FTC PROGRAMMING GUIDE

March 2, 2019

Dwight-Englewood Robotics

Contents

1. Control System

In this section, we will give an overview of the hardware behind the programming aspect of FTC. Note that the information here is specific to using REV stuff; Modern Robotics is something that while (maybe) still legal, you should avoid using.

Rough explanations of the various sensors, motors, etc will be provided. However, detailed information on usage and the like are in a separate section.

1.1. REV Expansion Hub

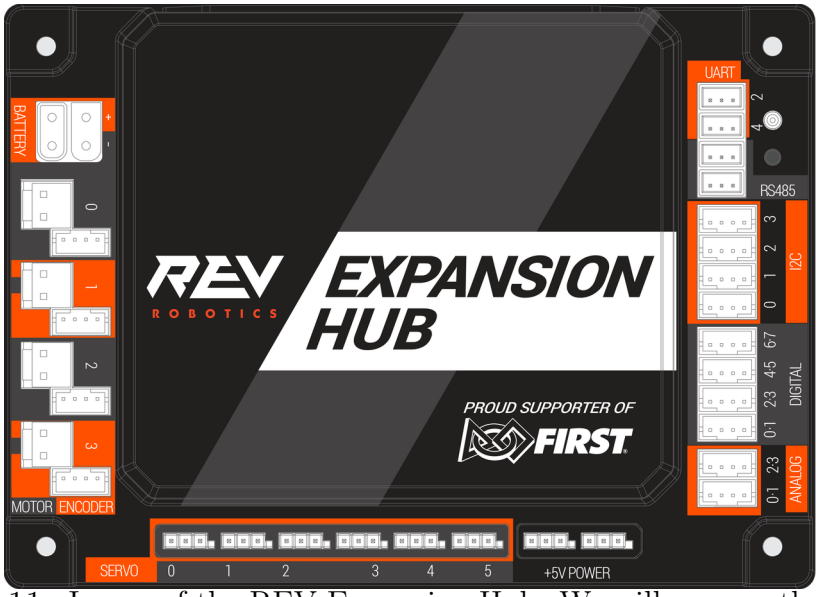


FIGURE 11. Image of the REV Expansion Hub. We will go over the specifics of what each port is for in this section.

The REV Expansion Hub is the core of the current control system. This is the piece of hardware which controls the various pieces of hardware on the bot.

1.2. Phones

1.3. Gamepad

1.4. Motors

1.5. Servos

1.6. Sensors

1.7. Wheels

1.8. Wiring

2. Programming in FTC

In this section, we will go over the high-level overview of programming for FTC. Specific guidelines for writing the actual code for the robot will be in different sections.

2.1. Blocks

2.2. Java

2.2.1. OnBot Java

2.2.2. Android Studio

3. Vision

While this really should go under sensors, it is a big enough category to warrant a section for itself.

3.1. What is it

We have previously discussed various sensors. However, the limitation inherent with all of them is the fact that they are all highly specialized with specific target ranges. The primary advantage of vision systems is that they offer a much larger range of view.

3.2. OK but why do I need it

3.3. Vuforia

3.4. PixyCam

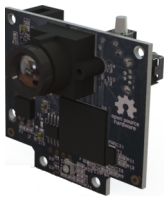


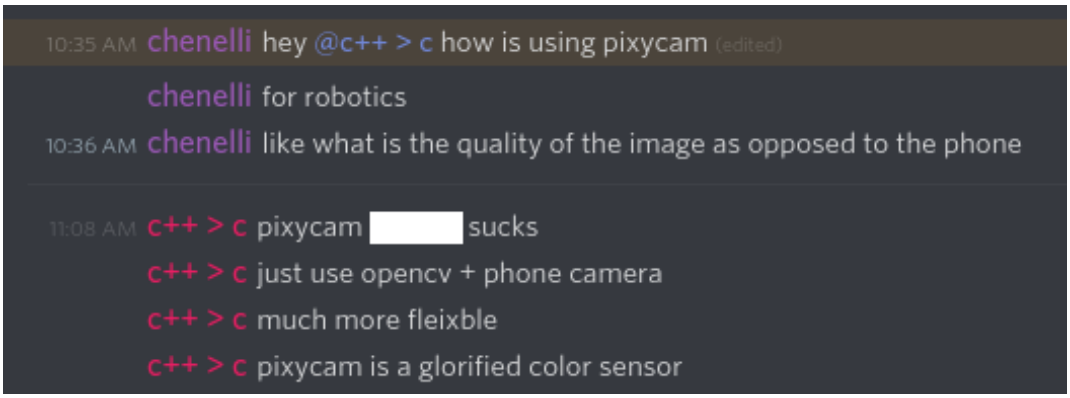
FIGURE 31. A PixyCam, so you know what it looks like.

PixyCam is another vision system. It provides object recognizing capability by grouping together various pixels of a similar color. The color, thresholds, etc. can be set using a program you install on your computer. The main takeaway from this is that the Pixy is best suited to identifying objects of a roughly uniform color, based on the presence of that color in the image it sees.

TOOD: Insert some pictures of the training thing here. Requires using a pixy

We do not have example code for using a PixyCam, since we have never used it on our robot. This is for a couple of reasons. The first of which is that PixyCam is INCREDIBLY susceptible to lighting changes. It is highly likely that on competition day your settings will not work, depending on the lighting conditions at the meet. The second is that even when under proper settings, it isn't that great. It uses RGB, which is not the greatest for color detection. It has tons of false positives and noise, and all around isn't great. It works best on bright colors - the particles/jewels from Velocity Vortex/Relic Recovery, the relic from Relic Recovery, and the gold mineral from Rover Ruckus would work well. The glyphs from Relic Recovery worked horribly.

3.4.1. **TL;DR**



3.5. **Tensorflow**

Tensorflow is an interesting vision system. The premise is this; there is some neural network trained to identify different pieces present in the game. This was first done in the Rover Ruckus season, as since I am currently writing this while that season takes place, I don't know whether it will be done again, as FTC trained the model to detect the minerals themselves. It is something you could do yourself, see subsection 3.6 DIY Neural Network.

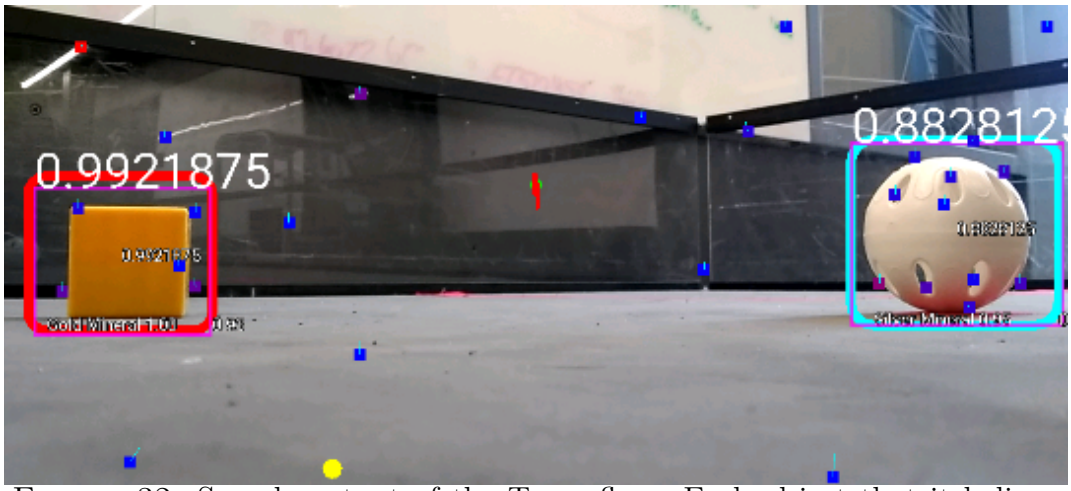


FIGURE 32. Sample output of the Tensorflow. Each object that it believes to be a trained object, in this case the minerals, is boxed. Next to each box is a number, representing how confident the neural network is that the box is a mineral. In the FTC SDK, methods for getting the location of the box, confidence, and type of identified object the box contains are exposed

So, what does this look like from a programming standpoint? A full teleop which implements the Tensorflow identification (at least for the 2018-2019 season) can be found in TODO: figure out appendixes.

For now, assume that `tfod` is a standard Tensorflow object provided by the FTC SDK.

3.6. **DIY Neural Network**

A quick disclaimer before we get into this section, this will method for vision will take a long time (and if you do not have a GPU, even longer), so only use it if nothing is working. It requires a lot of patience and a lot of StackOverflow tabs to be opened, but if you really want to do it go for it. In order to train, you need to atleast have python installed.

The specific algorithm you will have to use is called YOLO object detection. In short, it takes in continuous video and splits the frame into a bunch of little rectangles and trys to find whatever you are looking for in those little rectangles. If it finds it, it'll but an little box around it.

The first part of this method is to find find and annotate your dataset. Look around the internet or take some pictures and start to gather a dataset. Once you have your data, you have to start annotating. In order to annotate these images, you will need to use LabelImg. LabelImg makes the annotations in a way that the neural network can understand it (a.k.a. makes xml files). You can find LabelImg here.

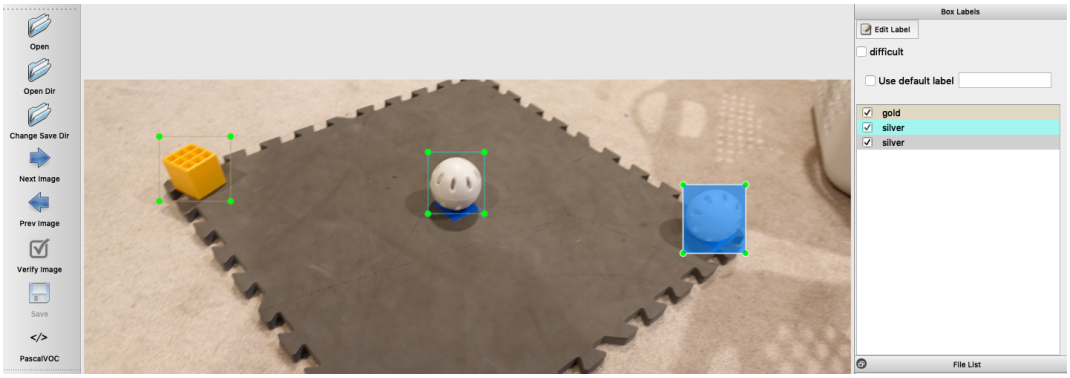


FIGURE 33. Sample screencap of labelling the images.

3.7. **OpenCV**