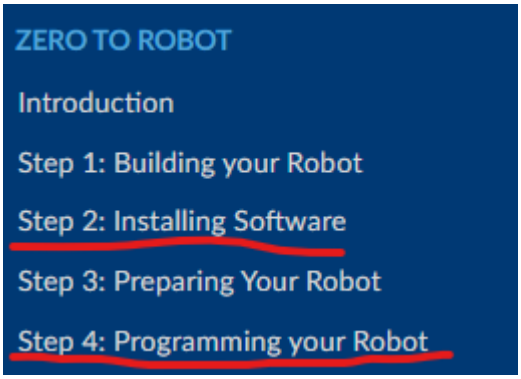
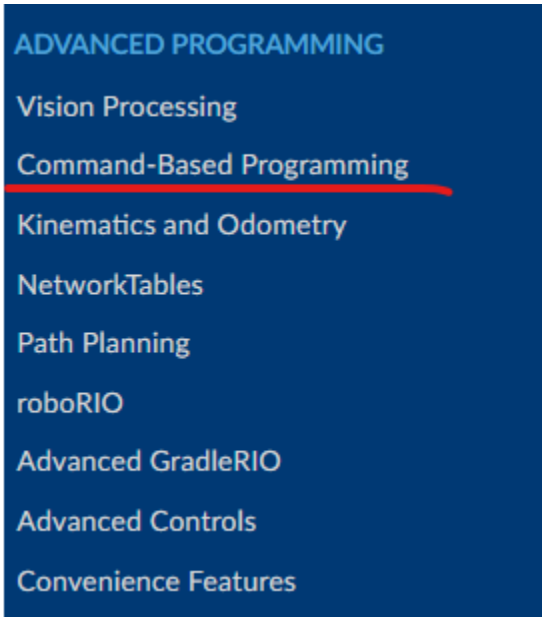
Programming Student Guide

Every year, there is a protocol to update firmware and drivers. So, we decided to have this written for students so that they know what to do and how to do it rather than have the mentors do it.

Reinstall computer software

As always, when in doubt, visit the FIRST website. Easiest way to do so is to look up “frc control system”. However, the whole point to this guide is that their website isn’t always straight forward. When it’s first opened, there are two sections that seem important and are underlined in red:





In reality, it’s best if we all read everything, but these are two good places to start.

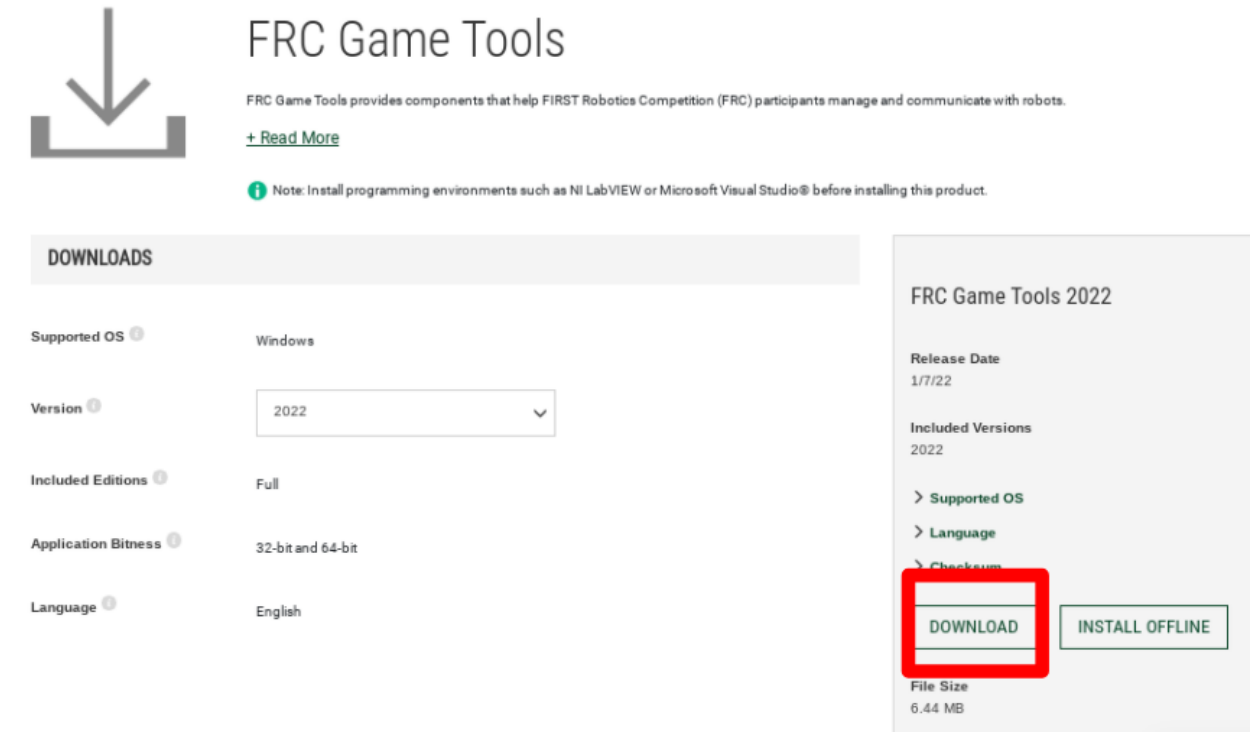
First, to install the software, start at step 2.

Step 2: Installing Software

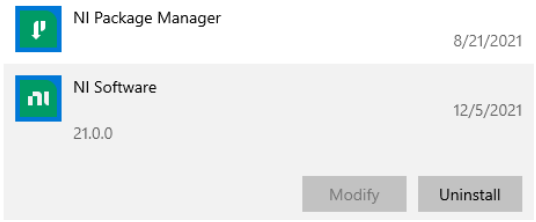
An overview of the available control system software can be found here.

- Offline Installation Preparation
- Installing LabVIEW for FRC (LabVIEW only)
- Installing the FRC Game Tools
- WPILib Installation Guide
- Next Steps

Under Offline Installation Preparation, there's a link for FRC Game tools.

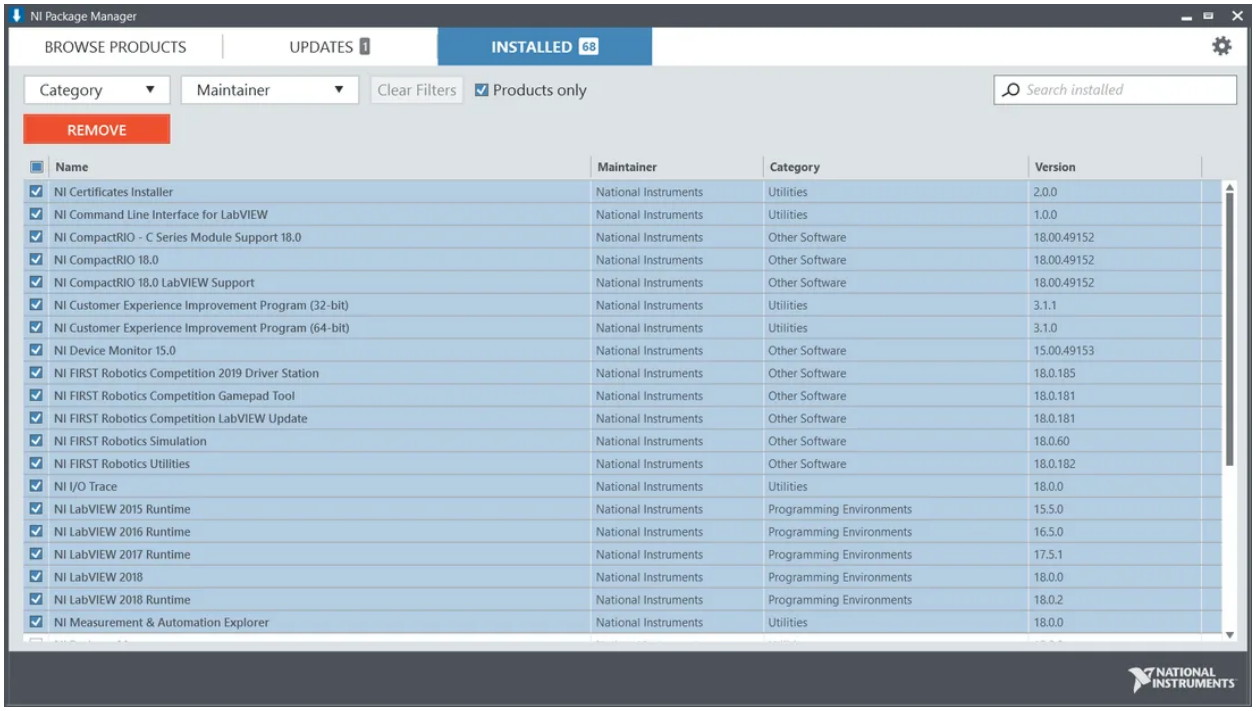


Delete the previous years then Download and install.
To delete the previous years, either look on the website, or do the following. Click the windows key, type add and press enter on **"add or remove programs."** Then scroll down to the:

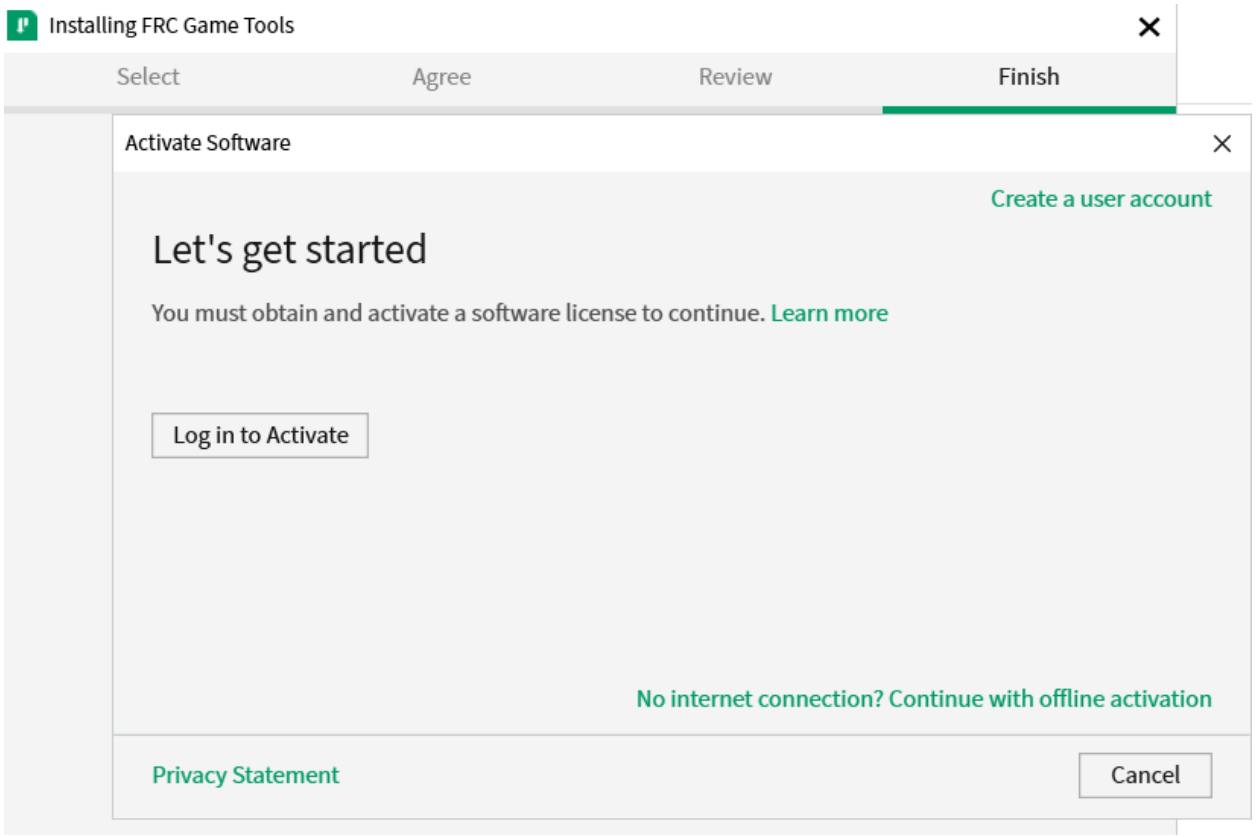


Note: uninstall the software.

Then remove everything. And install the new stuff.



If the license popped up, like so:

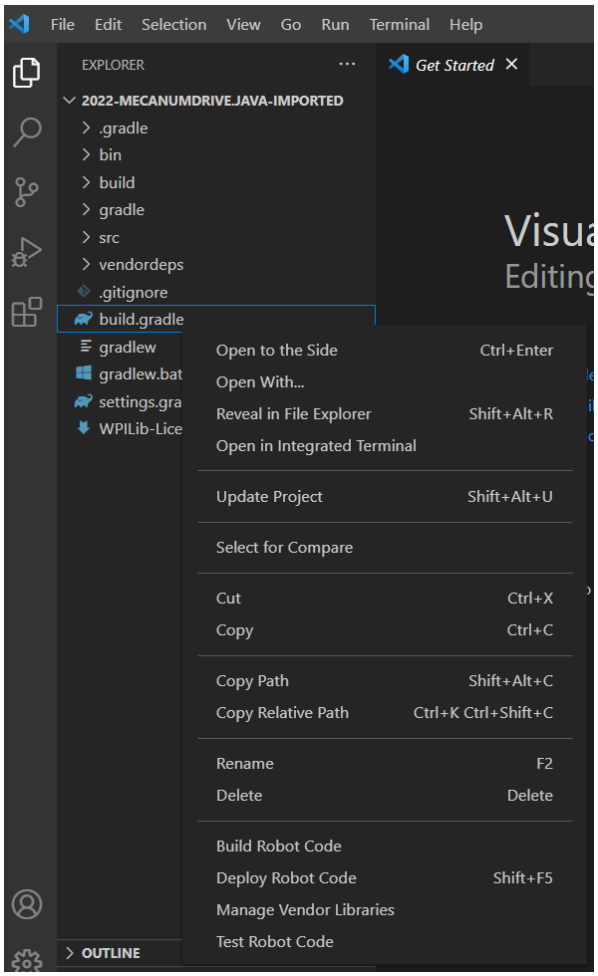


Take a look at the passwords spreadsheet to log in. If it doesn't work, click cancel and see if it installs regardless.

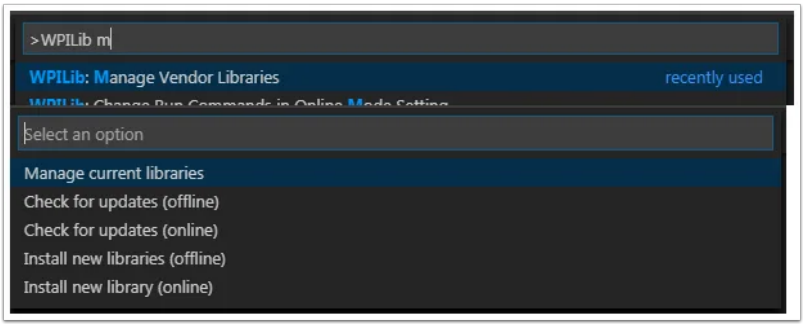
Do the same for the FRC Radio Configuration Utility and the FRC Radio Configuration Utility Israel Version.
Note: we do **not** need LabVIEW Teams because we don't use it to program.

Next, the Java/C++ WPILib Installer(we will probably use it for Java). However, chances are, it's already installed. If it is installed and works, then there's no need to install it. To install it, follow the steps on the FIRST website. This program is the one where the code is written and run. It isn't the exact same as the regular visual studio code.

Lastly, if the computer doesn't have the CTRE_Phoenix_Framework, then install it. For this year, we got it from this link. This counts as a third party library in the code. So to use it, follow the link or the instructions below this paragraph. If this step isn't done at all, then the **imports won't work**.



After your project is opened, right click on build.gradle and click **Manage Vendor Libraries**. Then click the **Install new libraries (offline).**

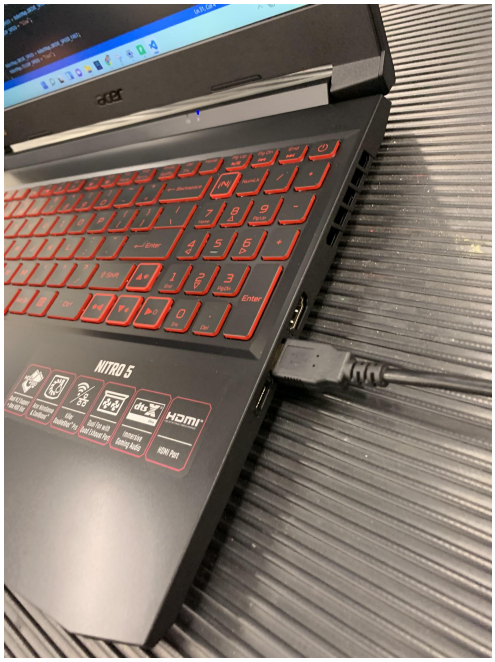


The Firmware of the robot.

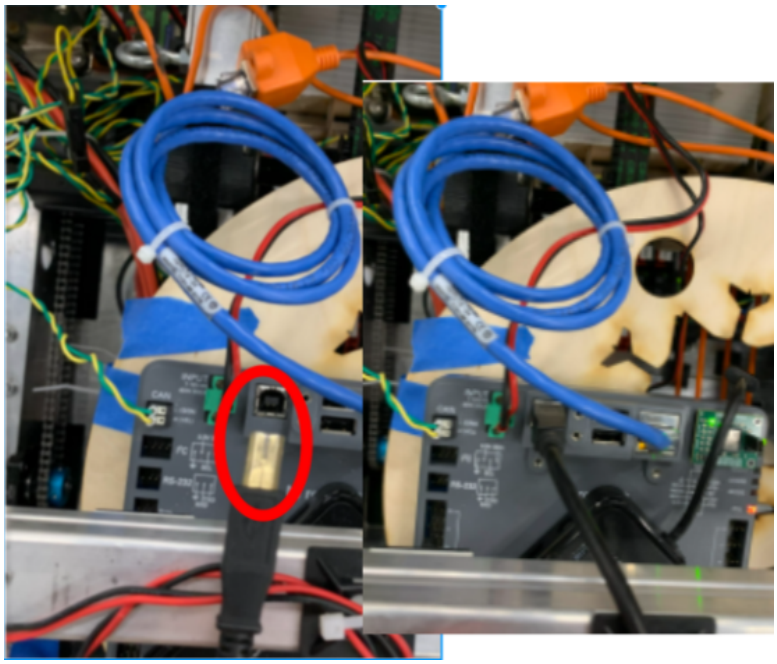
Next steps are to update the firmware of the robot.

Updating the RoboRIO

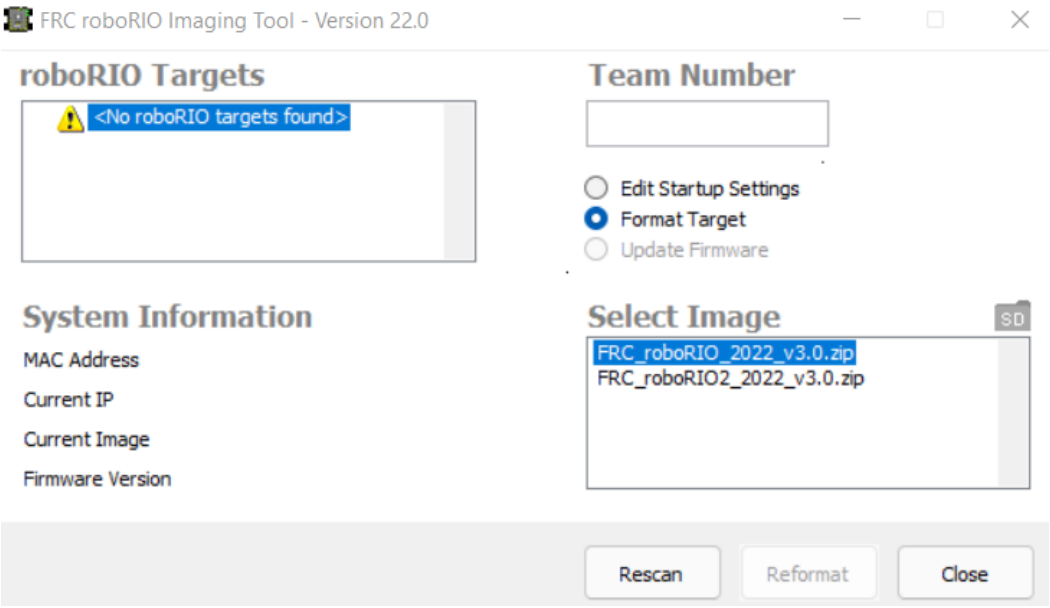
The RoboRio is the small, gray box that functions as the robot's computer. It runs our code. To update this, open the RoboRio imaging tool on the computer, and plug in a USB to the RoboRio like shown (MAKE SURE THE ROBORIO HAS POWER / IS TURNED ON):



(Plugging in.)



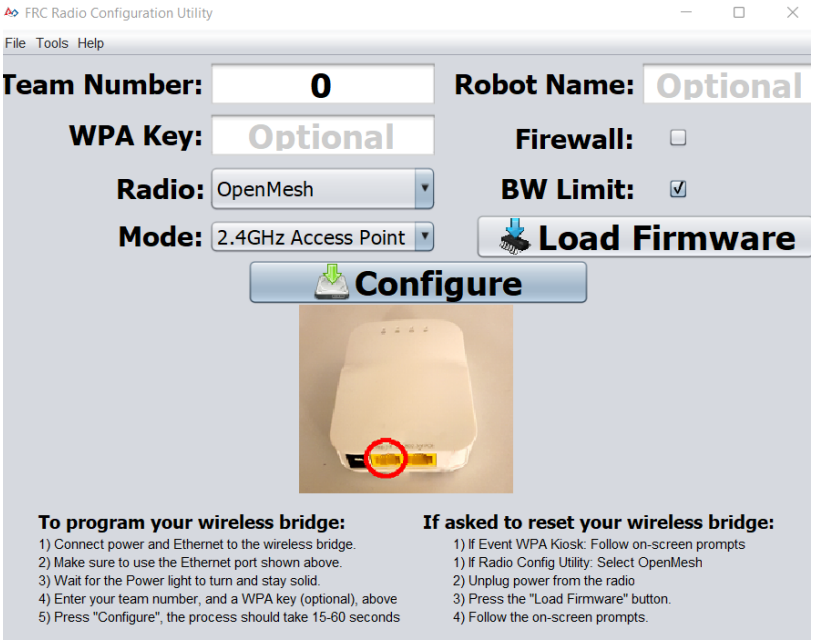
Once you have it plugged in, it should show up on the imaging tool.



Make sure to include the team number 4121 (this applies to other programs too, make sure to always check that it always says 4121). Then, select the image and click “reformat.” This will take a while (like maybe 10 mins). If it fails, try again.

Updating the radio device

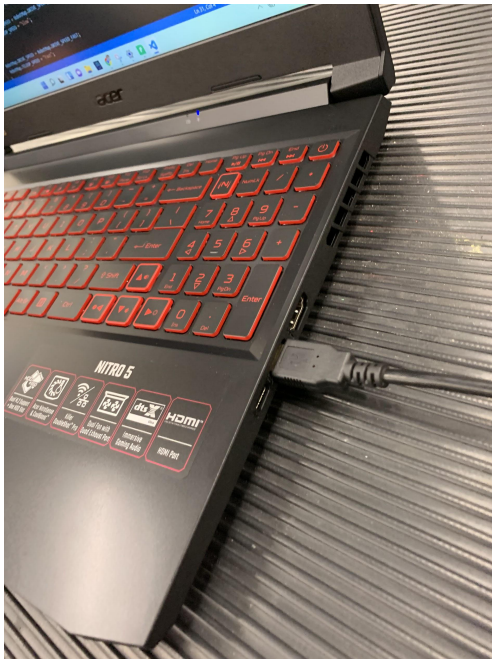
To update the radio device, run the FRC Radio Configuration tool while plugged into the radio with an ethernet cable. It should look like:

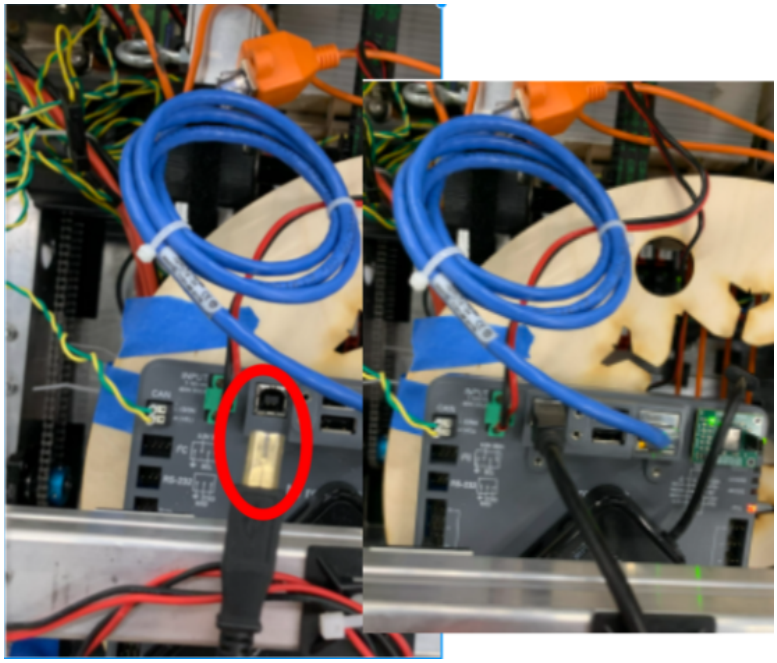


On the config tool, write in the team number (4121) and give the robot a name. It says the name is optional, but you should give it name; it makes everything easier. Once all that's done, click load firmware and wait for it to update. If it fails, try again.

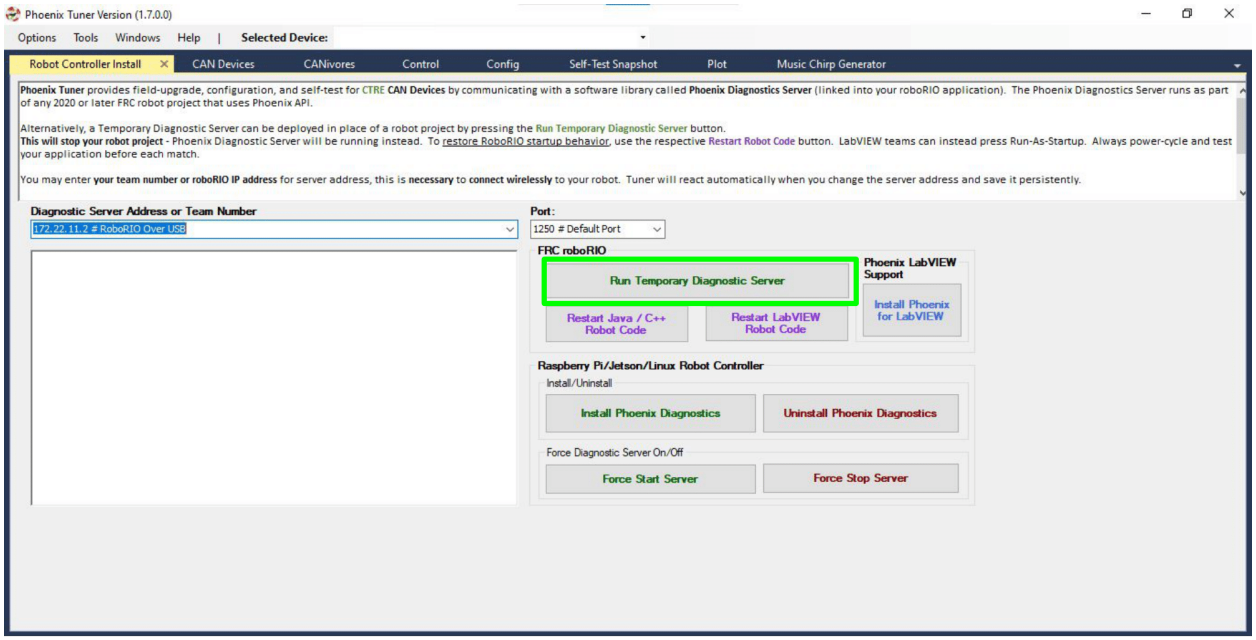
Firmware update

The first step is to open the **Phoenix Tuner** app from the desktop. This app controls the motors. Make sure the computer is plugged into the robot and keep the "FRC control Driver app" opened.

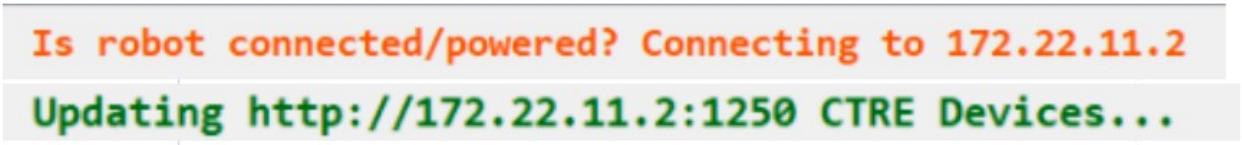




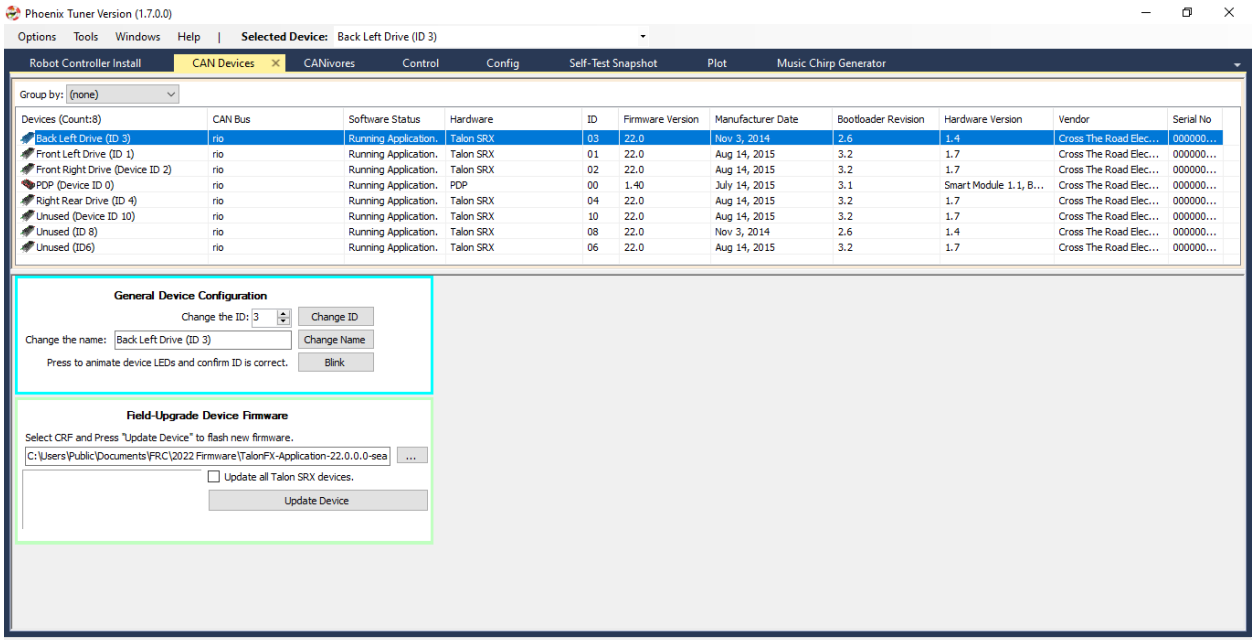
Your screen should look something like the following picture. The first time the robot is run click the green **Run Temporary Diagnostics Server** button. Let it finish.



Looking at the bottom of the screen, you should see if the robot is connected to the computer or not. Generally, when it's connected, the text turns green and it says "OK" See the picture below for examples. (The green is good)



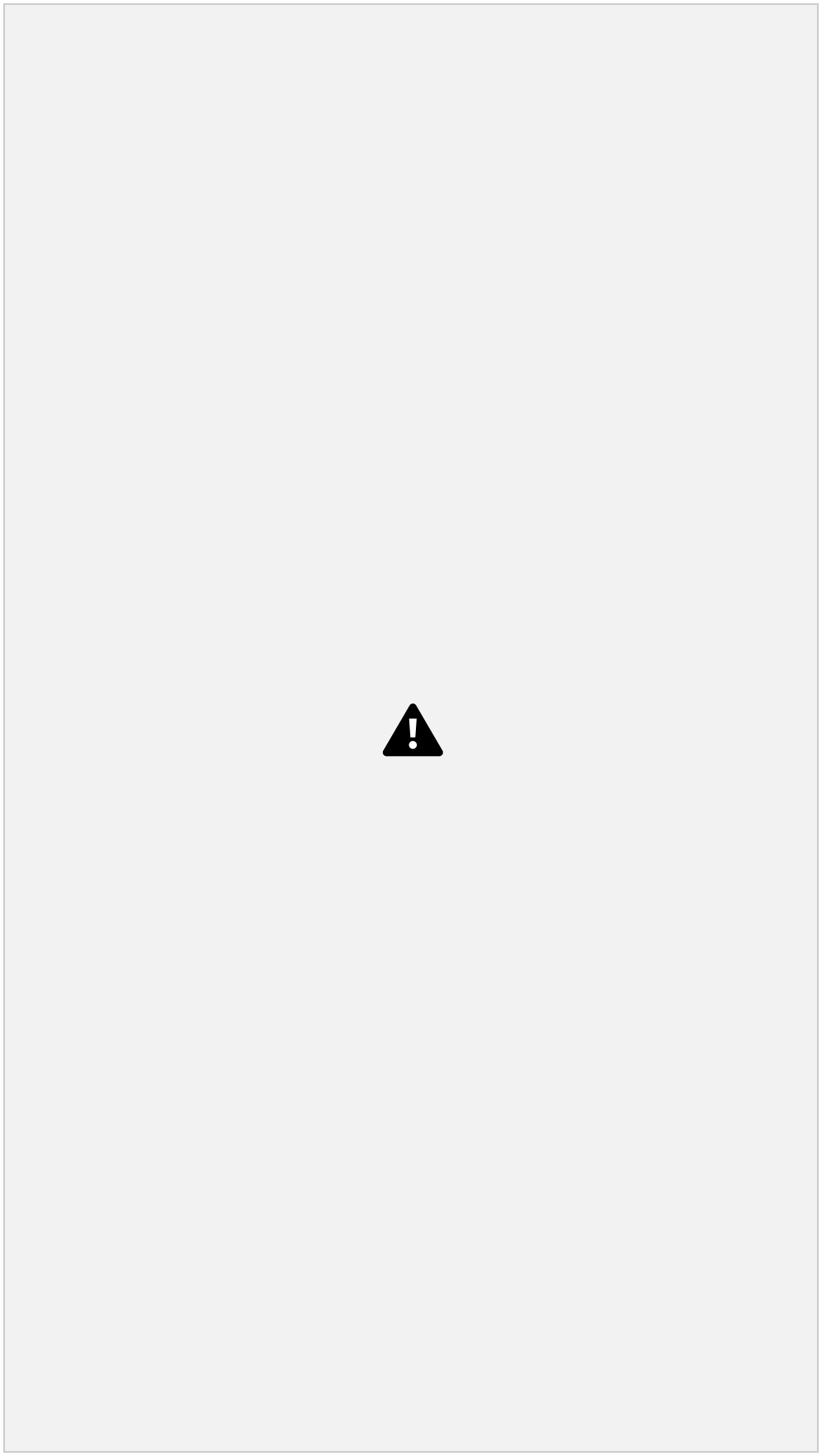
The next tab (such as in the following image) is probably the most important. Here, you can select each of the items in the list. Basically, it's a list of different ports of motors and other things. For now, select one of the motors. You can see that there is an **ID** field and a **Name** field. The ID is crucial because it's what goes into the code. The ID is the code's way of knowing which motor is which. The name is important for not mixing the motors up. Usually you want to put the ID in the name also. You can notice there's a **Blink** Button I will explain this later



For now, you can update the firmware by finding the newest firmware files and clicking **update device.** Make sure that you are selecting on one motor and/or any of the other devices(like PDP).

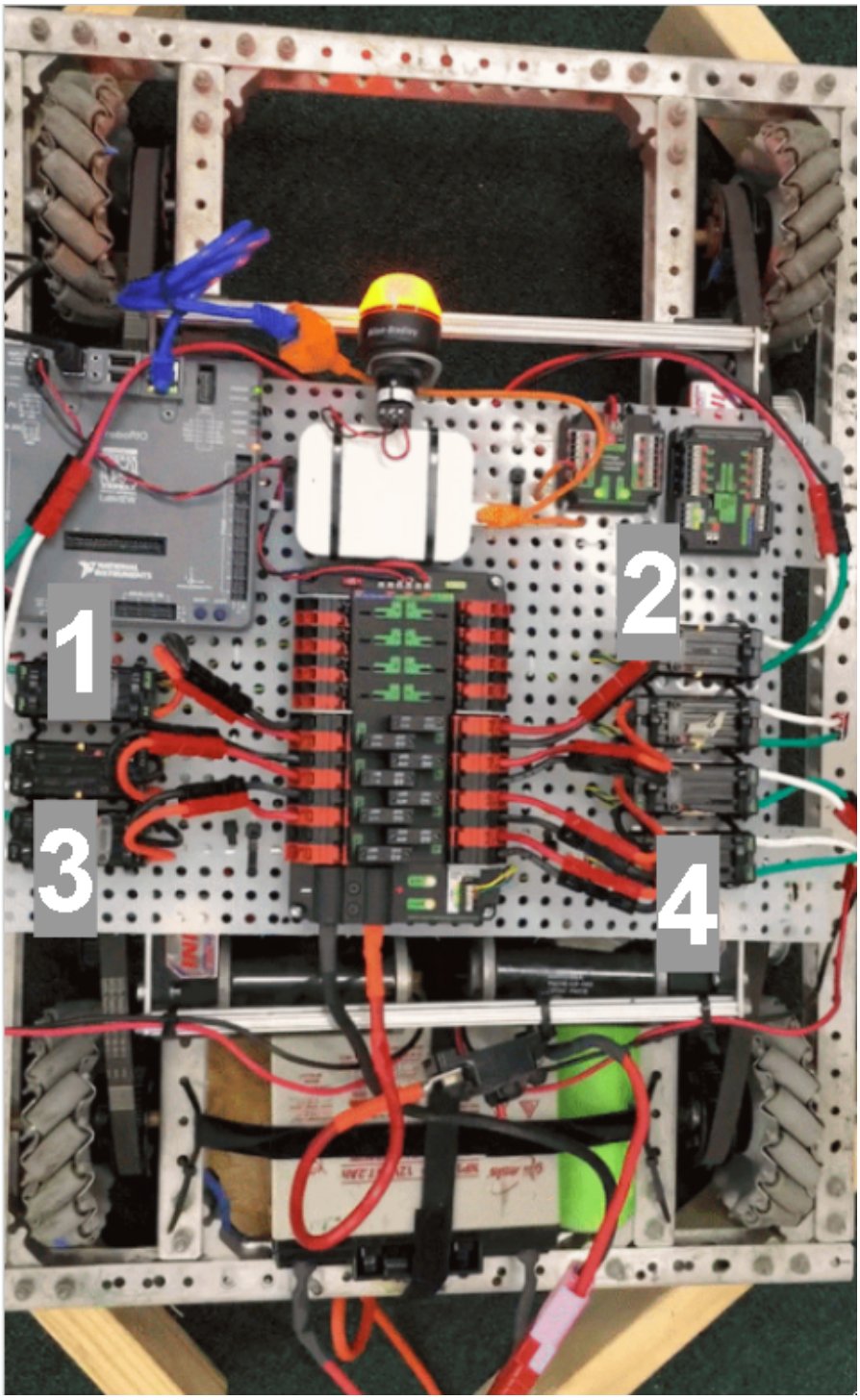
Also, you can manually test each of the motors in the Control tab where you can give it power and see if it turns. However, be careful you don't give it 100% power when the robot isn't mounted correctly and rigidly. Note: when you're on the control tab, look a little bit higher and make sure that you selected the right motor under **Selected Device.**

BLINK. This button is for you to find out which motor is which. For example, you want a physical motor to be ID 1 and named Front_Left. Which physical motor is it? So you click blink and watch for the motors to see if they blink. When using the Talons RX, the black bricks blink. However, when using the Talons SRX, the motor itself blinks because it skips going through the black bricks (IDK what they're actually called. Refer to the photo/gif)



In the GIF above, can you see which one is blinking? That's right! It's the one in the bottom left of the image, that one black brick. It blinks more rapidly than the other ones.

Lastly, ask around to make sure you understand how to ID and name the motors. In the year 2022, we had the system like this because it made more sense. Driving motors(motors linked directly to the wheels) were ID's 1,2,3, and 4. And it made most sense to do them like this.



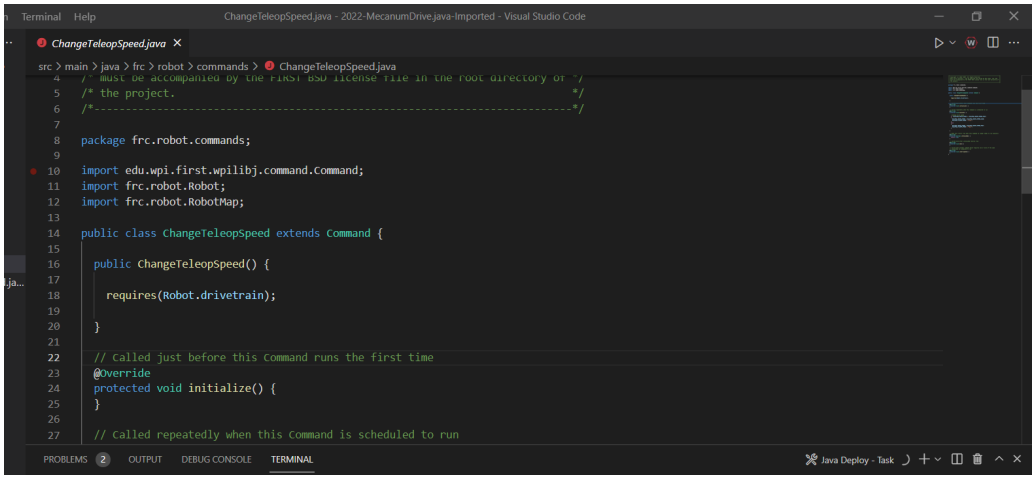
Where motor ID 1 was Front_Left_motor, and Motor ID 4 was Back_Right_motor, etc.

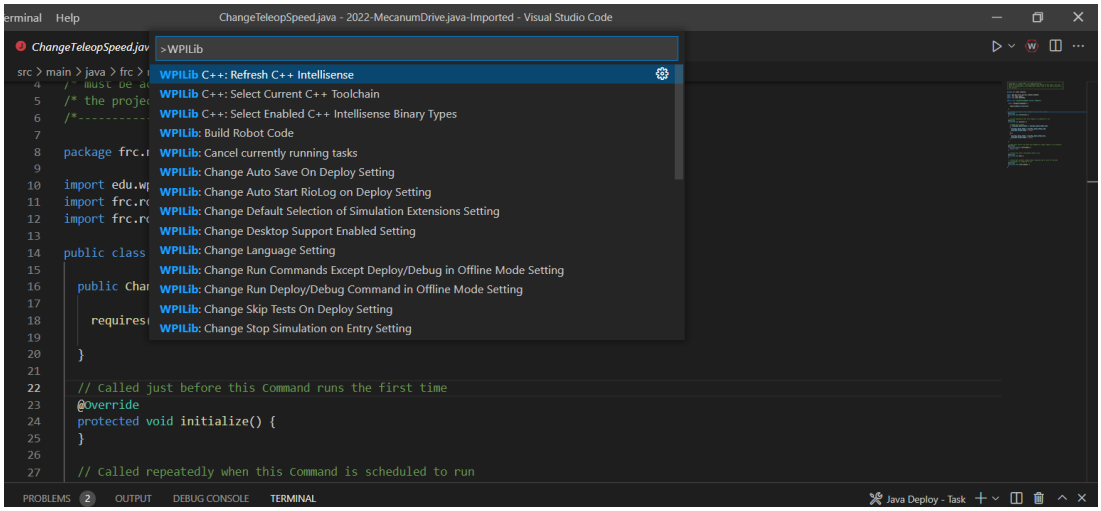
Coding and Testing Guide

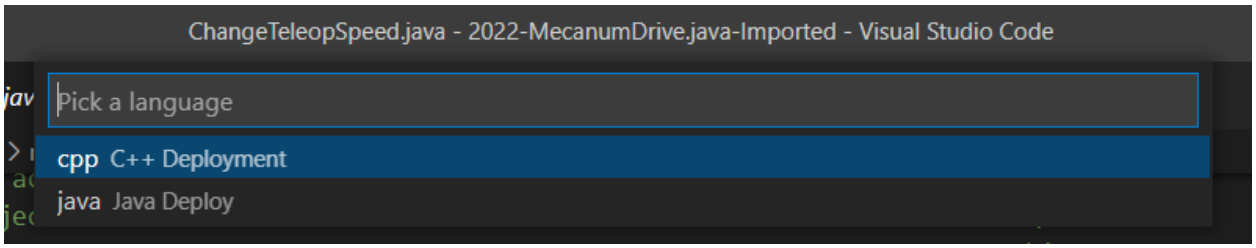
Making a new code project

Follow the FIRST website: link
Note: when you are importing classes, check the API DOCS

Basically press the  icon in the top right corner, then click on Build Robot Code and follow the rest of the instructions.

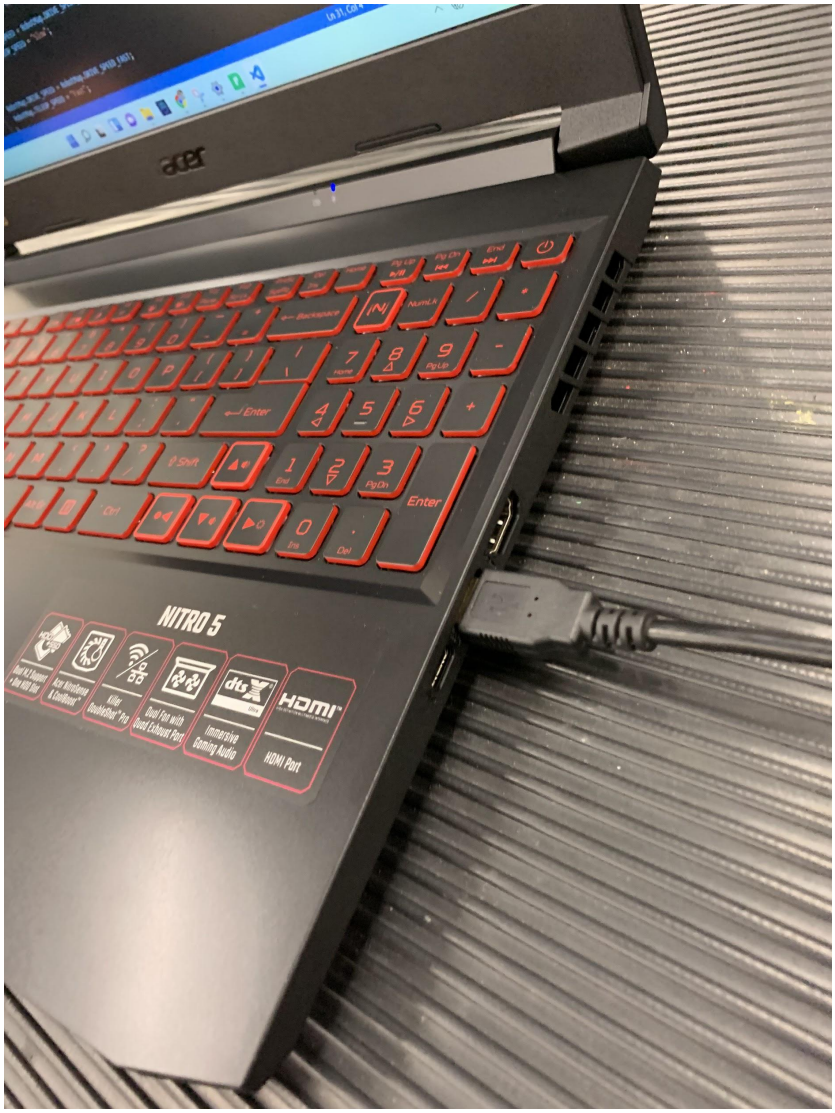


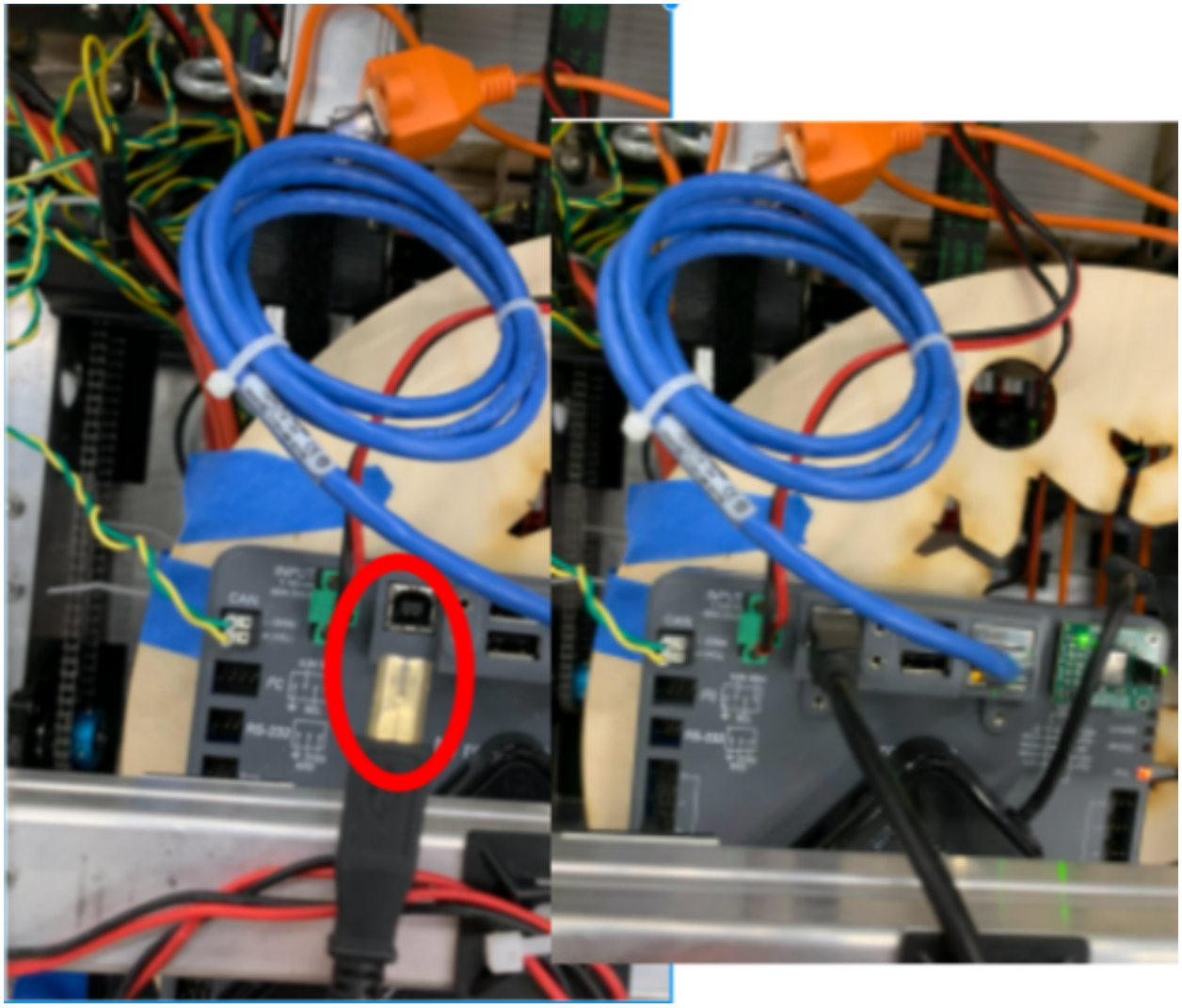




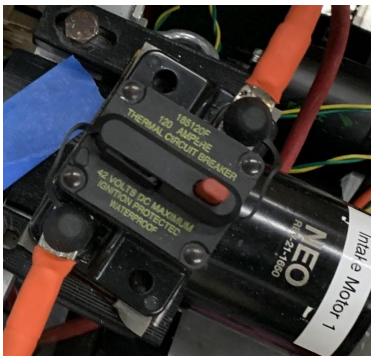
Running code on robot

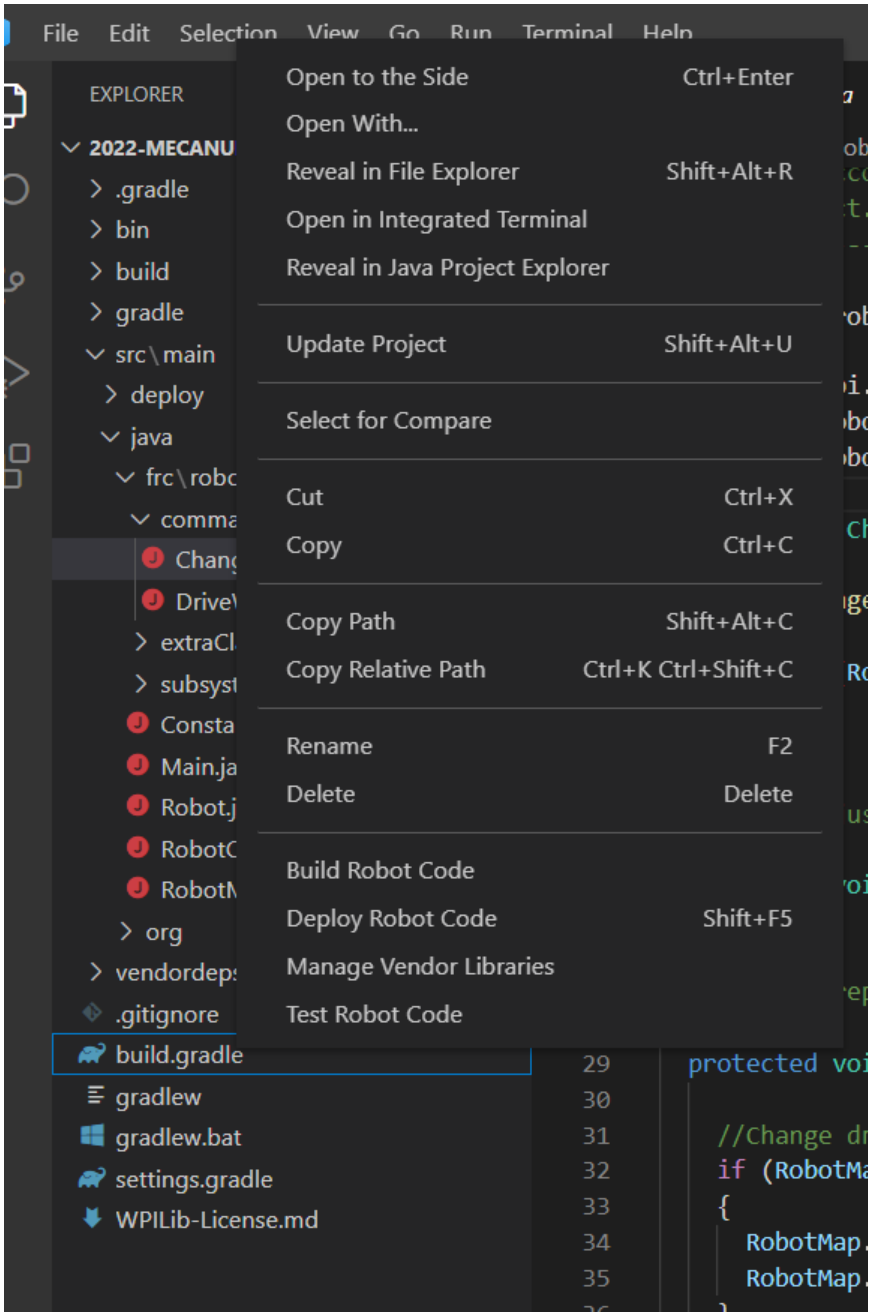
1. Plug in the plug into the computer's USB
2. Plug in the other side into the roboRIO
3. Turn the computer on by flicking the lever(to turn it off click the red button).
4. Right click on the build.gradle
5. Click **Deploy Robot Code**
    a. Make sure the FRC Control Driver app is running and has "teleoperated" set to enabled. (no picture attached yet)
6. Click java

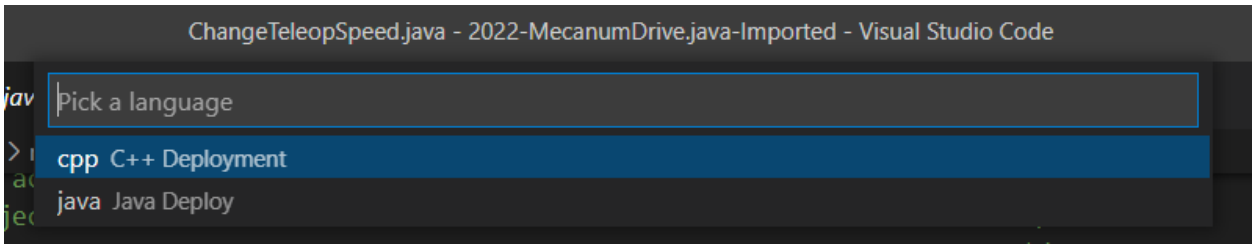





185120F
120 AMPERE
THERMAL CIRCUIT BREAKER

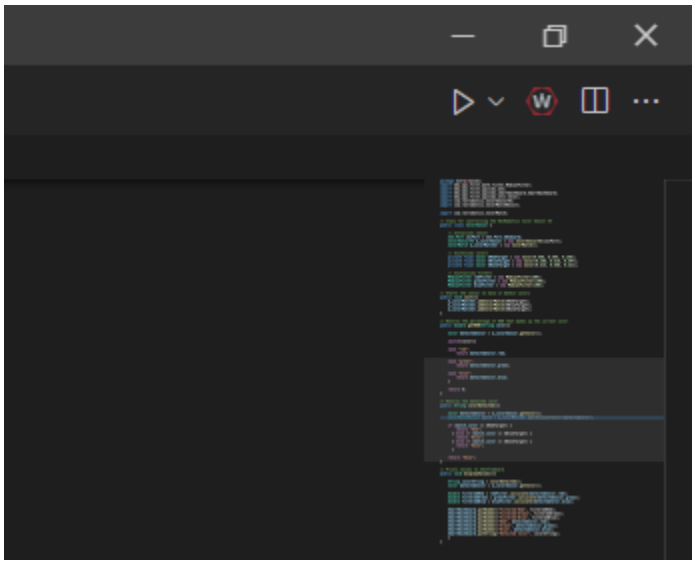

185120F
120 AMPERE
THERMAL CIRCUIT BREAKER
42 VOLTS DC MAXIMUM
IGNITION PROTECTED
WATERPROOF


File Edit Selection View Go Run Terminal Help
EXPLORER
2022-MECANU
.gradle
bin
build
gradle
src\main
deploy
java
frc\robo
comma
Chang
Drive
extraCl
subsyst
Consta
Main.ja
Robot.j
RobotC
RobotM
org
vendordeps
.gitignore
build.gradle
gradlew
gradlew.bat
settings.gradle
WPILib-License.md
Open to the Side Ctrl+Enter
Open With...
Reveal in File Explorer Shift+Alt+R
Open in Integrated Terminal
Reveal in Java Project Explorer
Update Project Shift+Alt+U
Select for Compare
Cut Ctrl+X
Copy Ctrl+C
Copy Path Shift+Alt+C
Copy Relative Path Ctrl+K Ctrl+Shift+C
Rename F2
Delete Delete
Build Robot Code
Deploy Robot Code Shift+F5
Manage Vendor Libraries
Test Robot Code
29 protected voi
30
31 //Change dr
32 if (RobotMa
33 {
34 RobotMap.
35 RobotMap.

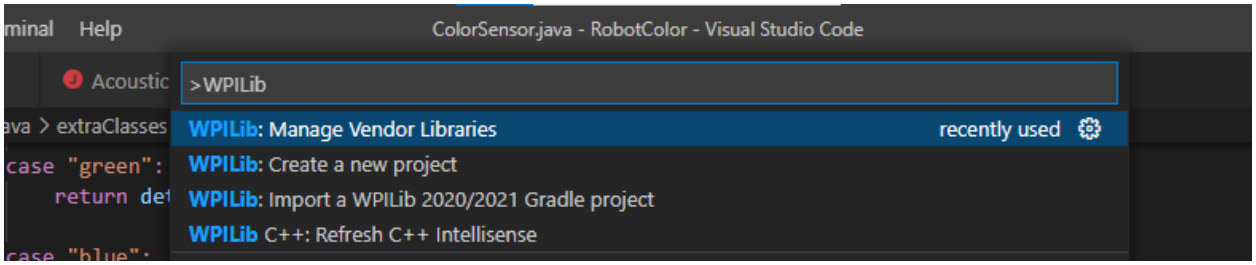

ChangeTeleopSpeed.java - 2022-MecanumDrive.java-Imported - Visual Studio Code
Pick a language
cpp C++ Deployment
java Java Deploy

How to Install Libraries

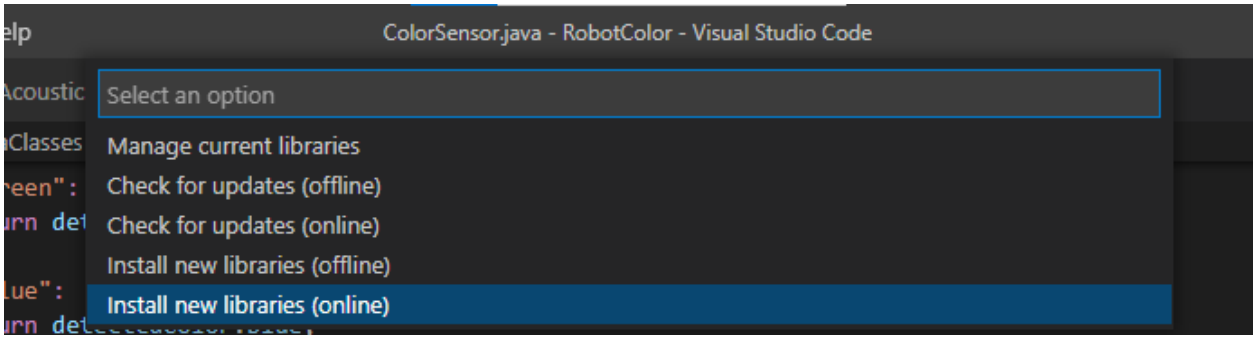
Occasionally, you might have to install a library before you are able to use certain tools. To do this, you first need to click on the “W” in the top right corner of the screen.



Once you have clicked on this, a text box should appear. In the box, go to “Manage Vendor Libraries” then go to “Import New Libraries (Online)”.

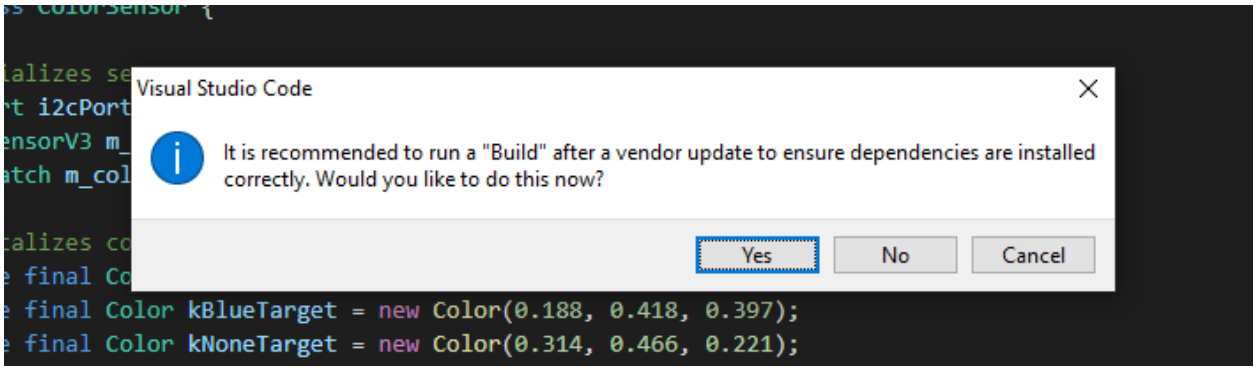


-



Next, in the box, paste in the link that the website provides to install the library.

A text box should then appear, select yes.



Once you have done this, you will now be able to compile code using those tools.

How commands work in java

initialize() runs once and once only when the command is scheduled.
After that, execute() will keep running until isFinished() turns to be true.

Subsystems

Subsystems are very simple commands. These run things like motors or pneumatics. They basically have only one job or very basic jobs. Click here for official info.

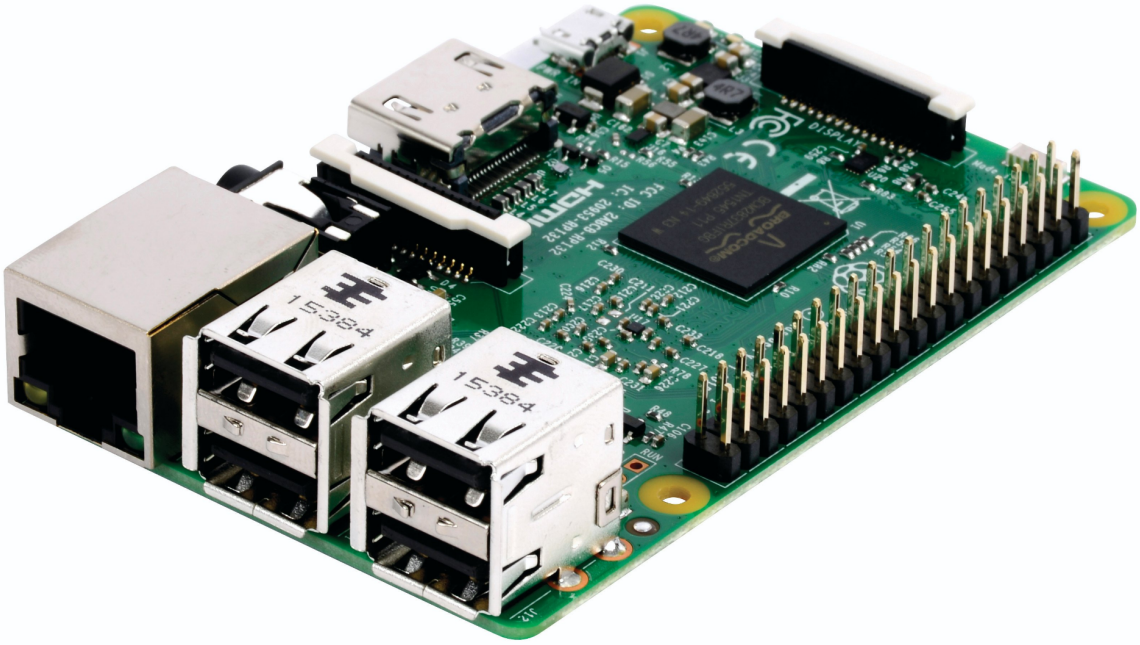
Commands

Commands are made up of multiple subsystems. Commands execute a much more complicated task, which may make use of many subsystems like motors, vision, pneumatics, arms, etc. So, commands are higher level than subsystems. More info here

Raspberry Pi stuff

Note: If there's time, follow this document to figure out how to make the raspberry pi boot up into read-only mode (where it doesn't edit files). In doing so, we lower the chances of the data getting corrupted because when the pi is on the robot, we don't get the luxury of shutting the pi down correctly. Instead the robot's power simply shuts off.

We will use Python only in the Raspberry Pi 3 which runs Linux and looks like this



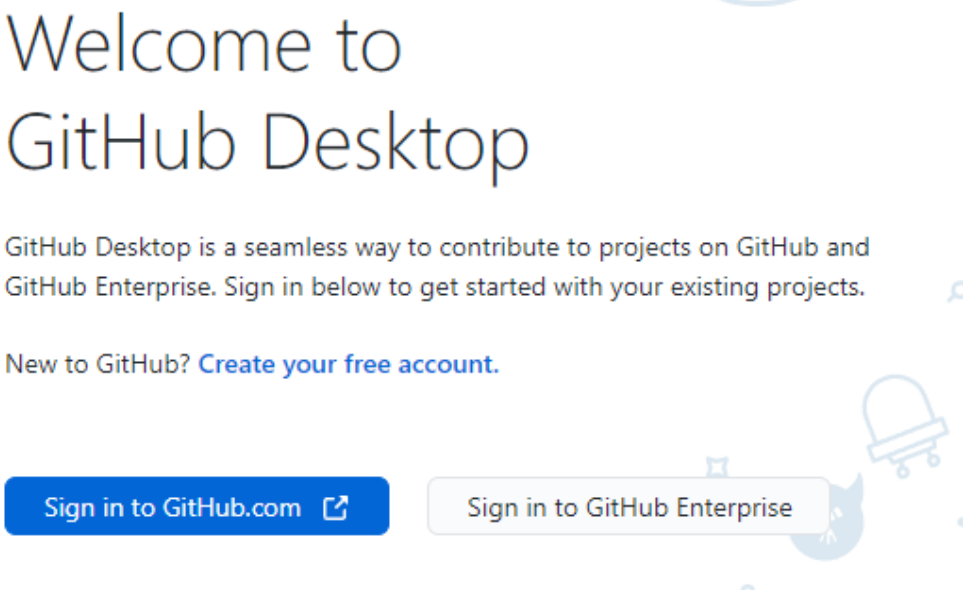
When you connect it to a monitor, keyboard, and mouse (and most of the time to a camera) you will be able to launch it. When you do, let the colorful screen pass and then it will automatically launch into command mode and run a whole bunch of commands. LET IT FINISH. Then type **startx** into the command prompt to open desktop mode.
In desktop mode, you can open file explorer to discover where files are. As of 2020, we are running TEST code in the directory: **pi\Team4121\Team4121Vision2022.py.** To run this, you can't just open it. Instead you need to run the command prompt app from the task bar near the top left and use the following commands:

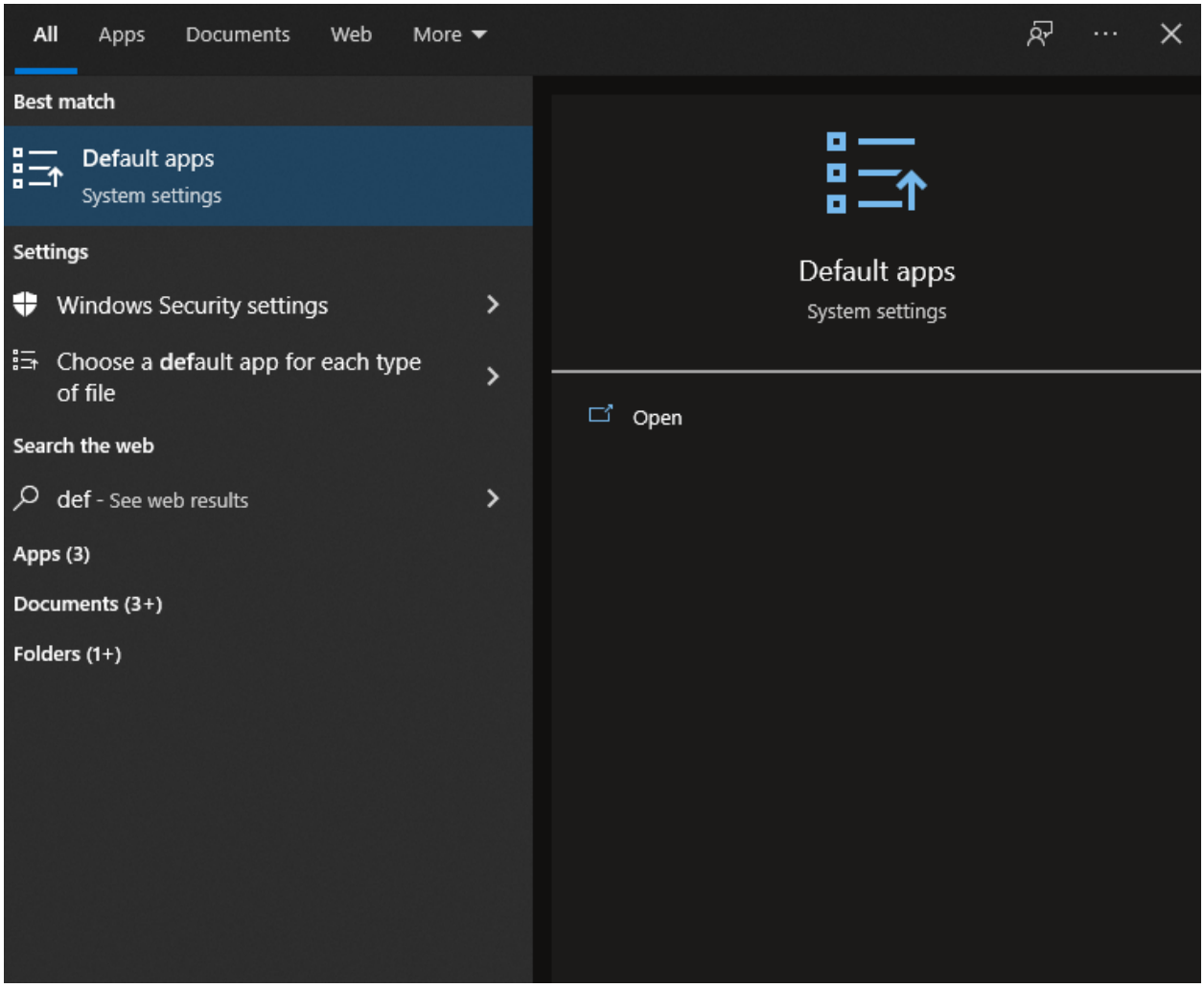
- **ls -** This is a lowercase L, not an i. Ls lists out the files in that directory and your goal is to navigate to the folders you want
- **cd -** this is used to open these folders. For example, you type ls and it lists out (System32, Pictures, Code). You would then do **cd Code** to open the Code folder.
- **python3 <file name> -** This opens a file. However it doesn't grant it permissions so you should add the keyword **sudo** in the beginning of it. For example, **sudo python3 Team4121Vision2022.py** would open the python code and grant it administrator permissions.
- **sudo pcmanfm -** In case of lack of permissions to edit files in the file explorer app, run this command. This command opens file explorer and grants the permissions to overwrite files. An easy way to remember this is to think "pacman fm" but remove the 'a'

How to use Github

First, download the desktop version of github. You can do this on school computers because it doesn't require any permissions. This app is how you can download and upload files into the github repository (another word for a program).

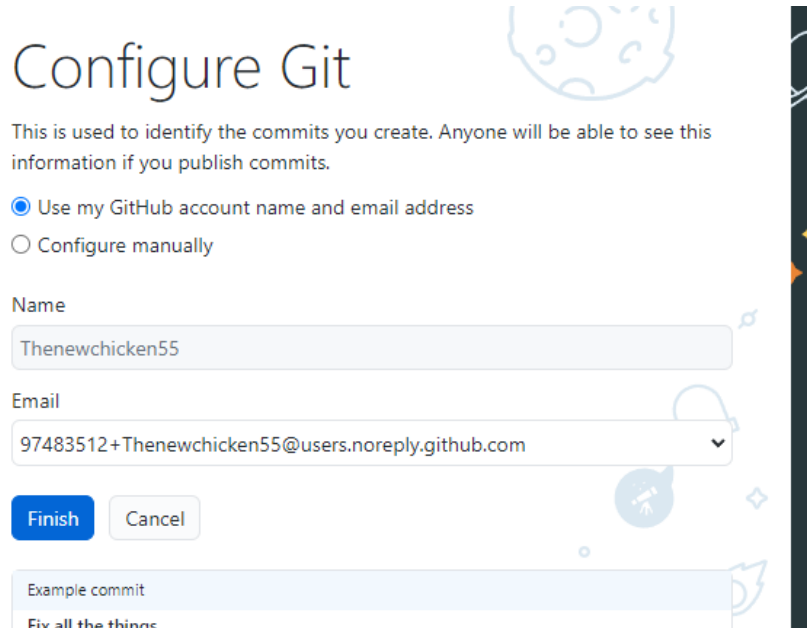
1. Download it
2. Run it
3. It's going to ask you to sign in.
    a. Click the blue button. HOWEVER, by default, windows makes the default browser Edge.
    b. But the school deleted edge from our computers
    c. So the blue button opens GitHub.com on a nonexistent browser.
    d. So to change this, go to default apps in settings and set it to chrome.



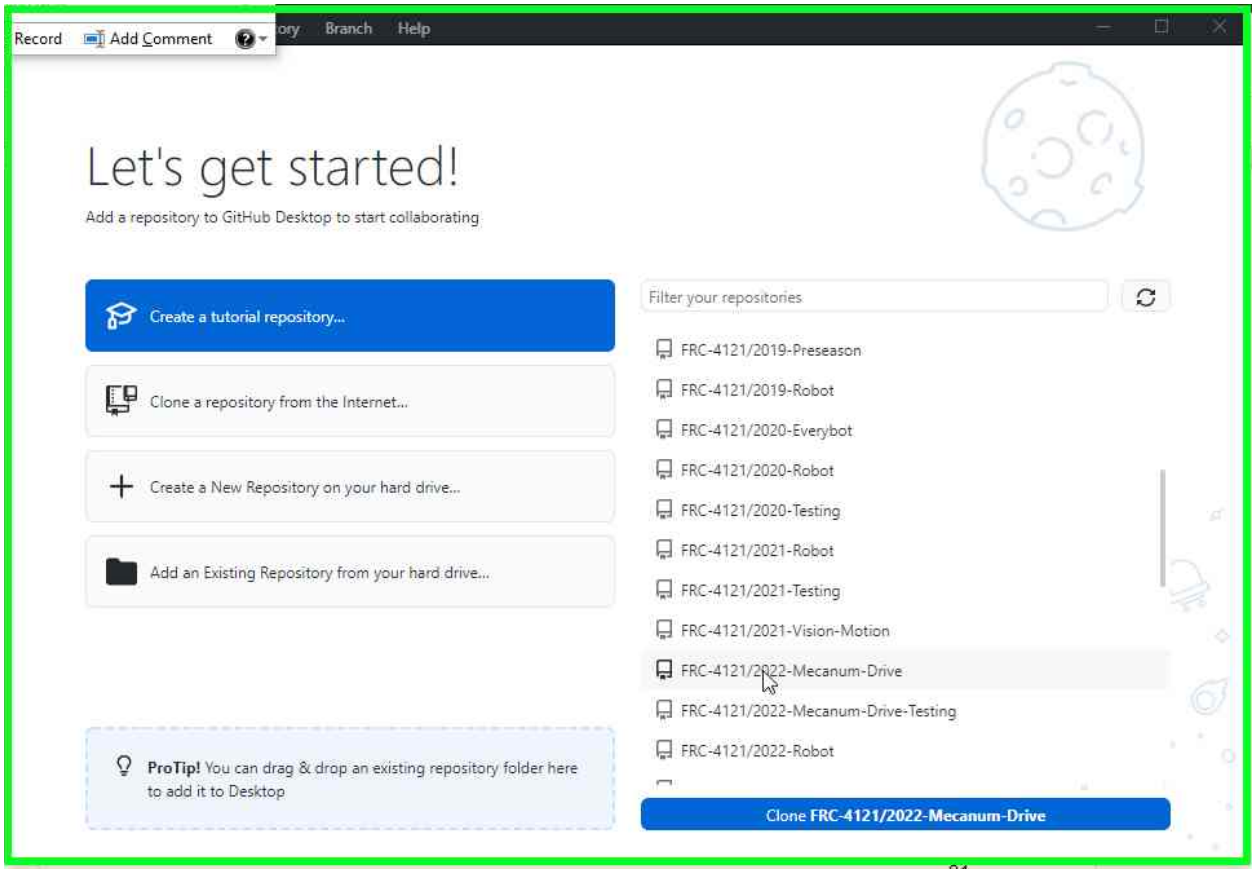


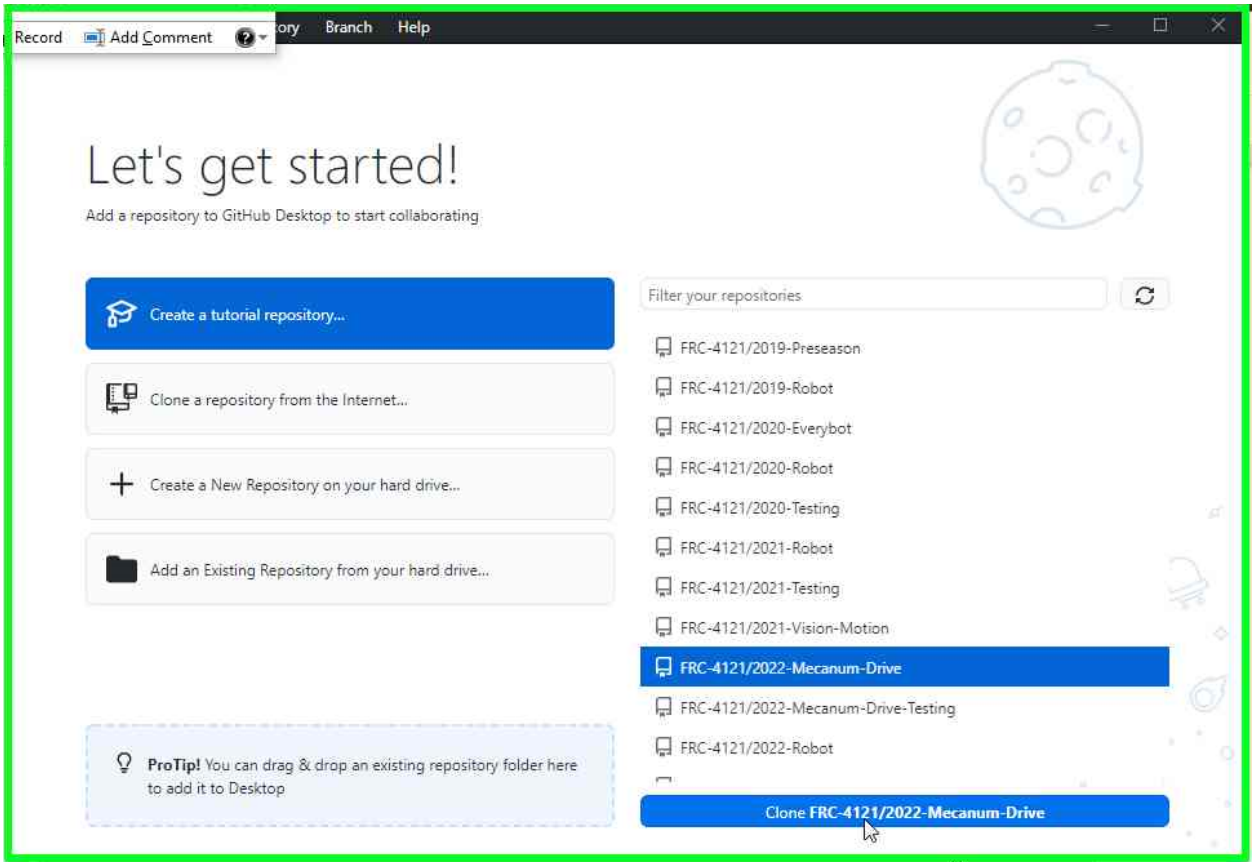
(In settings)

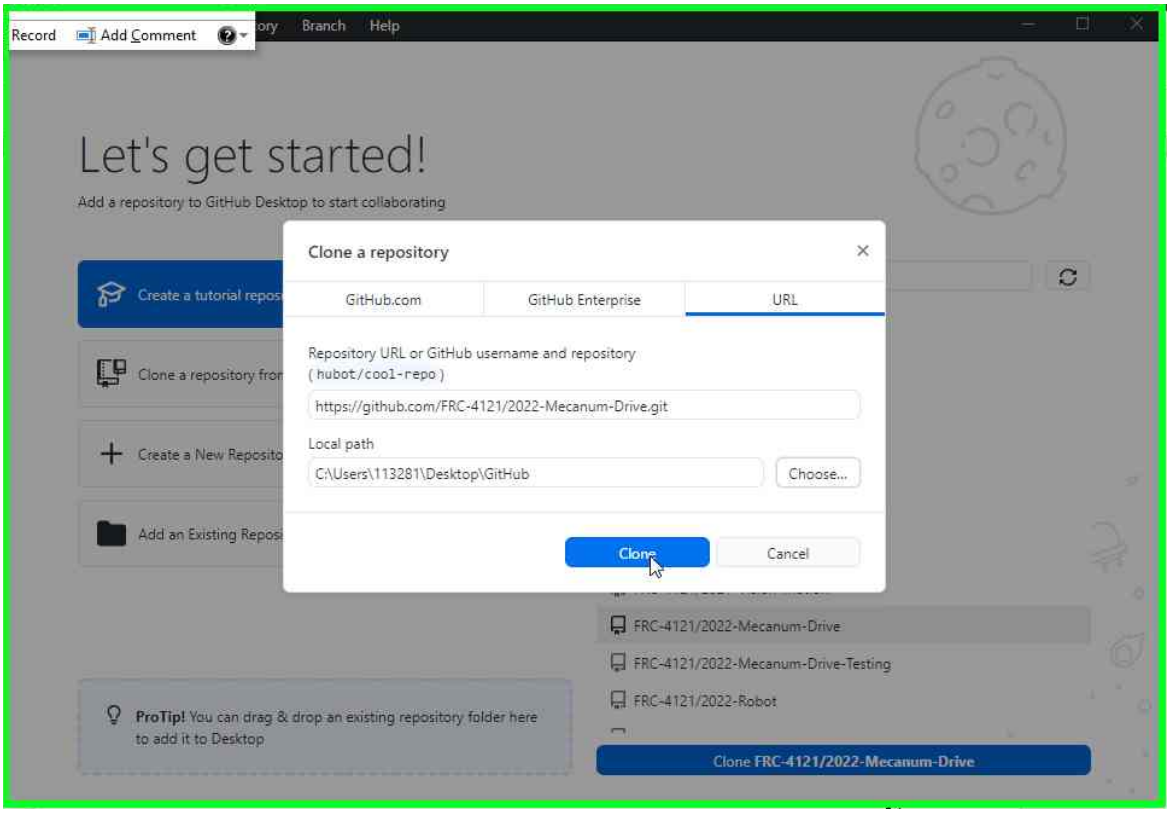
4. After you change the default browser to the right one (Chrome), you can sign into GitHub Desktop.
5. (note) Just press “finish” when you get this page
6. Now, make sure that you have permission to edit the repositories (probably ask an owner of the FRC-4121 group to make you owner).
7. Now, you will be able to make changes.



Cloning a repository

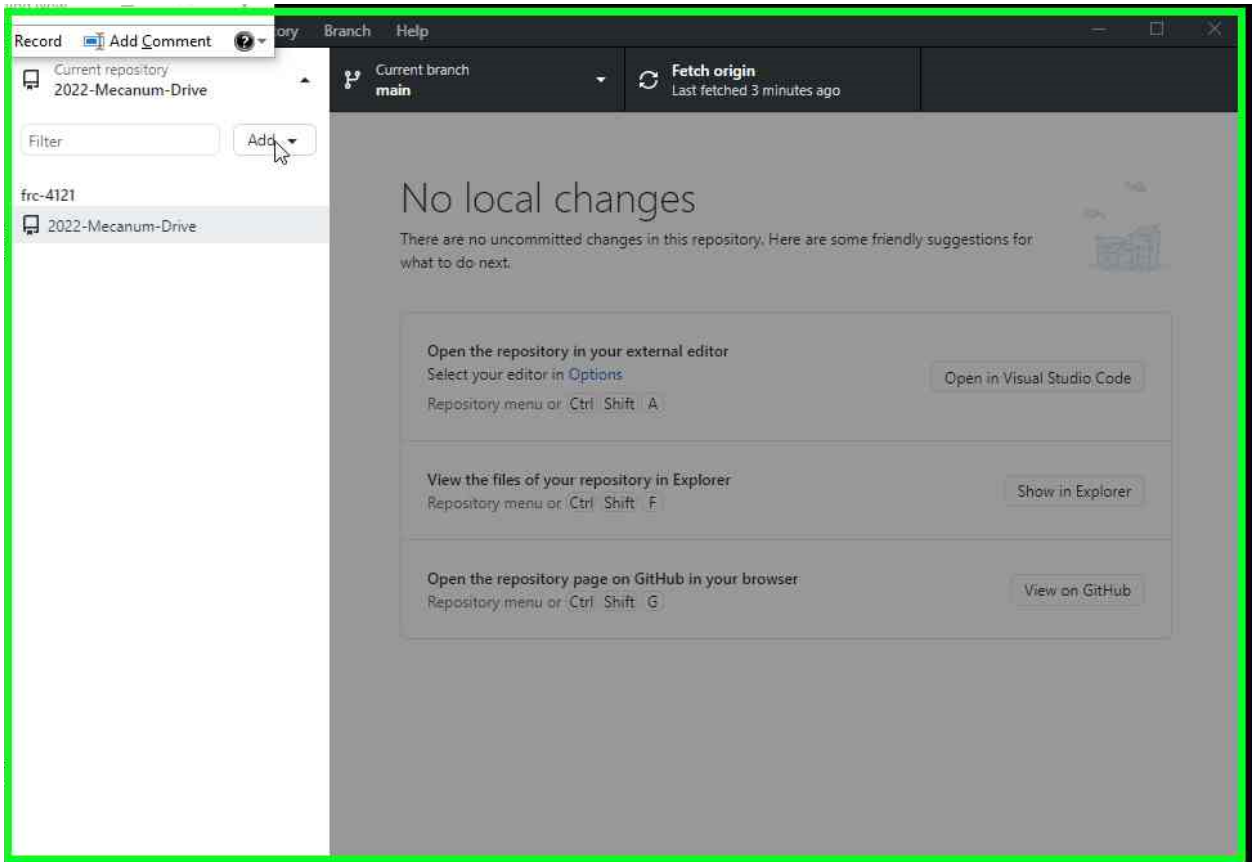


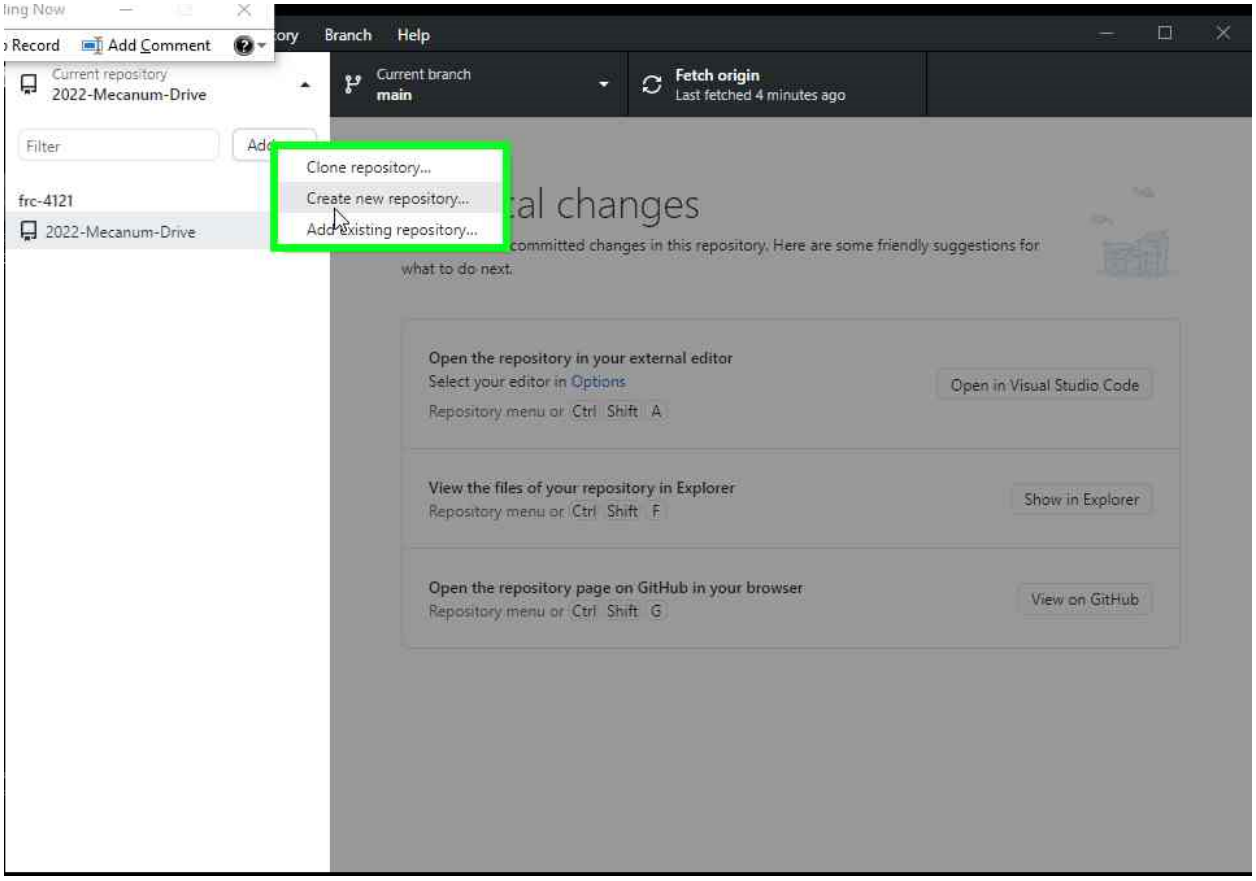


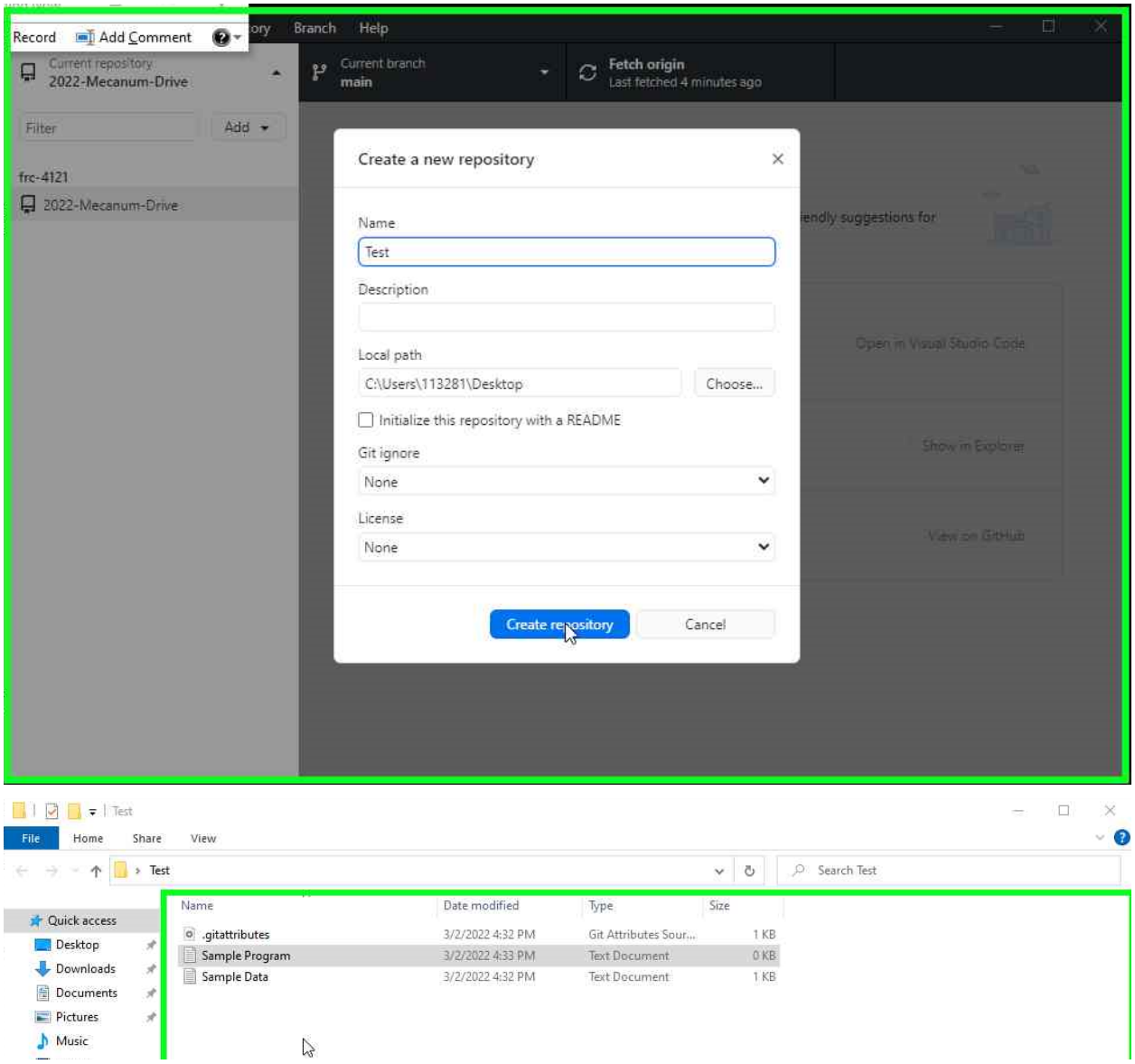


In this step, you can choose whatever directory you want. I chose desktop.

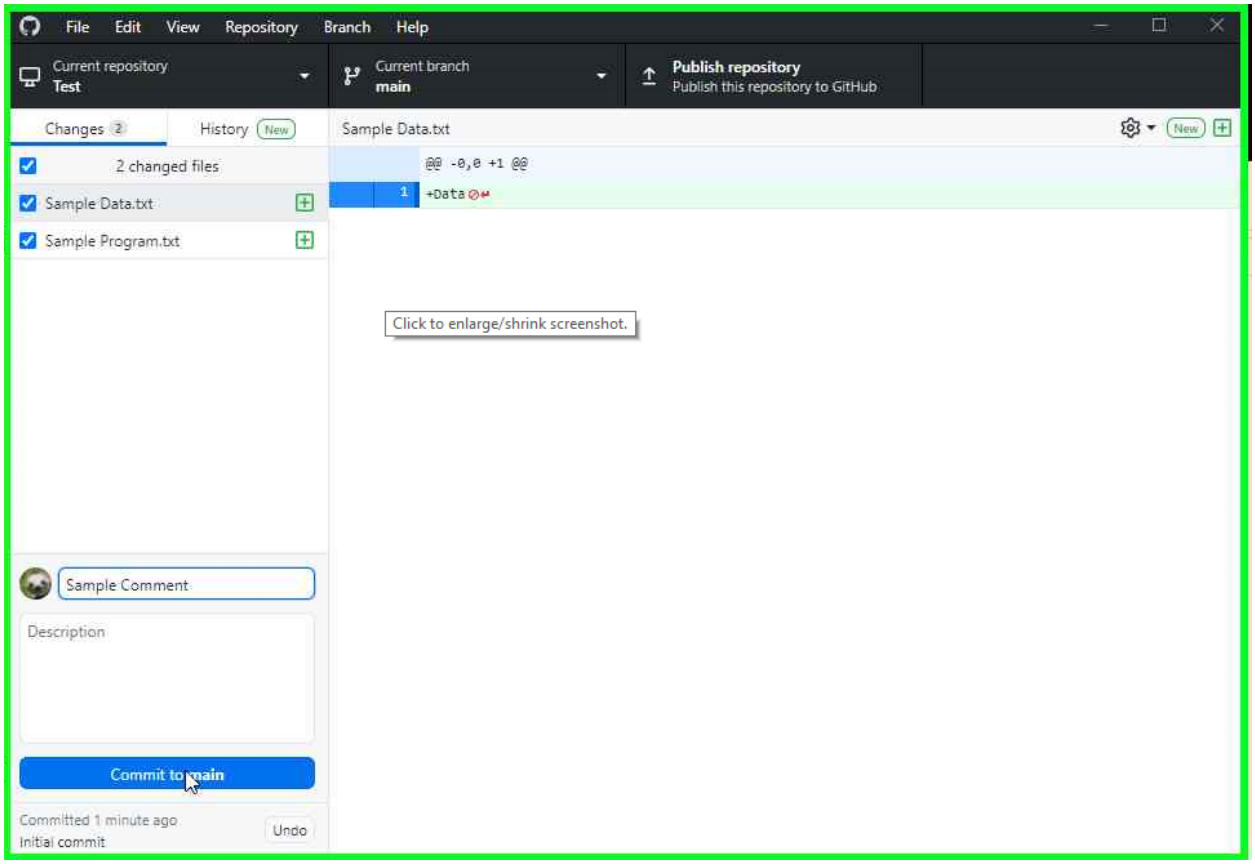
Making a new repository


Record
Add Comment
Branch
Help
Current repository
2022-Mecanum-Drive
Current branch
main
Fetch origin
Last fetched 3 minutes ago
Filter
Add
frc-4121
2022-Mecanum-Drive
No local changes
There are no uncommitted changes in this repository. Here are some friendly suggestions for what to do next.
Open the repository in your external editor
Select your editor in Options
Repository menu or Ctrl Shift A
Open in Visual Studio Code
View the files of your repository in Explorer
Repository menu or Ctrl Shift F
Show in Explorer
Open the repository page on GitHub in your browser
Repository menu or Ctrl Shift G
View on GitHub


Record
Add Comment
Branch
Help
Current repository
2022-Mecanum-Drive
Current branch
main
Fetch origin
Last fetched 4 minutes ago
Filter
Clone repository...
Create new repository...
Add existing repository...
frc-4121
2022-Mecanum-Drive
committed changes in this repository. Here are some friendly suggestions for
what to do next.
Open the repository in your external editor
Select your editor in Options
Repository menu or Ctrl Shift A
Open in Visual Studio Code
View the files of your repository in Explorer
Repository menu or Ctrl Shift F
Show in Explorer
Open the repository page on GitHub in your browser
Repository menu or Ctrl Shift G
View on GitHub

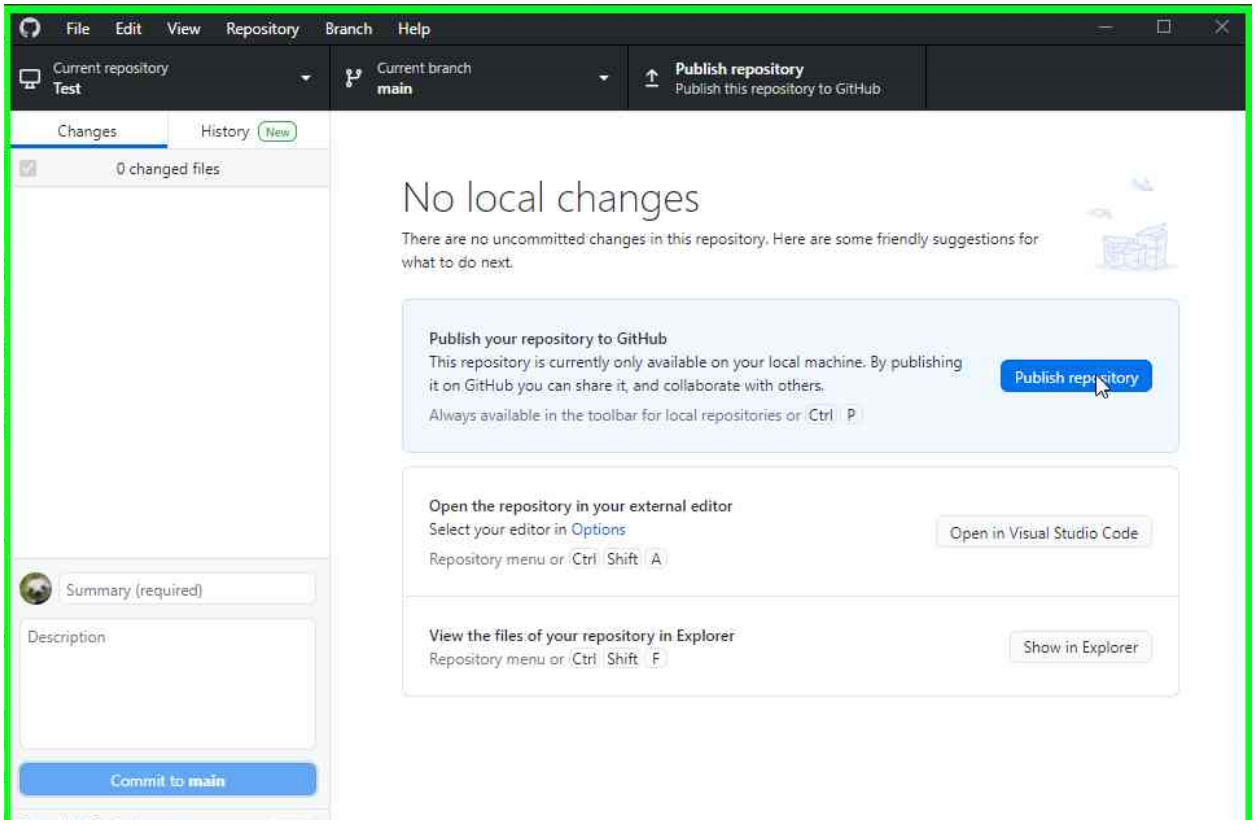


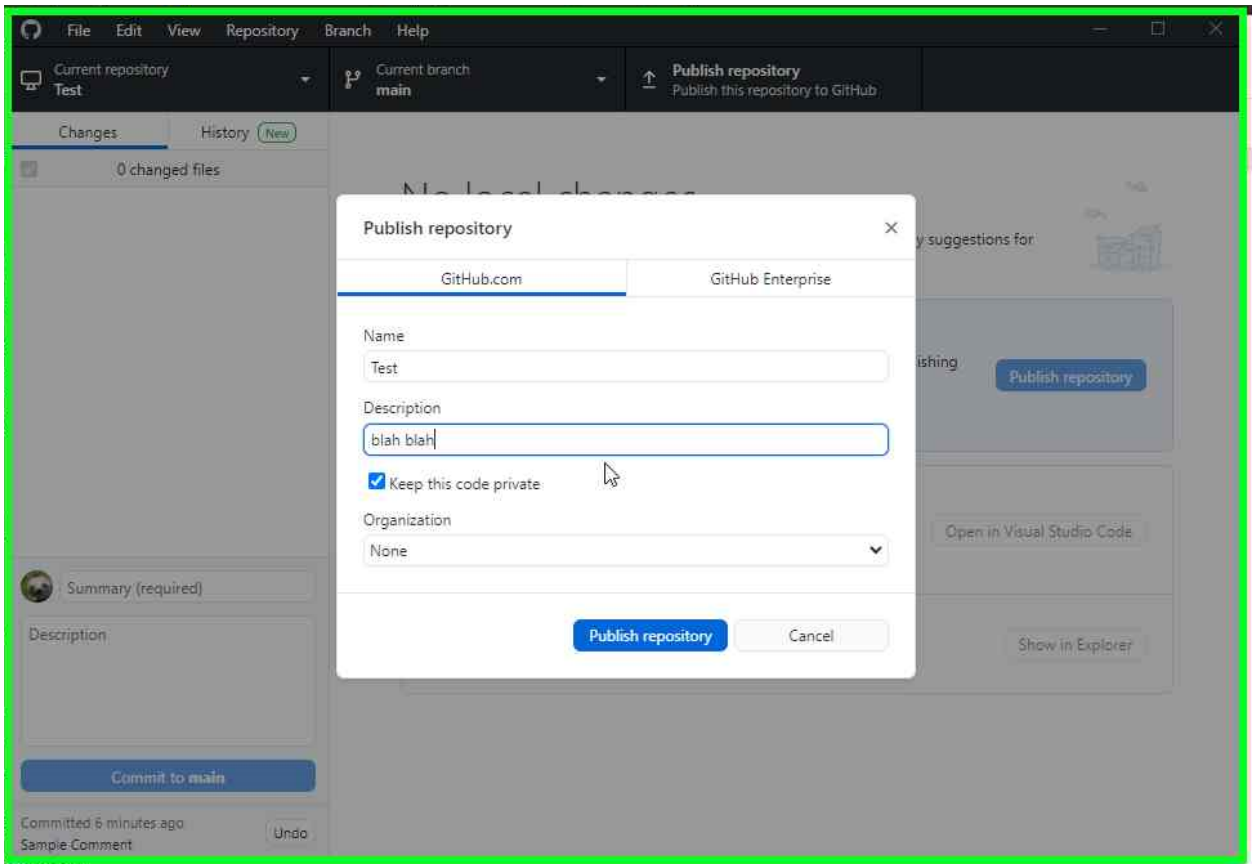
This is sample data and program in the folder that was created as a repository^^.



Once you have what you want in there (don't worry, you can edit and add more later), you must commit the data into gitHub with a comment that summarizes what that data is (in the screenshot, it's "Sample Comment" right above the cursor).

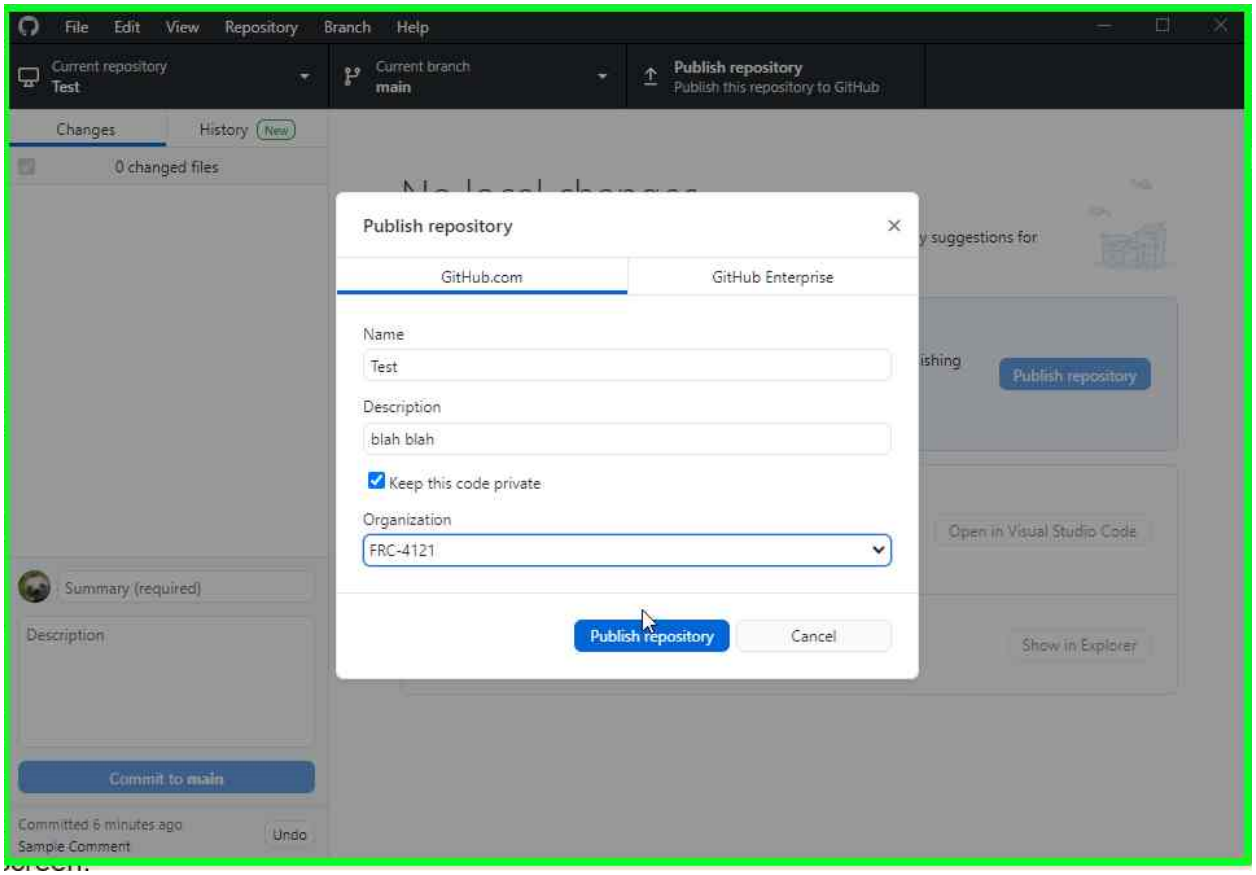
It's important to note that this commit doesn't mean that your data is in the cloud. It's still local to your computer. Once you're ready to commit to the cloud, do the following.


File Edit View Repository Branch Help
Current repository
Test
Current branch
main
Publish repository
Publish this repository to GitHub
Changes
History New
0 changed files
No local changes
There are no uncommitted changes in this repository. Here are some friendly suggestions for what to do next.
Publish your repository to GitHub
This repository is currently only available on your local machine. By publishing it on GitHub you can share it, and collaborate with others.
Always available in the toolbar for local repositories or Ctrl P
Publish repository
Open the repository in your external editor
Select your editor in Options
Repository menu or Ctrl Shift A
Open in Visual Studio Code
View the files of your repository in Explorer
Repository menu or Ctrl Shift F
Show in Explorer
Summary (required)
Description
Commit to main


File Edit View Repository Branch Help
Current repository
Test
Current branch
main
Publish repository
Publish this repository to GitHub
Changes
History New
0 changed files
Publish repository
GitHub.com
GitHub Enterprise
Name
Test
Description
blah blah
Keep this code private
Organization
None
Publish repository
Cancel
Summary (required)
Description
Commit to main
Committed 6 minutes ago
Sample Comment
Undo

Fill out your information like name and description (I don't think you can change the description later on).
HERE'S the IMPORTANT STEP vv

Selecting the right group makes sure that your repository doesn't get uploaded to your account but rather to the whole group.



Now you can finally publish it.

Sensors

Color Sensor (RevRobotics V3 Sensor)

How it Works

The way the color sensor works is by extracting the colors that make up the current frame being processed by the sensor.
(Note: The Color Sensor works best at 4-8cm).

How to Wire

The Color Sensor simply needs to be plugged into the I2C port on the RoboRIO to work.
(See Highlighted Area below).

How to Code

Before you can use it, you will need to install a library. Look at the section "How to Install Libraries." Select online installation and paste the following:

http://revrobotics.com/content/sw/color-sensor-v3/sdk/REVColorSensorV3.json

The following will show you how to use the color sensor for your own systems along with the system we created for using the sensor.

First, import the following libraries needed to use the sensor. (You don't need the "SmartDashboard" part to use the sensor, but for the class we made to use the sensor, you do need it.)

```
import edu.wpi.first.wpilibj.I2C;
import edu.wpi.first.wpilibj.smartdashboard.SmartDashboard;
import edu.wpi.first.wpilibj.util.Color;
import com.revrobotics.ColorSensorV3;
import com.revrobotics.ColorMatchResult;
import com.revrobotics.ColorMatch;
```

Next, you need to initialize the I2C port, which is the port the Color Sensor is connected to on the RoboRIO. Afterwards, you need to create a new ColorSensorV3 and a new ColorMatch object.

```
// Intializes sensor
I2C.Port i2cPort = I2C.Port.kOnboard;
ColorSensorV3 m_colorSensor = new ColorSensorV3(i2cPort);
ColorMatch m_colorMatcher = new ColorMatch();
```

Like what was said previously, The way the color sensor works is by extracting the colors that make up the current frame being processed by the camera. If you want to create a new color, you specify the percent of the RGB values, ***NOT*** 8 bit values (Technically you can with by using the 8BitColor class but it's better to use the percentages.)

```
// Initalizes colors
private final Color kRedTarget = new Color(0.468, 0.395, 0.168);
private final Color kBlueTarget = new Color(0.188, 0.418, 0.397);
private final Color kNoneTarget = new Color(0.314, 0.466, 0.221);
```

Next, run the “init” function to add the colors to the color matcher object created previously. This will add the colors to the list of colors to detect. Once you run this function, it will start looking for colors.

```
// Starts the sensor to have it detect colors
public void init(){
    m_colorMatcher.addColorMatch(kRedTarget);
    m_colorMatcher.addColorMatch(kBlueTarget);
    m_colorMatcher.addColorMatch(kNoneTarget);
}
```

We also added a “getRGB” function to our class that returns the percentage of the color passed into the function.

```
// Returns the percentage of RGB that makes up the current color
public double getRGB(String color){

    Color detectedColor = m_colorSensor.getColor();

    switch(color){

    case "red":
        return detectedColor.red;

    case "green":
        return detectedColor.green;

    case "blue":
        return detectedColor.blue;
    }

    return 0;
}
```

You can also call “colorDetected” in our class to return the color that is closest to the current image. This is the important part as this is the part that actually senses the color, not just percent values.

```
// Returns the detected color
public String colorDetected(){

    Color detectedColor = m_colorSensor.getColor();
    ColorMatchResult match = m_colorMatcher.matchClosestColor(detectedColor);

    if (match.color == kRedTarget) {
        return "Red";
      } else if (match.color == kBlueTarget) {
        return "Blue";
      } else if (match.color == kNoneTarget) {
        return "None";
      }

    return "None";
}
```

The last function in our class is the “displayValues” class that displays the current values of the sensor on Shuffleboard.

```
// Prints values to Shuffleboard
public void displayValues(){

    String colorString = colorDetected();
    Color detectedColor = m_colorSensor.getColor();

    SmartDashboard.putNumber("Red", detectedColor.red);
    SmartDashboard.putNumber("Green", detectedColor.green);
    SmartDashboard.putNumber("Blue", detectedColor.blue);
    SmartDashboard.putString("Detected Color", colorString);
    }
}
```

If you need help setting up and testing your code, refer to these previous topics in the guide:

1. How to make a new code project
2. How to run your code onto the robot
3. How to install new libraries

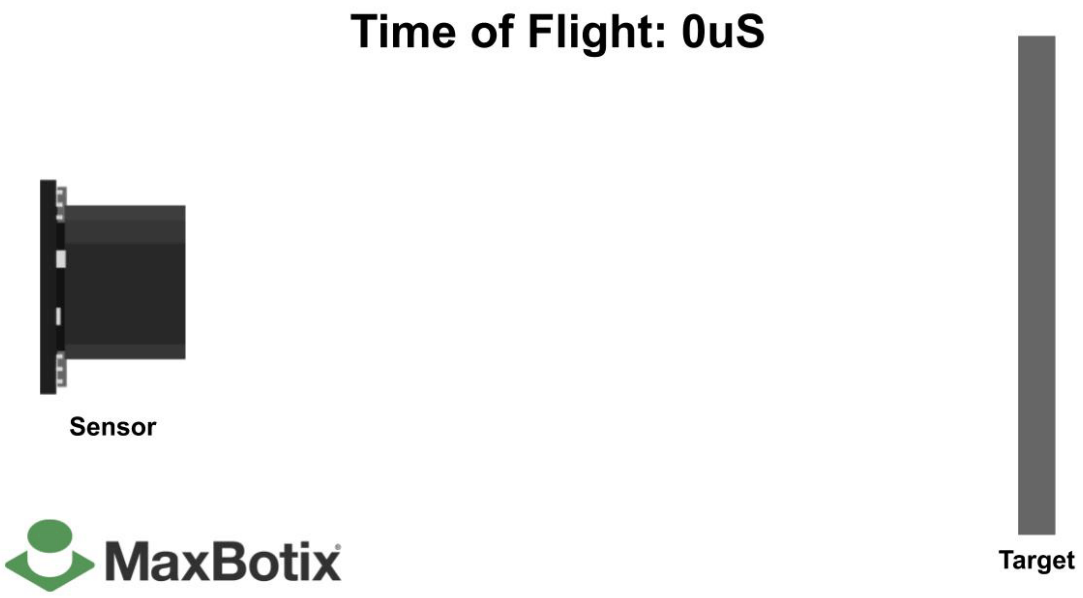
Sample Code

Here’s the link to the sample code

Acoustic Sensor (MaxBotix MB1043 MaxSonar Sensor)

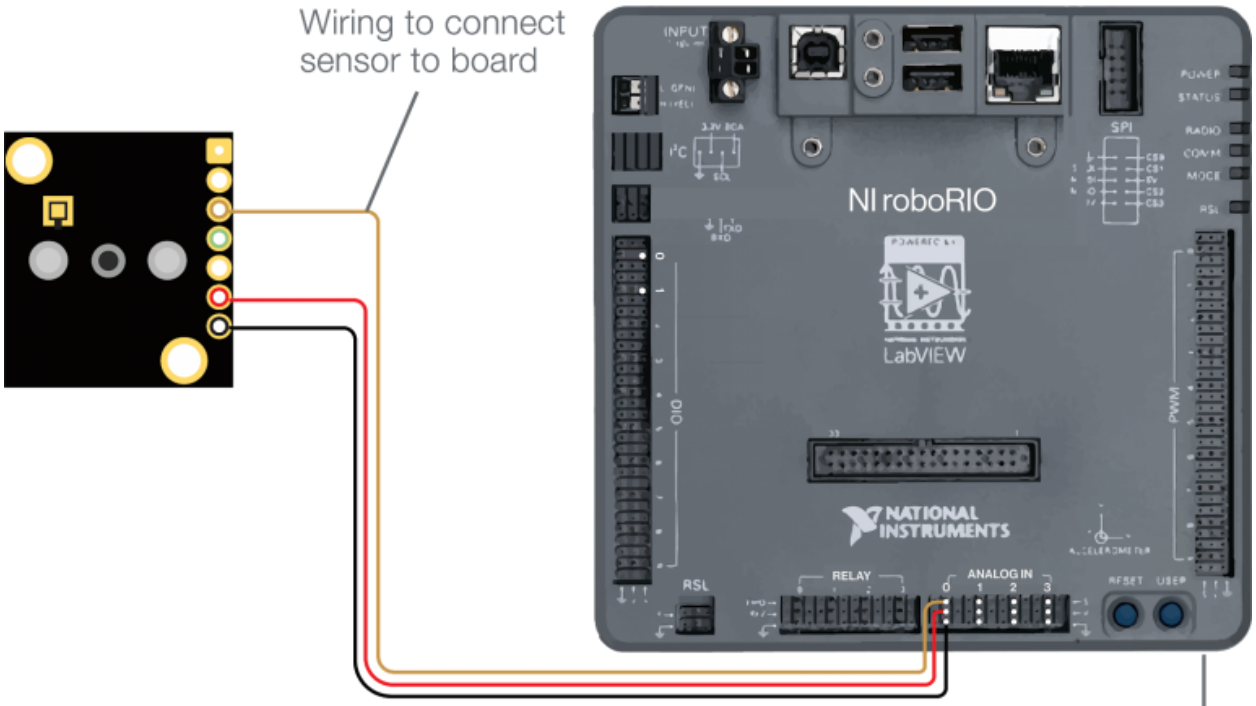
How it Works

The Acoustic Sensor (Also known as an ultrasonic sensor) uses sound waves to detect the distance of objects in front of it. The way this works is by taking the amount of time it took for the sound wave to travel to the object and back. It takes this amount of time and puts it into a mathematical equation in order to detect distance. For an example look to the diagram below: (Note: The Acoustic Sensor works best from 12-30 inches)



How to Wire

Now that you have a general understanding of how the Acoustic Sensor works, it is time to explain how to actually wire this sensor. The Acoustic Sensor uses 3 wires that go into an Analog In port. Looking at the Acoustic Sensor, you will see a line of multi-colored circular ports, with a square port at the top. Ignore the square port and you will want to use the yellow, red, and black circular ports. Connect all three wires to the Analog In (0) port in the order of yellow, red, and black from top to bottom. For an example look to the diagram below:



How to Code

First, to use this sensor, import the following:

```
import edu.wpi.first.math.filter.MedianFilter;
import edu.wpi.first.wpilibj.AnalogInput;
import edu.wpi.first.wpilibj.RobotController;
import edu.wpi.first.wpilibj.smartdashboard.SmartDashboard;
```

Next, create an Analog Input and if you want, and a Median Filter (The median filter will take the median of the values to avoid the sensor switching distance values rapidly, but it is not required.)

```
private final AnalogInput sensor = new AnalogInput(0); //Change the number to correspond to the port it is in
MedianFilter filter = new MedianFilter(100);
```

The “getValue” function in our class returns the distance in inches the sensor detected. It gets the raw value, does a calculation to account for voltage and calculates the distance by taking

the raw distance times the voltage scale factor times 0.0492. That calculates the final value which can be calculated by a filter for extra precision if you choose to use one.

```
// Returns inches
public double getValue(){
    double rawValue = sensor.getValue();
    double voltage_scale_factor = 5/RobotController.getVoltage5V();
    double currentDistanceInches = rawValue * voltage_scale_factor * 0.0492;
    filter.calculate(currentDistanceInches);
    return currentDistanceInches;
}
```

The final function in our class puts the distance on the Smartboard.

```
public void displayValues(){
    double currentDistanceInches = getValue();
    SmartDashboard.putNumber("inches", currentDistanceInches);
}
```

If you need help setting up and testing your code, refer to these previous topics in the guide:

1. How to make a new code project
2. How to run your code onto the robot
3. How to install new libraries

Sample Code

Here's a link to the sample code.

Lidar Sensor (Garmin Lidar Lite v3)

How it Works

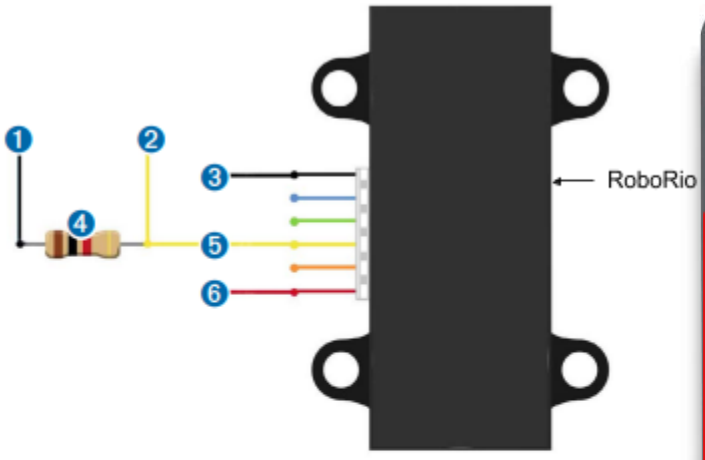
The Lidar Lite v3 works by sending out an infrared laser that bounces back after hitting an object. It takes the amount of time taken to bounce back to the Lidar and puts that into a mathematical equation in order to get the distance of the object.

How to Wire (PWM Wiring)

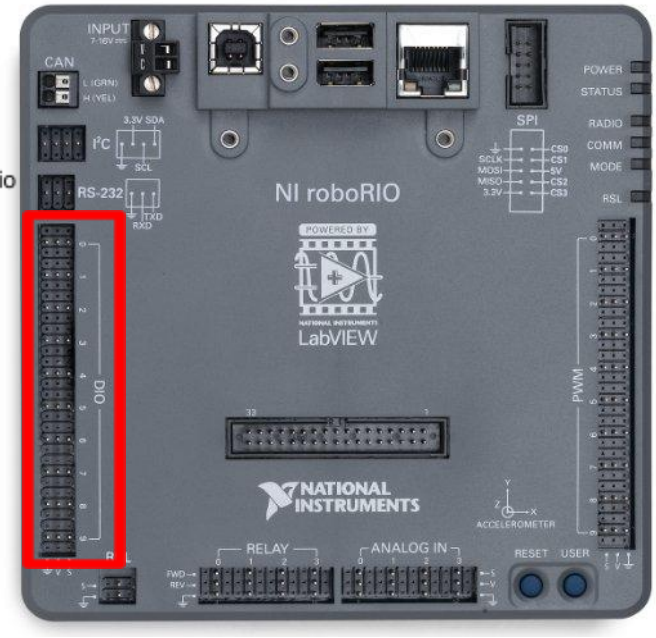
In order to wire the Lidar Sensor, you have to plug it into the DIO port (Highlighted below).

Next, you will need to use 3 of its wires. You will also need a 1K ohm resistor. You need to connect the yellow and black wires via the resistor and then plug all three of the wires in the order of black, yellow, then red from left to right into the same DIO port for each wire.

PWM Wiring



Item	Description	Notes
1	Trigger pin on microcontroller	Connect the other side of the resistor to the trigger pin on your microcontroller.
2	Monitor pin on microcontroller	Connect one side of the resistor to the mode-control connection on the device, and to a monitoring pin on your microcontroller.
3	Power ground (-) connection	Black Wire
4	1kΩ resistor	
5	Mode-control connection	Yellow wire
6	5 Vdc power (+) connection	Red wire The sensor operates at 4.75 through 5.5 Vdc, with a max. of 6 Vdc.



How to Code

To start, create a new counter object.

```
private Counter counter;
```

Next, initialize the sensor. Our class has a function for this. Make sure to remember that you need to pass in the DIO port when setting up the sensor. It also sets up the counter.

```
// Creates a new LidarSensor object, takes in the DIO port the sensor is connected to and configures pulses
public LidarSensor (DigitalSource source) {
    counter = new Counter(source);
    counter.setMaxPeriod(1.0);
    counter.setSemiPeriodMode(true);
    counter.reset();
}
```

The other function in the class returns the distance in inches. A calculation is done to get the distance in cm and then the line after it changes it into inches. If you want the distance in cm, remove the line after the cm calculation line.

```
// Returns the distance of the sensor in inches
public double getDistance() {
    double cm;
    double inches;
    cm = (counter.getPeriod() * 1000000.0 / 10.0);
    inches = (cm / 2.54);
    return inches;
    }
}
```

You can then make a new Lidar object like this (In this case, the DIO port is #0)

```
LidarSensor sensor = new LidarSensor(new DigitalInput(0));
```

If you need help setting up and testing your code, refer to these previous topics in the guide:

1. How to make a new code project
2. How to run your code onto the robot
3. How to install new libraries

Sample Code

Here's a link to sample code.

Photoelectric Sensor (We couldn't figure out)

Link to the best source of information for wiring.
Link to the code for it.

Characterization Link

Link

PID Controller Basics

P.I.D means proportional, integral, derivative control system.
It is used to control the voltage going into the motor to make it more consistent and fluctuate less.
Error signal: this is the difference between the target and actual values. It's the amount that needs to be corrected by.
Proportional: is basically using a factor to multiply with the error signal to get correction to input.
Integral: sums the error over time
Derivative: change over time. How fast is error changing over time. This is the constant factor to add to proportional.