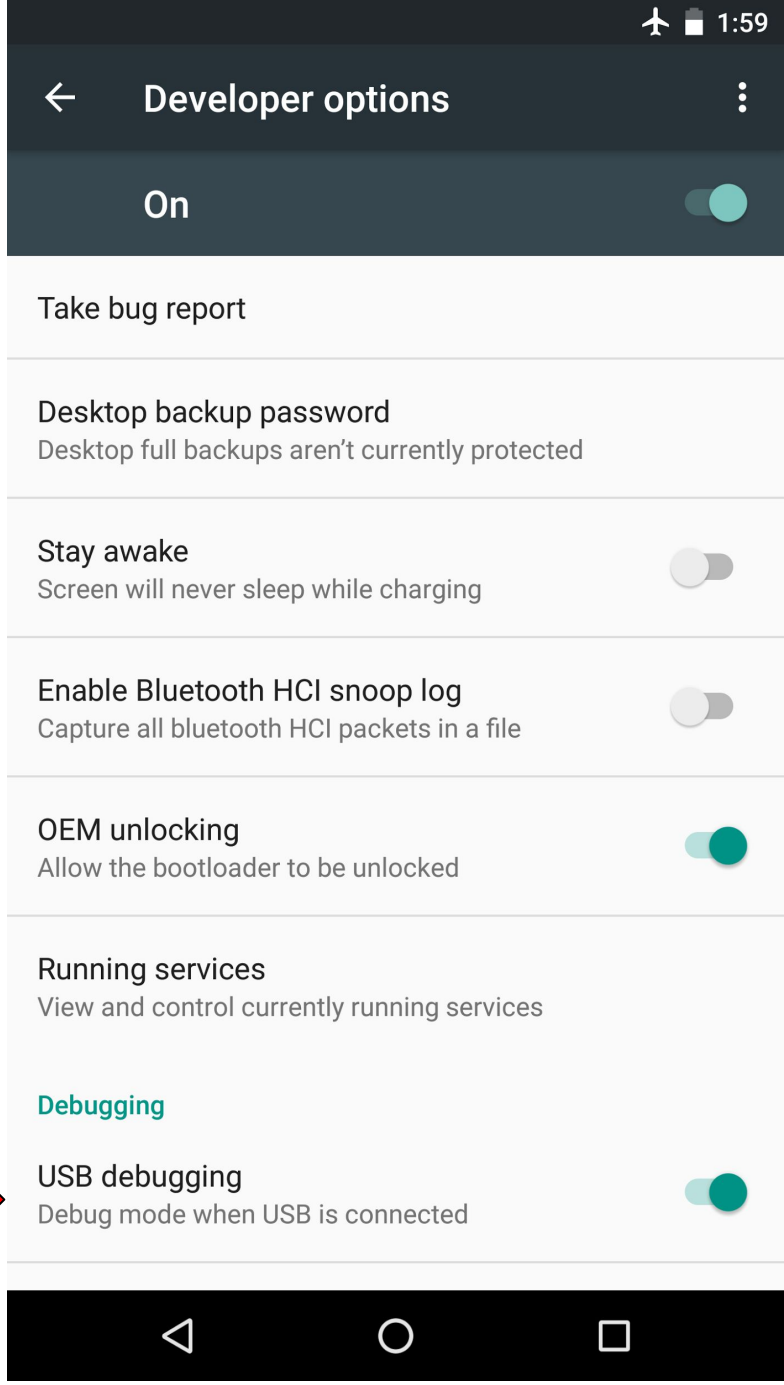




Enabling ADB

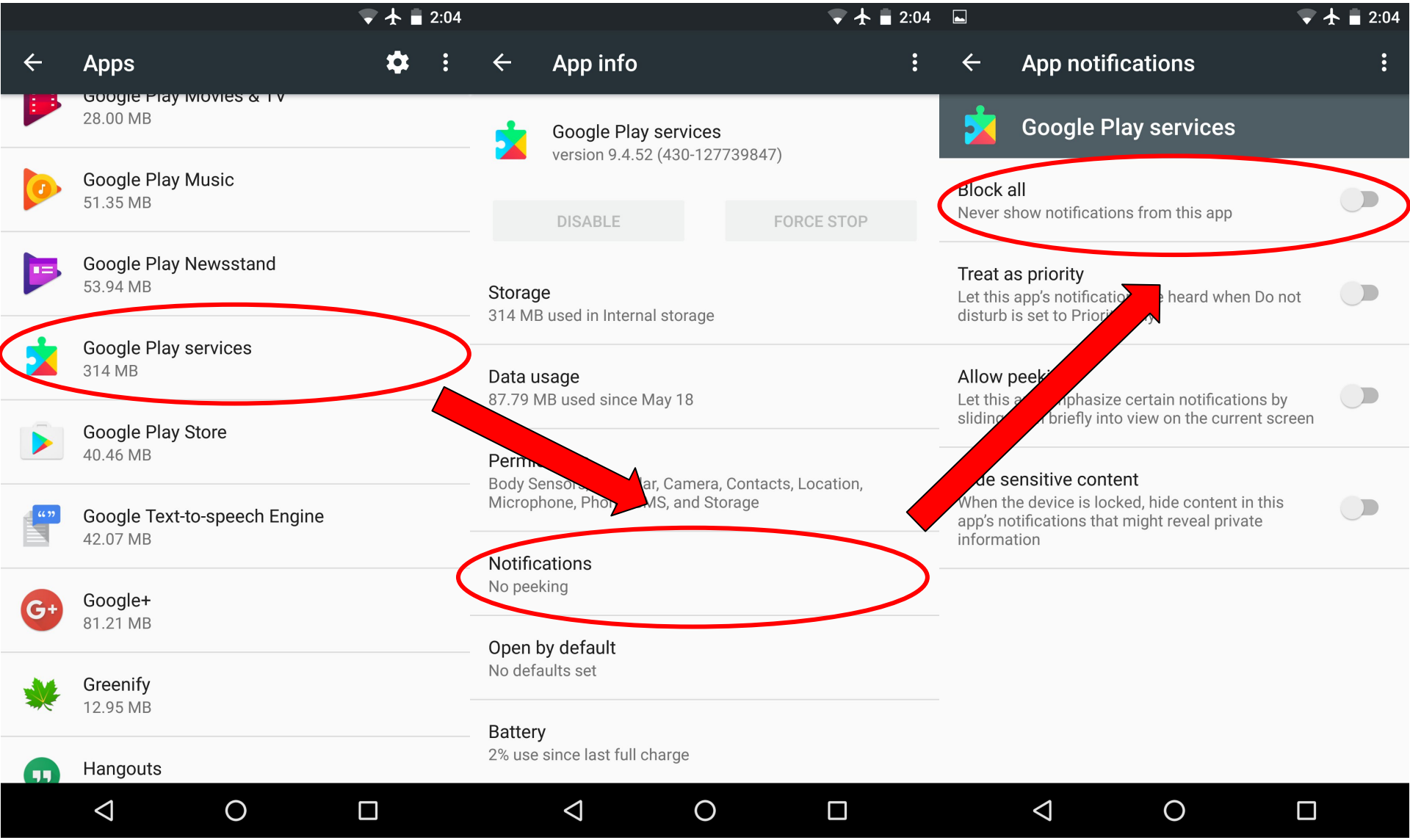
Under the main settings, there should now be a new tab labeled “Developer Options”. Within this tab, check the box labeled “Enable USB Debugging”





Disabling Google Notifications

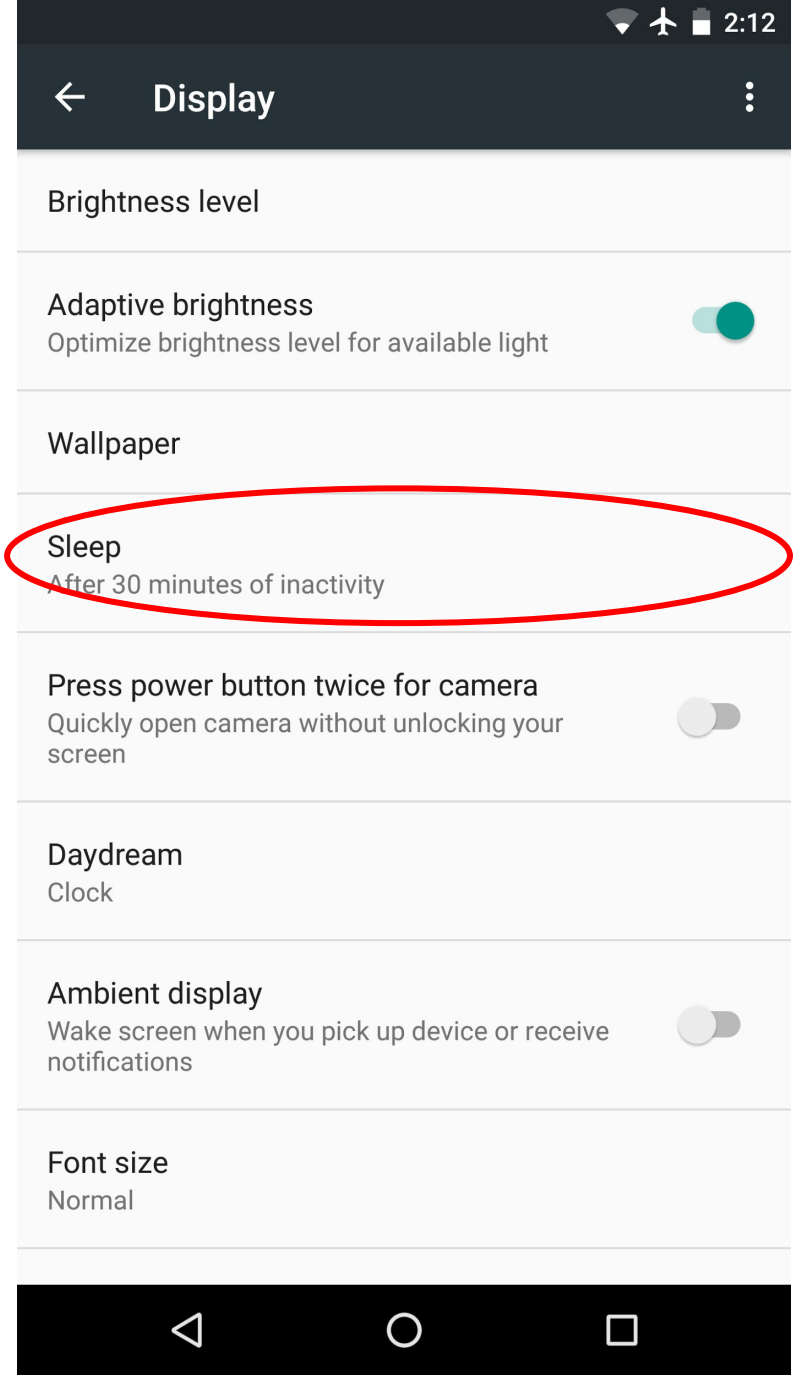
To ensure that the phone is not accidently updated to android 5.0, it is recommended that google services notifications be disabled. To do this, go to Settings > Apps > Google Play services, and disable all notifications.


2:04
Apps
Google Play Movies & TV
28.00 MB
Google Play Music
51.35 MB
Google Play Newsstand
53.94 MB
Google Play services
314 MB
Google Play Store
40.46 MB
Google Text-to-speech Engine
42.07 MB
Google+
81.21 MB
Greenify
12.95 MB
Hangouts
2:04
App info
Google Play services
version 9.4.52 (430-127739847)
DISABLE
FORCE STOP
Storage
314 MB used in Internal storage
Data usage
87.79 MB used since May 18
Perm
Body Sensor
ar, Camera, Contacts, Location,
Microphone, Pho
MS, and Storage
Notifications
No peeking
Open by default
No defaults set
Battery
2% use since last full charge
2:04
App notifications
Google Play services
Block all
Never show notifications from this app
Treat as priority
Let this app's notificatio
e heard when Do not
disturb is set to Priori
Allow peeki
Let this a
phasize certain notifications by
slidin
briefly into view on the current screen
de sensitive content
When the device is locked, hide content in this
app's notifications that might reveal private
information

Setting Up The Sleep Timer

To ensure the phone does not go to sleep while running, be sure to set the sleep timer to be as long as possible.

(Settings > Display > Sleep)



WIFI Direct

To comply with FTC control system rules, your phone must be named correctly in WIFI Direct. To do this, go to Settings > Wifi > Wifi Direct > Rename device. Device should be named TEAMNUMBER-RC for the robot controller and TEAMNUMBER-DS for the driver station. Note: the path maybe Settings > Wifi > Advanced > Wifi Direct > Rename Device on some phones.

Additional Phone Settings

Other required settings are as follows:

- Airplane mode: ON
- Wifi: ON
- Bluetooth: OFF
- Also remember, don't connect to any wifi networks
- Screen rotation: OFF

Important Game Rules Relating To Control System

For details consult the 2016-2017 Game Manual Part 1:

<RE06> Allowed phones: ZTE SPEED, Motorola Moto G 2nd or 3rd Generation, Google Nexus 5.

<RS01> Android device names must be XXXXX-DS on drive station, and XXXXX-RC where XXXXX is your team number

<RS02> Allowed programming environments: Android Studio, App Inventor, Java Native Interface & Android Development Kit.

<RS03> Allowed Android Operating Systems: ZTE Speed - 4.4 (KitKat), Moto and Nexus devices - 6.0 or higher (Marshmallow)

<RS08> Android device must be set to Airplane Mode on, WiFi on, Bluetooth off

<RS09> WiFi Direct Channel Changing App must be installed on a ZTE phone.

Android Studio Programming Environment

- Installation tutorial
- Getting around in Android Studio

Basics of Android Studio

- Android Studio is an **i**ntegrated **d**evelopment **e**nvironment (**IDE**)
- Used for programming Android applications
- The **J**ava **D**evelopment **K**it (**JDK**) is required for using Android Studio
- FTC robot controller app is imported into Android Studio, and then users can add code and upload to phones

Android Studio System Recommendations

Windows

- Microsoft® Windows® 7/8/10 (32- or 64-bit)
- 2 GB RAM minimum, 8 GB RAM recommended
- 2 GB of available disk space minimum, 4 GB Recommended (500 MB for IDE + 1.5 GB for Android SDK and emulator system image)
- 1280 x 800 minimum screen resolution
- Java Development Kit (JDK) 8
- For accelerated emulator: 64-bit operating system and Intel® processor with support for Intel® VT-x, Intel® EM64T (Intel® 64), and Execute Disable (XD) Bit functionality

Mac

- Mac® OS X® 10.8.5 or higher, up to 10.11.4 (El Capitan)
- 2 GB RAM minimum, 8 GB RAM recommended
- 2 GB of available disk space minimum, 4 GB Recommended (500 MB for IDE + 1.5 GB for Android SDK and emulator system image)
- 1280 x 800 minimum screen resolution
- Java Development Kit (JDK) 6

Installing Android Studio

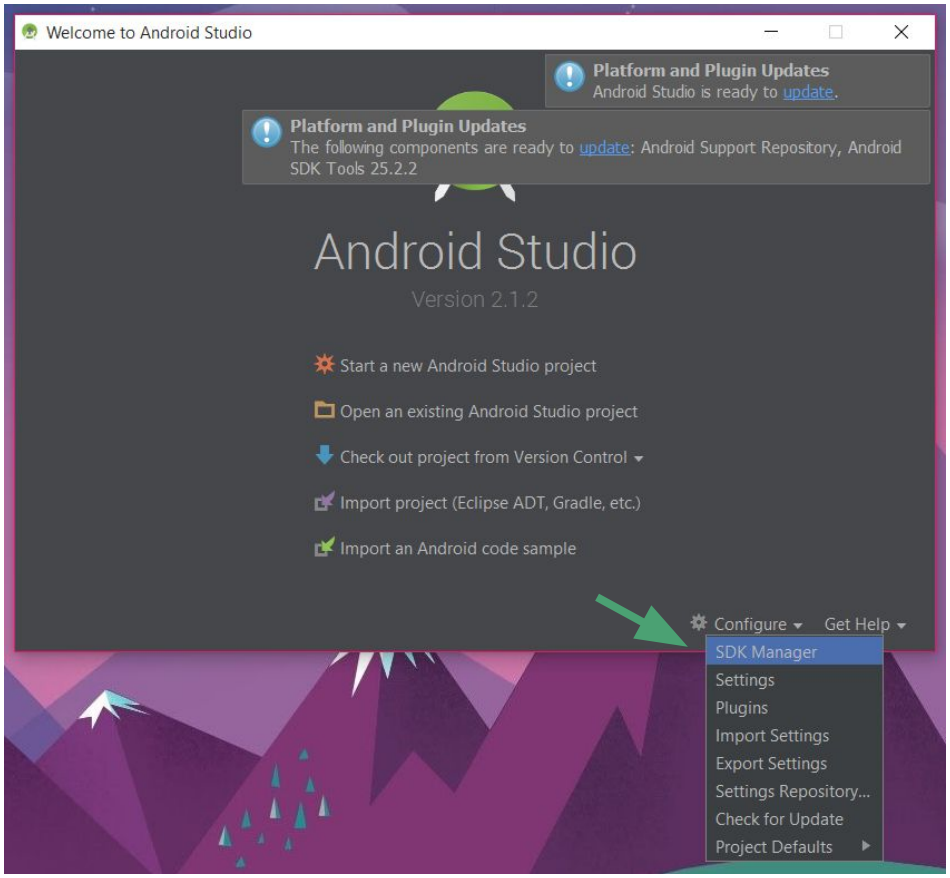
1. Download the Android Studio and JDK installer
    - Select the appropriate architecture for your development computer (x86, x64)
    - The FTC SDK requires 2.1.x minimum Android Studio version
2. Install the JDK
3. Install Android Studio

SDKs

- A **S**oftware **D**evelopment **K**it (**SDK**) includes the necessary files to install software for a specific platform
  - In our case, the platform is Android, and we are targeting a specific version
- The API 23 SDK and API 19 SDK **required**, regardless of Android phone version.

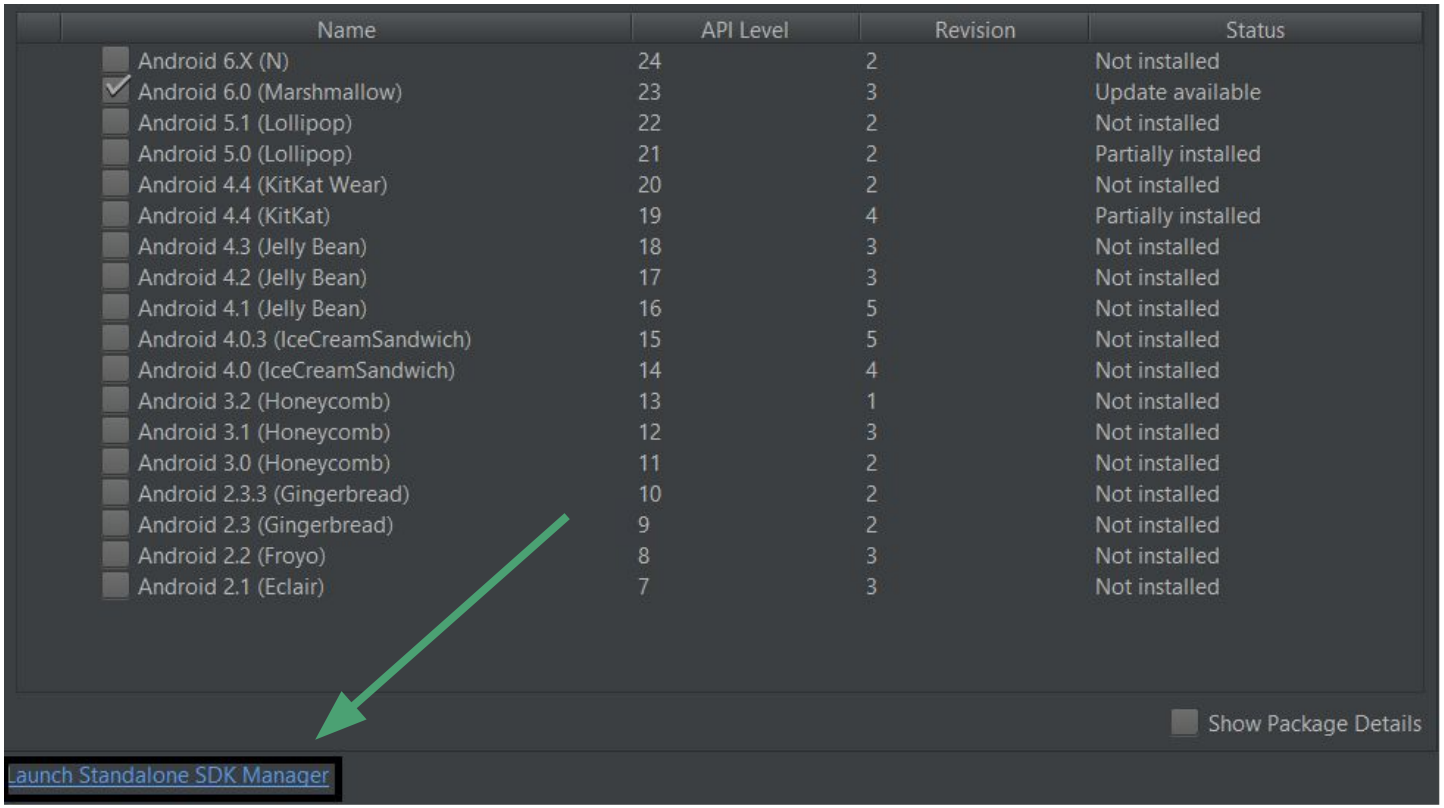
Adding SDKs

- Open Android Studio
- Click on Configure -> SDK Manager



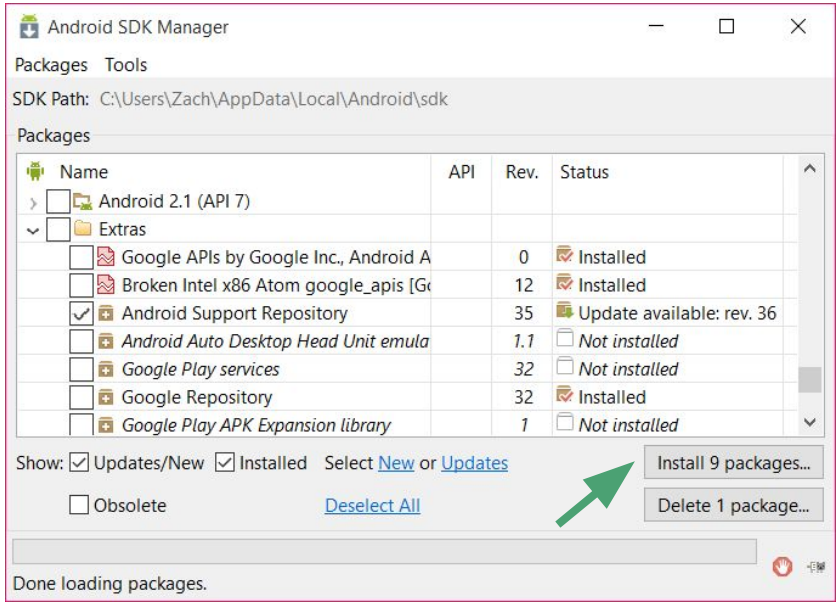
Adding SDKs

- Click on "Launch Standalone SDK Manager"



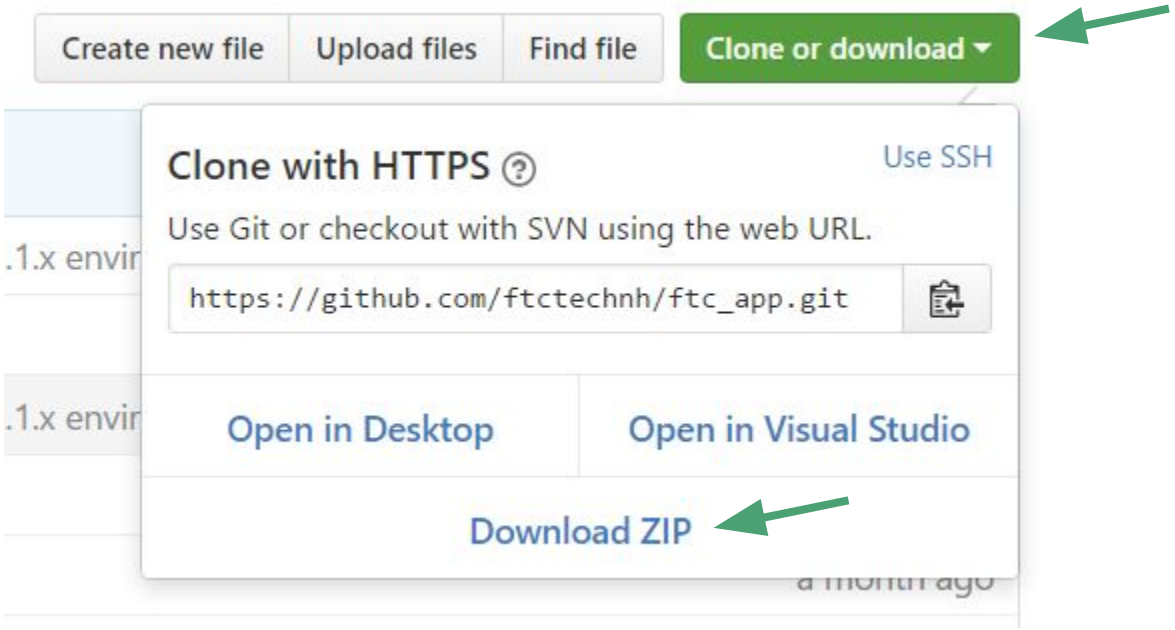
Adding SDKs

- Select "Tools -> SDK Build Tools 23.0.3"
- Select API 23 & 19
- Select "Extras -> Google USB Driver"
- Click "Install"



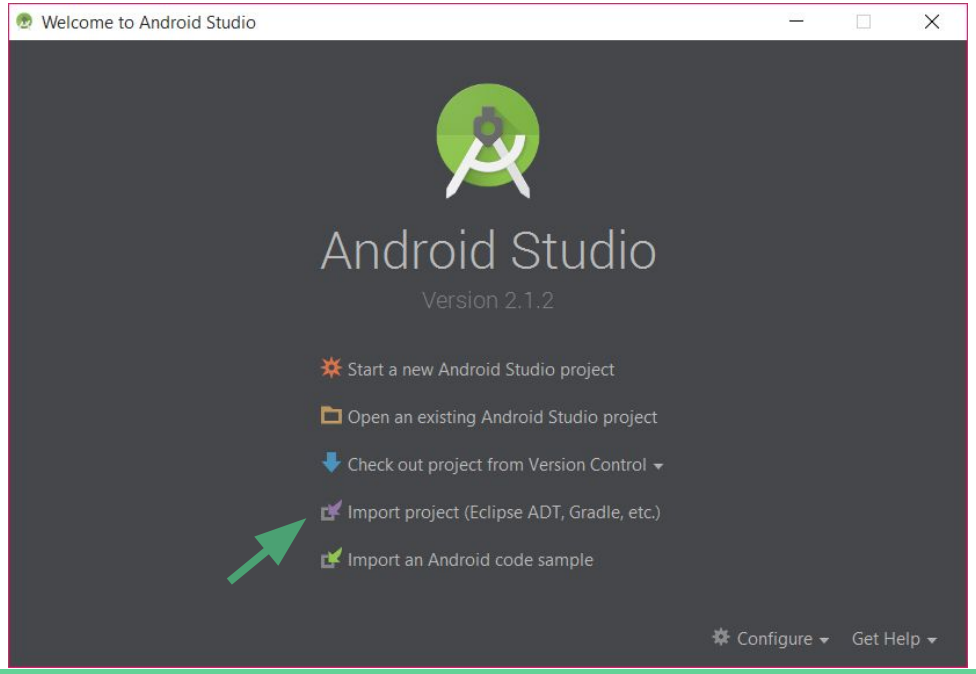
Getting the FTC Robot Controller App

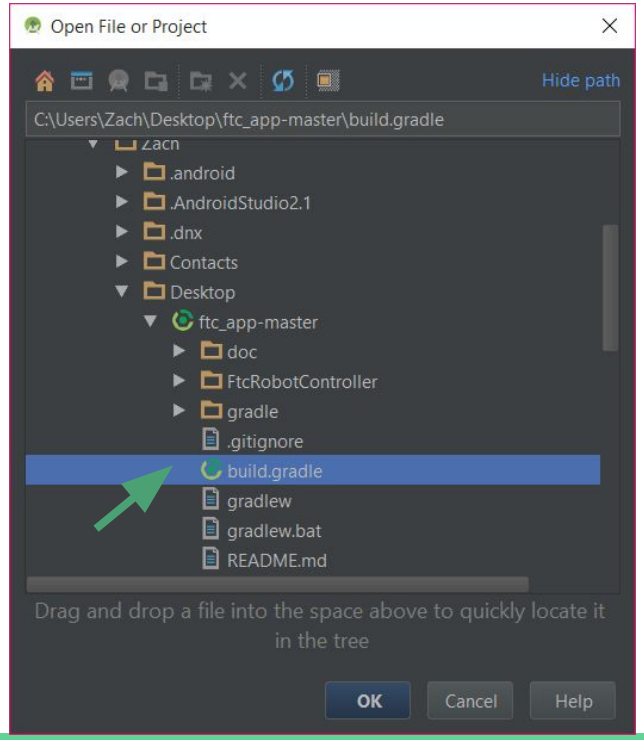
- Visit https://github.com/ftctechnh/ftc_app and select "Clone or download -> Download ZIP"
- Save the file and extract the folder



Importing the Project into Android Studio

- Unzip the ftc-app zip file
- Open Android Studio and click on "Import project (Eclipse ADT, Gradle, etc...)"
- Select ftc_app-master -> build.gradle and click "OK"



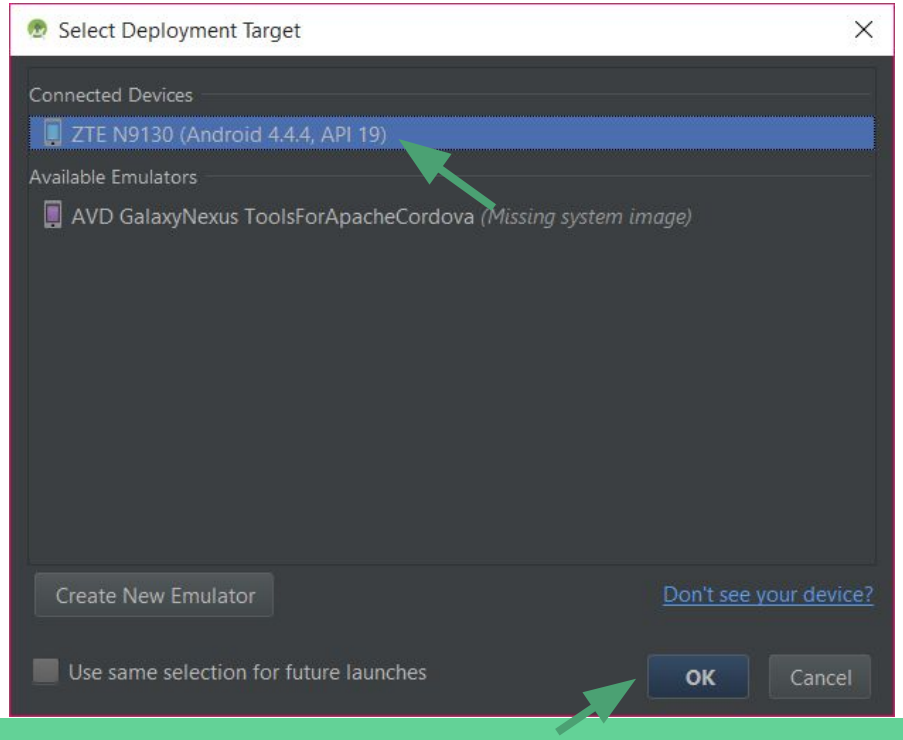


Downloading to the Robot Phone

- Connect the phone to the computer through the USB cable
- Click on the green arrow in the toolbar to upload the code to the phone

Downloading to the Robot Phone

- Under connected devices, select the appropriate phone and click "OK"
- *Note: For the first-time connection, click on "Authorize device" on the phone.*

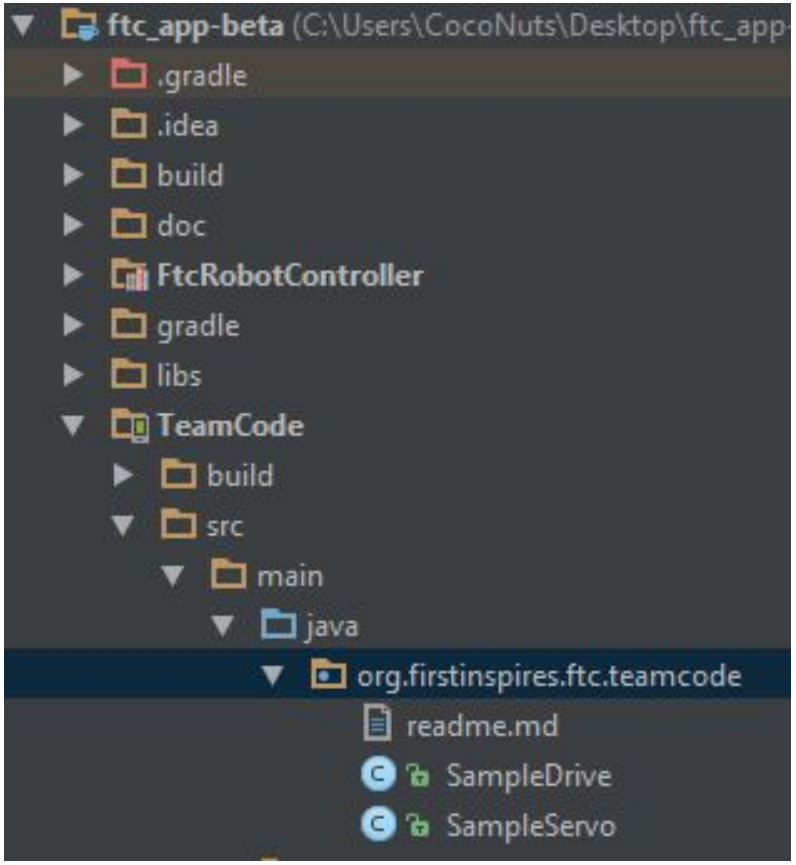


Simple Robot Program Examples

- Overview of the FTC Control System Software Development Kit (SDK)
- Using the gamepad
- Controlling motors and servos with the gamepad
- Displaying critical data on the Driver Station

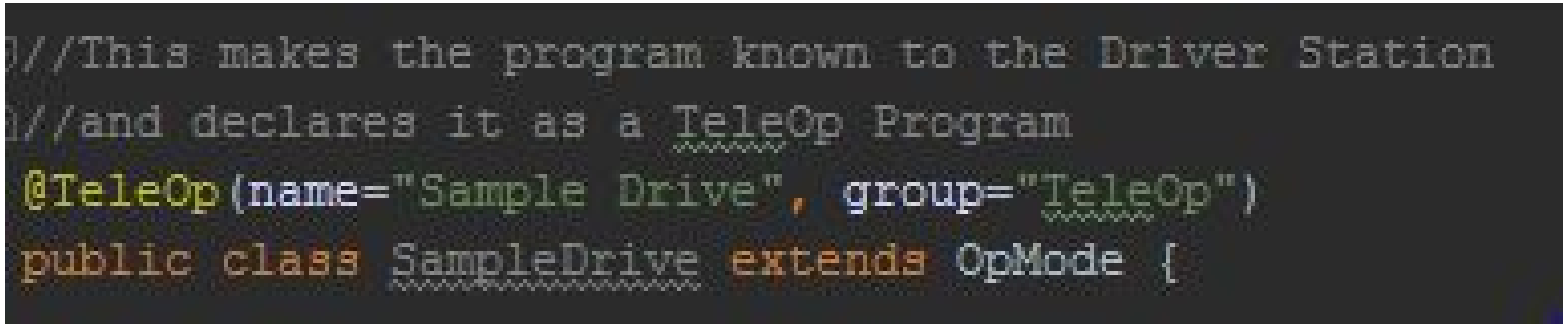
Where to Put Your Code?

- Navigate to ftc_app_master\TeamCode\src\main\java\org\firstinspires\ftc\teamcode and right-click then click New > Java Class to add new code.



How to Make a TeleOp Program

- Before the `public class SampleDrive` (which is automatically created), type `extends OpMode`. This makes allows your program to inherit attributes that are necessary for a TeleOp Program.
- In the picture below, the line `@TeleOp(name="Sample Drive", group="TeleOp")` makes the program accessible from the Driver Station App.
  - This is new to 2016 and you don't have to edit FTCOpModeRegister



Attributes of Your TeleOp Program

- The `init()` method contains code you want/need to run during the initialization phase, such as mapping the motors or servos, setting them to certain positions, et cetera. This only runs once at the beginning when you push the Init button on the Driver Station App.
- The `loop()` method contains code that runs continuously while the program is running.
- `@Override` is an annotation good to have for debugging. It overrides the logs that record system events.

```
//Initializes the code/robot
@Override
public void init() {
```

```
//The code that is run continuously
public void loop() {
```

Creating Motor Objects

- First, before the `init()` method, you need to declare your DcMotor variables.
- Then, inside the `init()` method, map the motors so you can control them.
- Lastly, you may need to reverse one drive motor, left or right, because they might be facing opposite directions.

```
//Creates the motors as objects
DcMotor motorRight;
DcMotor motorLeft;

//Initializes the code/robot
@Override
public void init() {
    //Specifies what motors motorRight and motorLeft control
    motorRight = hardwareMap.dcMotor.get("right");
    motorLeft= hardwareMap.dcMotor.get("left");
    //Reverses the direction of the right motor because the motors are flipped
    motorRight.setDirection(DcMotor.Direction.REVERSE);
}
```

Creating Servo Objects

- First, above the `init()` method, you need to declare your Servo variables. If you were to declare it inside `init()`, only `init()` could use it
- Then, inside the `init()` method, map the servos so you can control them.
- You may need to reverse one servo, left or right, because they might be facing opposite directions (i.e. arm claws).
- Lastly, you may want to start your servo at a specific spot, you do that by typing `ServoName.setPosition(position);`.

```
//Creates your Servos as objects
Servo leftClaw;
Servo rightCLaw;

//A variable used for the Servo Position
double servoPosition;

//Initializes the code/robot
@Override
public void init() {
    //Specifies what servos leftClaw and rightClaw control
    leftClaw = hardwareMap.servo.get("servo2");
    rightCLaw = hardwareMap.servo.get("servo1");
    //Sets leftClaw to be backwards because the motors are flipped
    leftClaw.setDirection(Servo.Direction.REVERSE);

    //Sets the initial position of the claws
    leftClaw.setPosition(0.25);
    rightCLaw.setPosition(0.25);
}
```

Reading From the Gamepads for Motors

- Create a float value for the left and right sticks inside the `loop()` method. I am using only the Y-values because my system is a tank drive. You need to make sure the values are negative because the joystick values are backwards by default (up is negative, down is positive).
- Use these variables to set the motor power.
- Make sure the values don't go out of the range of -1 to 1.

```
//The code that is run continuously
public void loop() {
    //Creates a variable that is easier to use rather than typing it out
    //Also negates the values because the joysticks are flipped
    float leftStick = -gamepad1.left_stick_y;
    float rightStick = -gamepad1.right_stick_y;
```

```
//Makes sure that the values don't go over 1 and under -1
if (leftStick > 1.0) {
    leftStick = 1;
}
if (leftStick < -1.0) {
    leftStick = -1;
}

if (rightStick > 1.0) {
    rightStick = 1;
}
if (rightStick < -1.0) {
    rightStick = -1;
}
```

Setting the Motor Power

- The lines `motorRight.setPower(rightStick);` and `motorLeft.setPower(leftStick);` set the motor power to the joystick values.

```
//Sets the power to be the joystick values
motorRight.setPower(rightStick);
motorLeft.setPower(leftStick);
```

Reading From the Gamepads for Servos

- Create an `if else` statement that sets a variable called `servoPosition` (created outside the `init()` and `loop()` methods) depending on whether or not a button is pushed.
- In the `if` statement, type `if (gamepad.a == true)`. Inside the `if` statement, change `servoPosition` to the closed position. In the `else` statement, set `servoPosition` to the open position.

```
//Actual code that is run continuously
public void loop() {
    //If the A button is pushed, the value will be set so it closes.
    //If the A button is not pushed, the value will be set so it opens.
    if (gamepad1.a == true) {
        servoPosition = 0;
    }
    else {
        servoPosition = 0.25;
    }
```

Setting the Servo Position

- After the variable `servoPosition` has been set based on the button value, set the servo position with the command: `ServoName.setPosition(servoPosition);`.

```
//Sets the Position of the Servos to the value above
leftClaw.setPosition(servoPosition);
rightCLaw.setPosition(servoPosition);
```

Displaying Values to your Phone

- The command `telemetry.addData(“name”, value);` displays information on the Driver Station App.
- This is useful for displaying motor values, servo positions, joystick values, et cetera.
- Make sure the command `telemetry.update();` is present so that your values update on the Driver Station App.

```
//Outputs the motor values to the Drive Station
telemetry.addData("Left Stick", leftStick);
telemetry.addData("Right Stick", rightStick);
telemetry.update();
```

FTC Kickoff Programming Presentation

2.0 - Introduction to FTC Programming

Programming Workshop 2.0 Overview

Programming Workshop 2.0

- Programming for the Autonomous Period
- Vision Processing using VuForia
- Using Gyros for navigation
- Using Encoders for navigation
- Using Other sensors.

Major updates to the 2016-2017 FTC Control System SDK

- Robot control system stability greatly improved. For example, if robot looses connectivity with a core module, robot will keep running with modules that are connected.
- Registering program with "FtcOpModeRegister" is obsolete. Use instead @Autonomous and @TelepOp specification immediately above the "public class" statement. Examples are in the autonomous presentation.
- The FTC Control System SDK comes with open source code.
- API 19 && API 23 are required this year.

Major updates to the 2016-2017 FTC Control System SDK

- Turn on your phone speakers! Robot Controller Apps have audio sound cues to indicate control system status.
- Vuforia Computer Vision SDK is included in release.
- Support for Continuous Rotation Servos.
- More sensor support (compass sensor, gyros, and IMUs)
- "Fix" mechanism added to FTC Robot Controller App making it easier to swap out core module devices.
- For a full list of updates see: https://github.com/ftctechnh/ftc_app

Programming for Autonomous Modes

- Using `LinearOpMode` class
- Making motors move
- Driving by time

`LinearOpMode` Class

- Extending the `OpMode` class will run the robot iteratively, meaning it is looped until stopped via the driver station.

```
//This makes the program known to the Driver Station
//and declares it as a TeleOp Program
@TeleOp(name="Sample Drive", group="TeleOp")
public class SampleDrive extends OpMode {
```

- Extending `LinearOpMode` will have the class only run through once.

```
//Annotation for registering the class.
@Autonomous(name = "FTC Linear Demonstration", group = "Autonomous OpMode")
public class LinearDemonstration extends LinearOpMode{
```

Adding the Annotation

- Use either the `@TeleOp` or `@Autonomous` annotation for adding the class to the OpModes.
  - Only difference is categorization in the driver station app.
- Annotation goes above class declaration.
- Independent from `extends`.

```
import com.qualcomm.robotcore.eventloop.opmode.Autonomous;
//Annotation for registering the class.
@Autonomous(name = "FTC Linear Demonstration", group = "Autonomous OpMode")
```

Implementing `runOpMode()`

- Create a new method called:

  `public void runOpMode() throws InterruptedException { }`.

- All code to run goes inside the method, between the brackets.

```
@Override
public void runOpMode() throws InterruptedException{
```

`runOpMode()` Structure

- Similar to how TeleOp works.
- `init()` method is not available, instantiate variables like `DcMotor` inside `runOpMode()` before `waitForStart()` is called.
- Use `waitForStart()` to pause robot until start is pressed.
- After, you can run motors and get sensory data.

Moving Motors Autonomously

- Create a `DcMotor` variable.
- Instantiate it inside the `runOpMode()` method.

```
@Override
public void runOpMode() throws InterruptedException{
    leftMotor = hardwareMap.dcMotor.get("left_motor");
    rightMotor = hardwareMap.dcMotor.get("right_motor");

    waitForStart();
}
```

Moving Motors Autonomously

- After instantiating all variables, call `waitForStart()`.
  - This pauses the program until the start button is pressed. It will automatically continue to the next step afterwards.

```
@Override
public void runOpMode() throws InterruptedException{
    leftMotor = hardwareMap.dcMotor.get("left_motor");
    rightMotor = hardwareMap.dcMotor.get("right_motor");

    waitForStart();
}
```

Moving Motors Autonomously

- Move the motors by calling `dcMotorVar.setPower(double power)`.
- The motors will move at that power *indefinitely*. `dcMotorVar.setPower(0);` must be called to stop it.

```
//Run the motors at full speed for two seconds and then stop.
leftMotor.setPower(1.0d);
rightMotor.setPower(1.0d);
sleep(2000);
leftMotor.setPower(0);
rightMotor.setPower(0);
```

Driving by Time

- Without a time delay, it will proceed to the next step prematurely.
- `LinearOpMode` class provides a fully implemented `sleep(long time)` method.
  - Calling `sleep(1000)` pauses the program for 1000 milliseconds, or 1 second. All robot operations are disabled during this time. Motors will move with the last power they received.

Driving by Time

```
if(sensor.getHeading() > 90){
    sleep(1000);
}
else{
    sleep(2000);
}
```

Project Vuforia

Note: This presentation only provides basic concepts of Vuforia. Additional information may be found online.

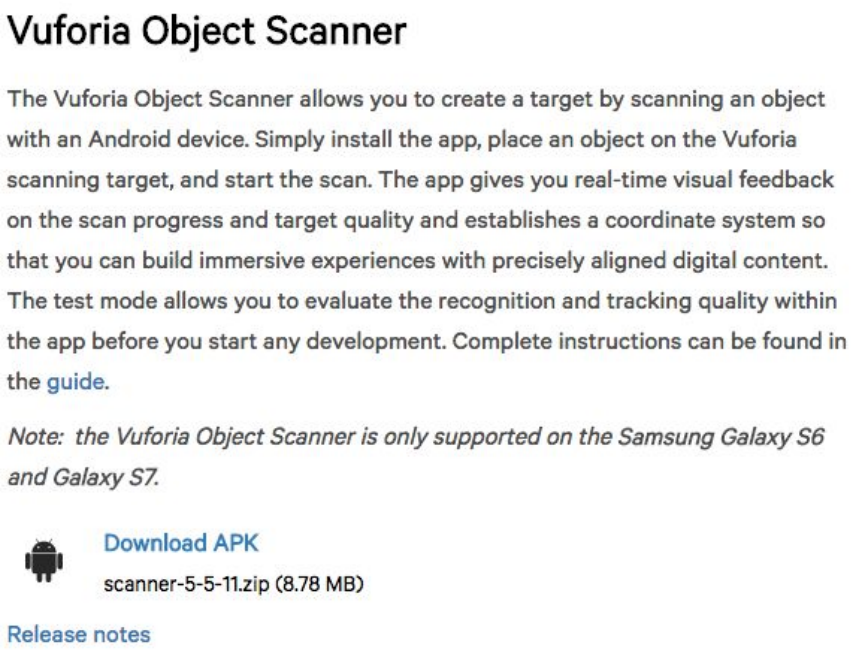
- Vuforia overview
- Vuforia abilities
- Resources

What is Project Vuforia?

- Project Vuforia is an object detection app available for phones.
- It will allow you to scan objects on the playing field and receive sensory information, such as position and location.
- Remains to be determined how robust the software is inside a robot application.

Preparing a Scan

- Download the scanner app from Vuforia's website and use ADB to install it onto a phone.
  - Available at team2486.org/vuforia



Vuforia Object Scanner

The Vuforia Object Scanner allows you to create a target by scanning an object with an Android device. Simply install the app, place an object on the Vuforia scanning target, and start the scan. The app gives you real-time visual feedback on the scan progress and target quality and establishes a coordinate system so that you can build immersive experiences with precisely aligned digital content. The test mode allows you to evaluate the recognition and tracking quality within the app before you start any development. Complete instructions can be found in the guide.

Note: the Vuforia Object Scanner is only supported on the Samsung Galaxy S6 and Galaxy S7.

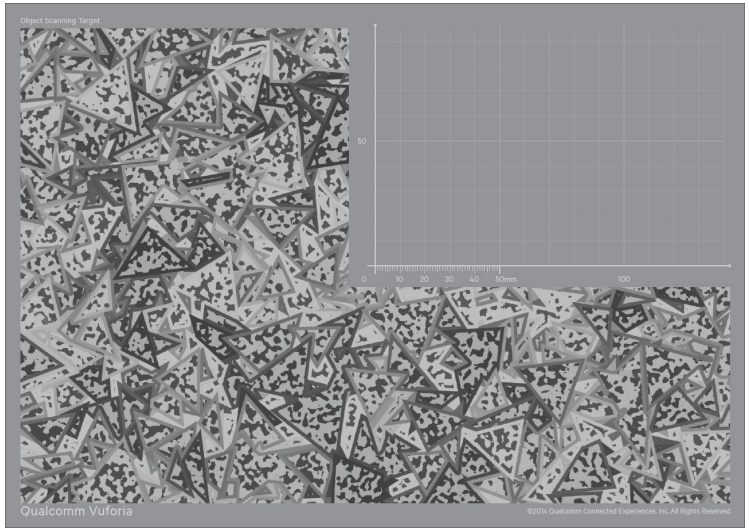
Download APK

scanner-5-5-11.zip (8.78 MB)

Release notes

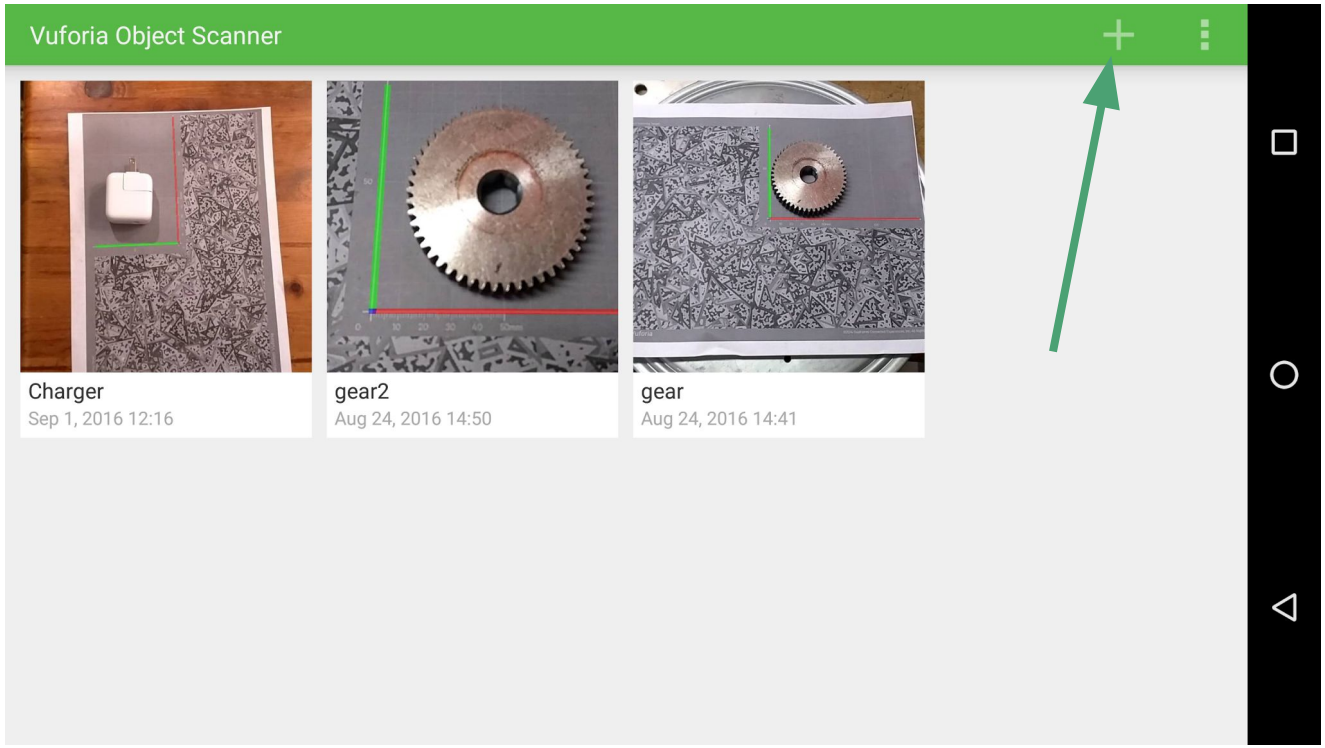
Preparing a Scan

- Download the Object Scanning Target from the Vuforia website and print it.
  - Make sure not to modify the scale of the paper!
  - Available at team2486.org/vuforia



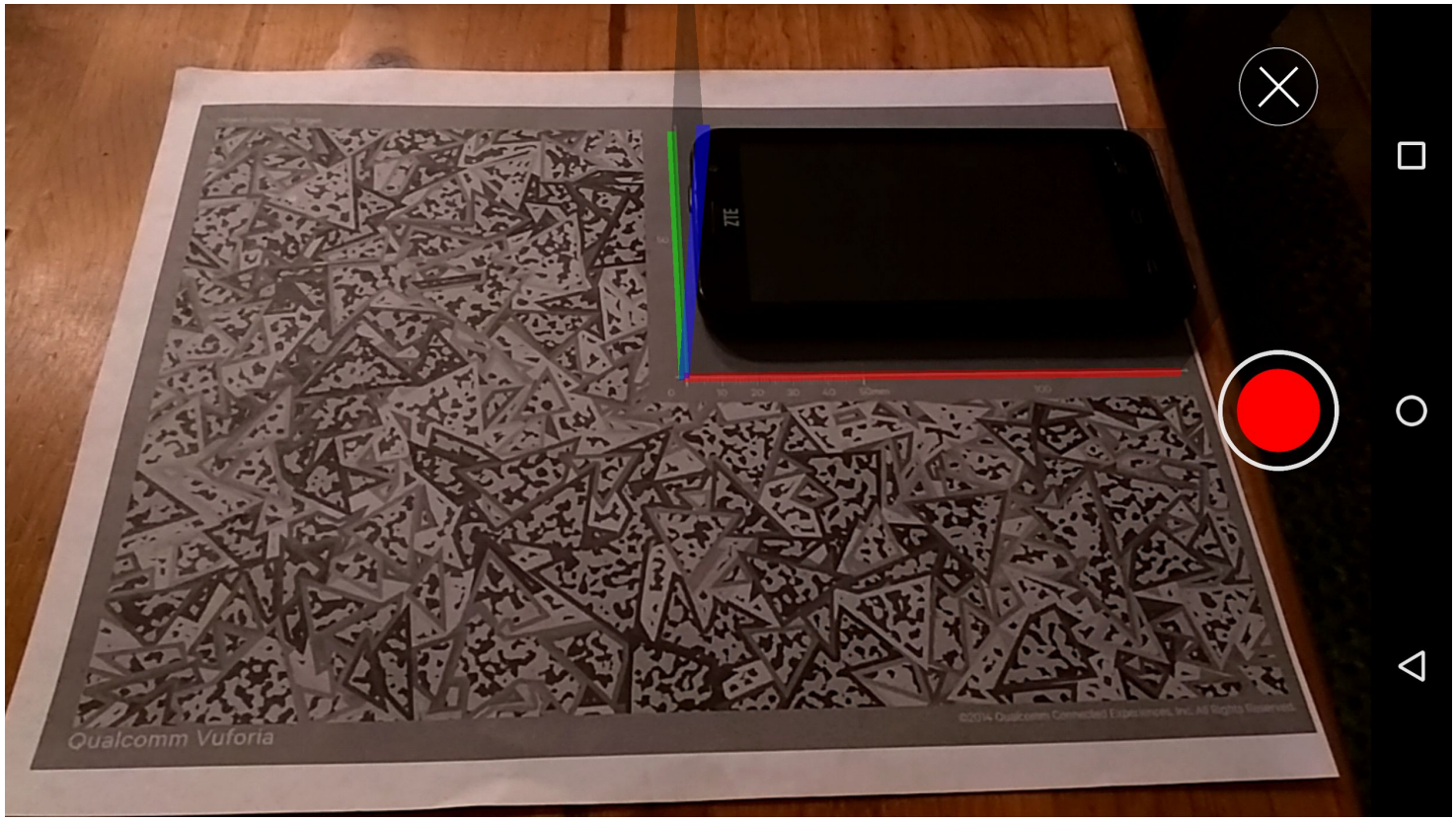
Scanning an Object

- Open the "Scanner" app and create a new model.



Scanning an Object

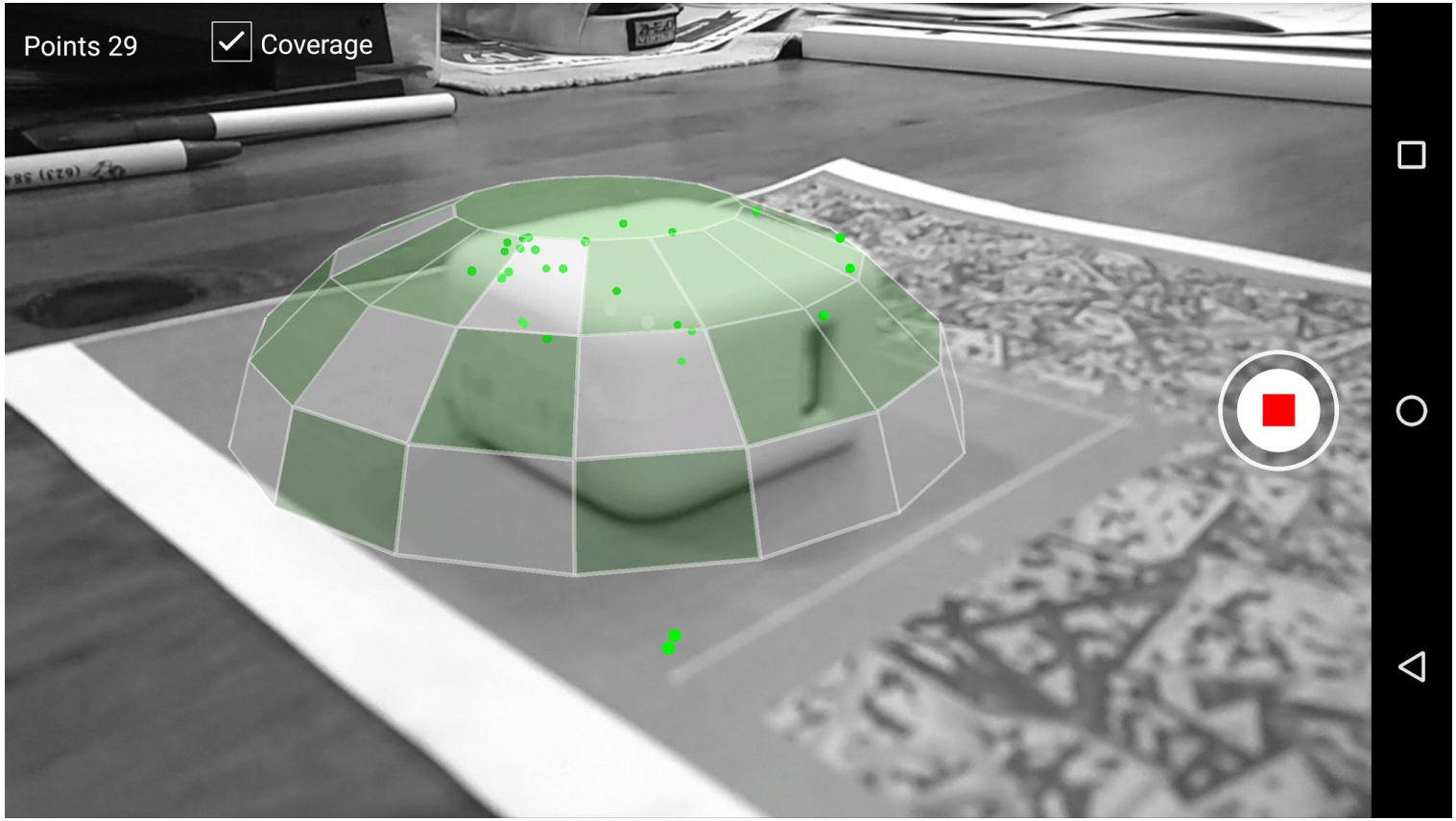
- Place the object into the paper, and align it with the view that appears on-screen. Press the red button to start.



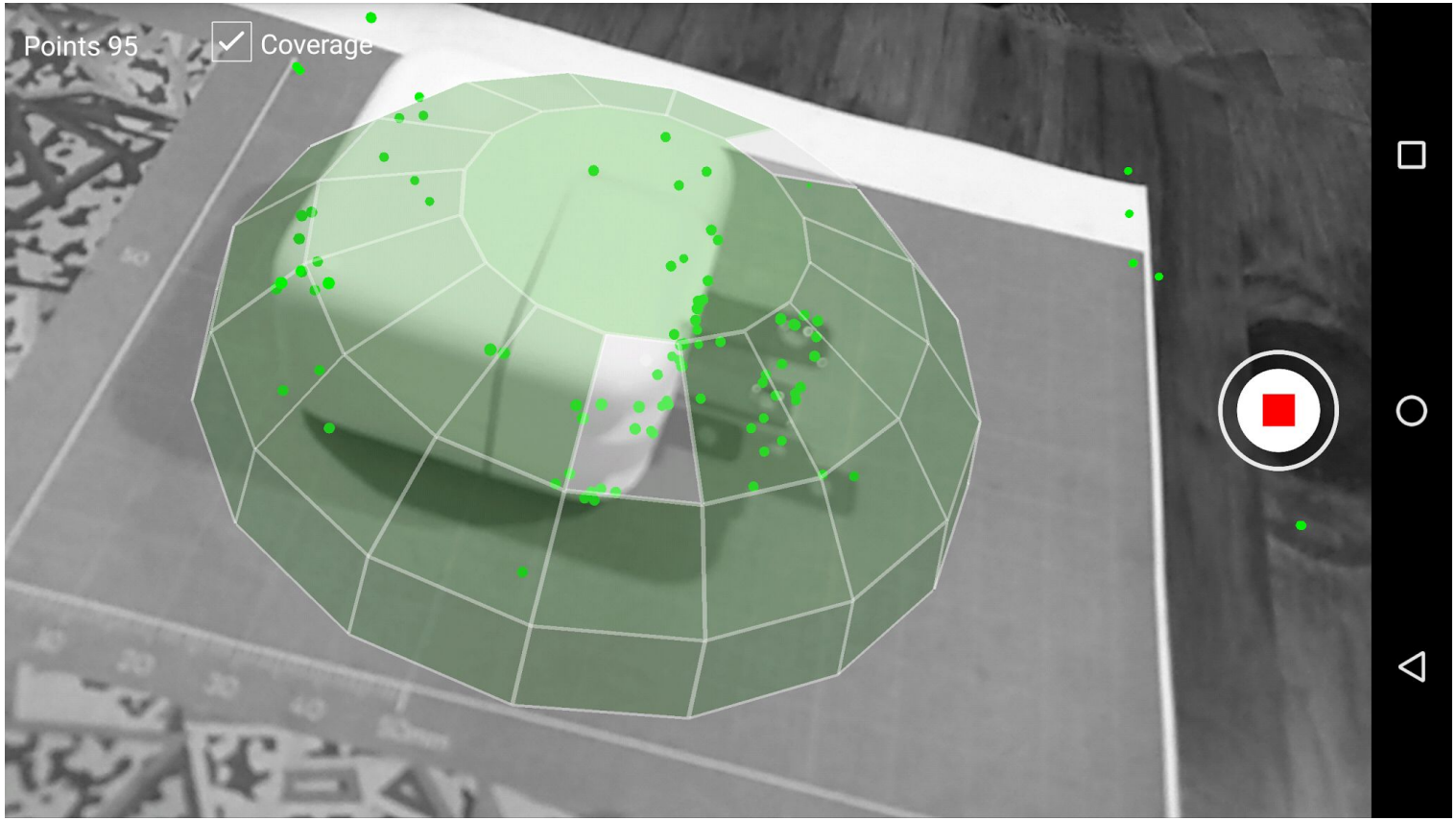
Scanning an Object

- Orient the phone to be facing the panels on the half-sphere that appears.
  - This adds points on-screen that shows the model.
  - Try to keep a high contrast between the paper and the background. Otherwise, it will identify points for the model from the background.

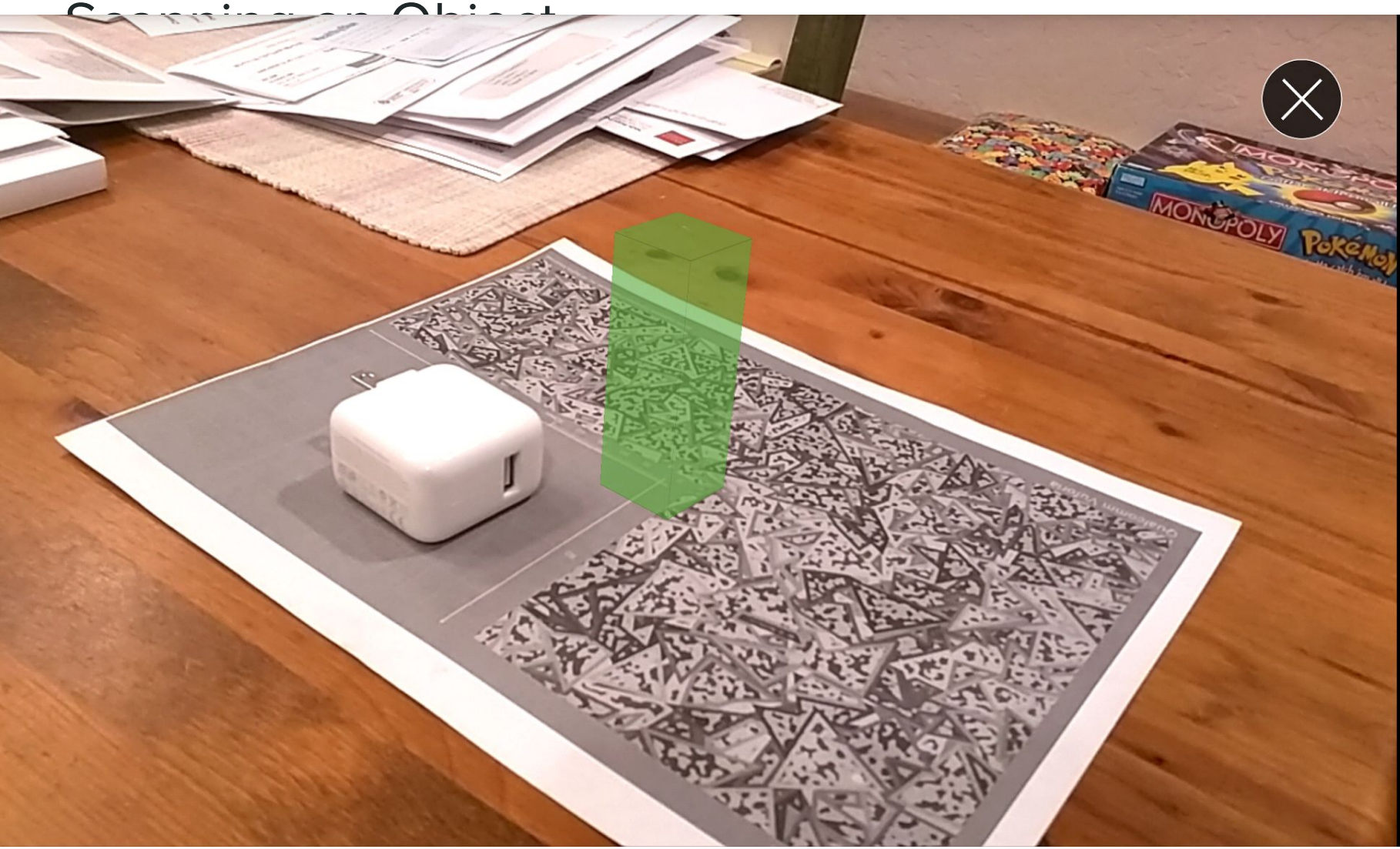
Scanning an Object



Scanning an Object



Scanning an Object



Tips and Tricks for a Good Scan

- The more complex an object is, the better Vuforia will recognize it.
- Attempt to fully complete all green rectangles on the sphere.
  - Sometimes, there will be one panel left that is difficult to get, and testing the model will determine whether or not it is needed.

Implementing a Custom Scan into a Program

- Unfortunately, we did not have time to fully test Vuforia, and do not have information on this.
- Video tutorials on how to use and implement Vuforia are courtesy of team 3491 FIXIT, linked at team2486.org/vuforia

Chips and Stones Data Set

- The SDK comes with example OpModes, in the external.samples folder.
- The Vuforia example included in the FTC SDK uses a data model (a scan) called *chips and stones*.
  - The chips and stones data set identify the beacons and the color of them, to which you can program the robot to respond accordingly.

Chips and Stones Data Set

```
VuforiaTrackables stonesAndChips = this.vuforia.loadTrackablesFromAsset("StonesAndChips");
VuforiaTrackable redTarget = stonesAndChips.get(0);
redTarget.setName("RedTarget");  // Stones

VuforiaTrackable blueTarget  = stonesAndChips.get(1);
blueTarget.setName("BlueTarget");  // Chips

/** For convenience, gather together all the trackable objects in one easily-iterable collection */
List<VuforiaTrackable> allTrackables = new ArrayList<~>();
allTrackables.addAll(stonesAndChips);
```

This is a very small snippet of code, and does not reflect the total amount of code required.

Chips and Stones Data Set

- The phone then creates a virtual field to help itself navigate, and initializes the beacons on the virtual field.

```
OpenGLMatrix redTargetLocationOnField = OpenGLMatrix
        /* Then we translate the target off to the RED WALL. Our translation here
        is a negative translation in X.*/
        .translation(-mmFTCFieldWidth/2, 0, 0)
        .multiplied(Orientation.getRotationMatrix(
                /* First, in the fixed (field) coordinate system, we rotate 90deg in X, then 90 in Z */
                AxesReference.EXTRINSIC, AxesOrder.XZX,
                AngleUnit.DEGREES, 90, 90, 0));
redTarget.setLocation(redTargetLocationOnField);
RobotLog.ii(TAG, "Red Target=%s", format(redTargetLocationOnField));
```

Chips and Stones Data Set

- Lastly, it provides methods available for determining what is in view.

```
while (opModeIsActive()) {

    for (VuforiaTrackable trackable : allTrackables) {
        /**
         * getUpdatedRobotLocation() will return null if no new information is available since
         * the last time that call was made, or if the trackable is not currently visible.
         * getRobotLocation() will return null if the trackable is not currently visible.
         */
        telemetry.addData(trackable.getName(), ((VuforiaTrackableDefaultListener)trackable.getListener()).isVisible() ? "Visible" : "Not Visible");

        OpenGLMatrix robotLocationTransform = ((VuforiaTrackableDefaultListener)trackable.getListener()).getUpdatedRobotLocation();
        if (robotLocationTransform != null) {
            lastLocation = robotLocationTransform;
        }
    }
    /**
     * Provide feedback as to where the robot was last located (if we know).
     */
    if (lastLocation != null) {
        //  RobotLog.vv(TAG, "robot=%s", format(lastLocation));
        telemetry.addData("Pos", format(lastLocation));
    } else {
        telemetry.addData("Pos", "Unknown");
    }
    telemetry.update();
    idle();
}
```

Programming for Encoders

- Basic motor commanding for encoders
- Using encoders to move a mechanism to an exact position
- Using encoders for driving by distance.

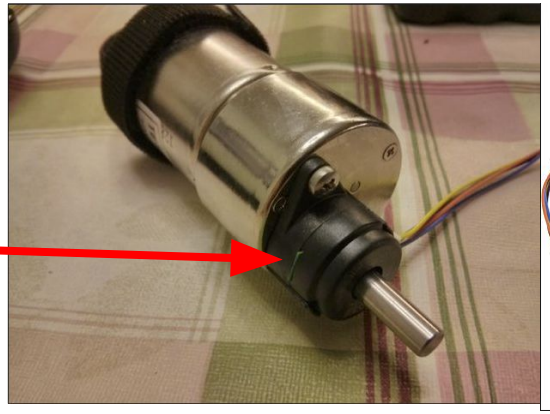
Encoders

- Accurately measure the distance traveled by a drive system
- Move a mechanism to a precise and reproducible position.
- Encoders are mounted on an electrical motor providing feedback on the position and speed of the motor shaft.
- Motors can be purchased with built-in encoders -or-
- Encoder kits can be purchased to mount on an existing motor shaft

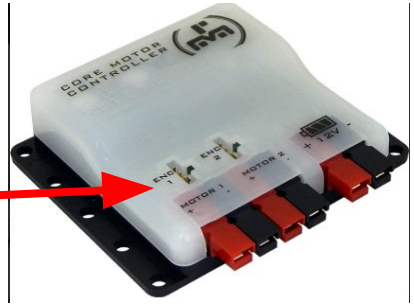
AndyMark motor with built in encoder

Encoder cable plugs in here.

Tetrix motor with add on encoder



Encoder kit mounted on shaft.



Encoder cables plug on top of the motor controller

Encoder Characteristics

- Encoders need to be reset to set current encoder value to 0.
- Encoder values can take on negative or positive values depending on the rotation direction from the starting position of the encoder.
- The number of encoder counts per revolution varies with manufacturer. (AndyMark has 1680 counts/revolution, Tetrix encoder kits have 1440)
- Encoders values will continue to increase or decrease as motor rotates (ie. one revolution = 1680 counts, two revolutions = 3360 counts, etc.)
- Be careful! Occasionally an encoder will skip a count so errors in rotational position can accumulate.
- With RUN_TO_POSITION commanding, encoders will rigidly and forcefully maintain a target position. Thus encoders can be used much like a servo for maintaining positions of heavier mechanisms.

DC Motor Command Summary

```
import com.qualcomm.robotcore.hardware.DcMotor;

DcMotor myMotor;
myMotor = hardwareMap.dcMotor.get("motor name");
/*
* Specify the DC motor operating mode, mode values:
* DcMotor.RunMode.STOP_AND_RESET_ENCODER
* DcMotor.RunMode.RUN_TO_POSITION
* DcMotor.RunMode.RUN_USING_ENCODER
* DcMotor.RunMode.RUN_WITHOUT_ENCODER  (Default mode)
*/
myMotor.setMode(DcMotor.RunMode.mode);

/* set target position for mode RUN_TO_POSITION */
myMotor.setTargetPosition(int encoderValue);

/* set power value for motor range -1.0 to 1.0 */
myMotor.setPower(double powerValue);

/* set motor velocity in encoder counts per second */
myMotor.setMaxSpeed(int countsPerSecond);

/* true if motor is still moving to the target position */
boolean advancing = myMotor.isBusy();
```

DC Motor Command Summary (continued)

```
/* Obtain current DC motor command mode */
int runMode = myMotor.getMode();

/* Obtain current DC motor encoder position */
int position = myMotor.getCurrentPosition();

/* Obtain the DC Motor target position */
int target = myMotor.getTargetPosition();

/* Obtain the current power value */
double power = myMotor.getPower();

/* Obtain current speed of motor (encoder counts per second)*/
int speed = myMotor.getMaxSpeed();
```

Using an DC Motor/Encoder like a Servo

```
@TeleOp(name = "SimpleEncoderExample", group = "TeleOp")
public class SimpleEncoderExample extends OpMode {

    DcMotor motorRight;
    int encoderPosition;
    int encoderPosition1 = 0;   // 1st encoder position
    int encoderPosition2 = 2000;// 2nd encoder position
    double motorPower = 0.5;    // motor power level.

    @Override
    public void init() {
        //initialize the motor
        motorRight = hardwareMap.dcMotor.get("right");
        //reset encoders, encoder position set to 0
        motorRight.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);

        //report the encoder mode to telemetry output
        telemetry.addData("Encoder Mode:",motorRight.getMode().toString());
        telemetry.update();
    }
```

Using an DC Motor/Encoder like a Servo (continued)

```
@Override
public void loop() {
    //set motor mode to RUN_TO_POSITION if not already set
    if (motorRight.getMode() != DcMotor.RunMode.RUN_TO_POSITION ) {
        motorRight.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        return;
    }
    //if gamepad right bumper pushed move to position 1
    if (gamepad1.right_bumper) {
        encoderPosition = encoderPosition1;
        motorRight.setTargetPosition(encoderPosition);
        motorRight.setPower(motorPower);
    }
    //else move encoder position to position 2
    else {
        encoderPosition = encoderPosition2;
        motorRight.setTargetPosition(encoderPosition);
        motorRight.setPower(motorPower);
    }
    //Report information to telemetry output
    telemetry.addData("Right Bumper",gamepad1.right_bumper);
    telemetry.addData("Encoder Mode",motorRight.getMode().toString());
    telemetry.addData("Target Encoder Position",encoderPosition);
    telemetry.addData("Current Encoder Position",motorRight.getCurrentPosition());
    telemetry.update();
}
```

Using Encoders to Drive by Distance

```
@Autonomous(name="DriveByEncoder", group="Autonomous")
public class DriveByEncoder extends LinearOpMode {

    static final double COUNTS_PER_MOTOR_REV = 1680;    //Encoder ticks for AndyMark-3103
    static final double DRIVE_GEAR_REDUCTION = 1.0;     //This is < 1.0 if geared UP
    static final double WHEEL_DIAMETER_INCHES = 4.0;    //For figuring circumference
    static final double COUNTS_PER_INCH = (COUNTS_PER_MOTOR_REV * DRIVE_GEAR_REDUCTION) /
            (WHEEL_DIAMETER_INCHES * 3.1415);
    static final double DRIVE_SPEED = 0.6;
    static final double TURN_SPEED = 0.5;


    DcMotor motorRight;
    DcMotor motorLeft;

    private ElapsedTime runtime = new ElapsedTime();
```

Using Encoders to Drive by Distance (continued)

```
@Override
public void runOpMode() throws InterruptedException {

    motorRight = hardwareMap.dcMotor.get("right");
    motorLeft = hardwareMap.dcMotor.get("left");
    motorRight.setDirection(DcMotor.Direction.REVERSE);

    motorLeft.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
    motorRight.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
    idle();

    motorLeft.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
    motorRight.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
    idle();

    //Wait for the game to start (driver presses PLAY)
    waitForStart();

    //Step through each leg of the path,
    //Note: Reverse movement is obtained by setting a negative distance (not speed)
    encoderDrive(DRIVE_SPEED,  48.0,  48.0, 5.0);  // S1: Forward 48 Inches with 5 Sec timeout
    encoderDrive(TURN_SPEED,   12.0, -12.0, 4.0);  // S2: Turn Right 12 Inches with 4 Sec timeout
    encoderDrive(DRIVE_SPEED, -24.0, -24.0, 4.0);  // S3: Reverse 24 Inches with 4 Sec timeout

    telemetry.addData("Path", "Complete");
    telemetry.update();
}
```

```java
public void encoderDrive(double speed, double leftInches,
                         double rightInches, double timeoutS) throws InterruptedException {
    int newLeftTarget;
    int newRightTarget;

    // Ensure that the opmode is still active
    if (opModeIsActive()) {

        // Determine new target position, and pass to motor controller
        newLeftTarget = motorLeft.getCurrentPosition() + (int)(leftInches * COUNTS_PER_INCH);
        newRightTarget = motorRight.getCurrentPosition() + (int)(rightInches * COUNTS_PER_INCH);
        motorLeft.setTargetPosition(newLeftTarget);
        motorRight.setTargetPosition(newRightTarget);

        // Turn On RUN_TO_POSITION
        motorLeft.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        motorRight.setMode(DcMotor.RunMode.RUN_TO_POSITION);

        // reset the timeout time and start motion.
        runtime.reset();
        motorLeft.setPower(Math.abs(speed));
        motorRight.setPower(Math.abs(speed));

        // keep looping while we are still active, and there is time left, and both motors are running.
        while (opModeIsActive() &&
                (runtime.seconds() < timeoutS) &&
                (motorLeft.isBusy() && motorRight.isBusy())) {
            // Display it for the driver.
            telemetry.addData("Path1",  "Running to %7d :%7d", newLeftTarget,  newRightTarget);
            telemetry.addData("Path2",  "Running at %7d :%7d",
                    motorLeft.getCurrentPosition(),
                    motorRight.getCurrentPosition());
            telemetry.update();
            idle();
        }
        // Stop all motion;
        motorLeft.setPower(0);
        motorRight.setPower(0);
        // Return to normal motor operating mode
        motorLeft.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        motorRight.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        sleep(250);   // optional pause after each move
    }
}
```

Using Gyroscopes

- What is a gyroscope?
- Modern robotics integrating gyroscope
- Example code

What is a gyroscope?

- Gyroscopes can be used to determine heading of robot in degrees.
- This data can be used to ensure your robot travels on a precise path, make exact turns, and even course correct.
- Data provided includes: Rate of change in yaw, pitch, and roll.

Modern robotics integrating gyroscope

- Heading data is processed on the gyroscope itself, so the load on the control system is significantly reduced.
- Note: only heading data is processed onboard.
- To ensure proper operation, leave gyroscope on for about thirty seconds before calibrating, this allows it to warm up properly.
- Always calibrate before operation for accurate values. This takes about 2 seconds.
- When calibrating or warming up, robot must be absolutly still.
- Costs only $32.95
- Can be purchased at http://www.modernroboticsinc.com/integrating-3-axis-gyro

Code example

Below is some example code for getting the heading value

```
GyroSensor gyroSensor;
@Override
public void init() {
    gyroSensor = hardwareMap.gyroSensor.get("gyro");
    //This is necessary for proper function
    gyroSensor.calibrate();
}
@Override
public void loop() {
    //Just some debug information
    telemetry.addData("Status: ", gyroSensor.status());
    //If still calibrating, return from function
    if(gyroSensor.isCalibrating()) return;
    //If finished calibrating, report heading
    else telemetry.addData("Heading: ", gyroSensor.getHeading());
}
```

Code example

Below is some pseudocode used for course correction

Error = Target - Current
Correction = Gain * Error *//Gain is determined from testing*
LeftPower = Power + Correction
RigtPower = Power - Correction
RunLeftMotor (LeftPower)
RinRightMotor (RightPower)

Questions?