

FTC Programming Guide

Wen Plotnick

Tanay Kane

Robert Aburustum

March 13, 2019

Dwight-Englewood Robotics: 207 Critical Mass, 13048 Absolute Zero

Contents

I	Hardware and Control System	3
1	REV Expansion Hub	3
2	Phones	3
3	Gamepad	3
4	Motors	3
5	Servos	4
6	Sensors	4
7	Wheels	4
8	Wiring	4
II	Programming in FTC	5
9	Blocks	5
10	Java	5
10.1	OnBot Java	5
10.2	Android Studio	5
III	Vision	6
11	What is it	6

12 OK but why do I need it	6
13 Vuforia	6
14 PixyCam	6
14.1 TL;DR	7
15 Tensorflow	7
16 DIY Neural Network	8
17 OpenCV	8
17.1 Overview	8
17.2 GRIP	9
17.3 Operations	9
17.3.1 Image Resizing	9
17.3.2 Color Converting	9
17.3.3 Blurring	10
17.3.4 Erosion	11
17.3.5 Dilation	11
17.3.6 Color Thresholding	12
17.3.7 Finding Contours	12
17.3.8 Finding Blobs	13
17.4 Example Pipelines	13
17.4.1 Gold Mineral Location Detection for Sampling in Rover Ruckus	13
17.4.2 Silver Mineral Location Detection in the Crater in Rover Ruckus	16
17.4.3 Glyph Detection in the Pit for Relic Recovery	17
17.5 Programming OpenCV	17
17.5.1 DogeCV	17
17.5.2 EnderCV	17
17.5.3 Autogenerated GRIP Code	17
17.5.4 Example Implementation	17

Part I

Hardware and 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 REV Expansion Hub



Figure 1.1: 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.

2 Phones

3 Gamepad

4 Motors

If you're building a robot, you can't escape motors. Most likely you will be using the AndyMark NeverRest motors for the competition as they are the most effective. There are 3 main motors in this series: 60s, 40s and 20s.



Figure 4.1: This is a picture of the NeverRest 40. If people can organize the room, it should be in a bin. Otherwise it's on the floor.

5 Servos

6 Sensors

7 Wheels

8 Wiring

Part II

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.

9 Blocks

10 Java

10.1 OnBot Java

10.2 Android Studio

Part III

Vision

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

11 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.

12 OK but why do I need it

13 Vuforia

14 PixyCam



Figure 14.1: 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.

14.1 TL;DR

```

10:35 AM chenelli hey @c++ > c how is using pixycam (edited)
chenelli for robotics
10:36 AM chenelli like what is the quality of the image as opposed to the phone
11:08 AM c++ > c pixycam [REDACTED] sucks
c++ > c just use opencv + phone camera
c++ > c much more flexible
c++ > c pixycam is a glorified color sensor

```

15 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.

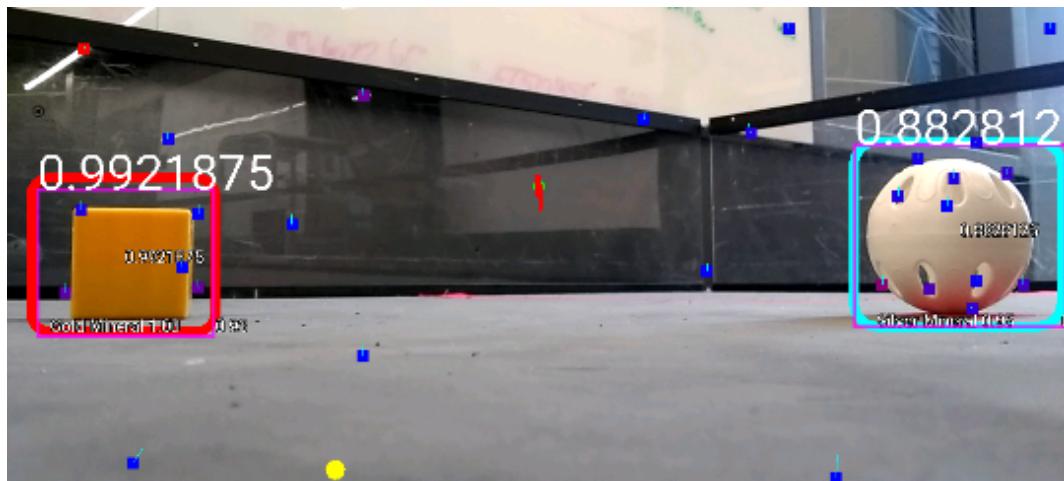


Figure 15.1: Sample output of the Tensorflow. Each object that it believes to be a trained object, in this case the minerals, is boxed. Next to each box is a number, representing how confident the neural network is that the box is a mineral. In the FTC SDK, methods for getting the location of the box, 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.

16 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 put an little box around it.

The first part of this method is to 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.

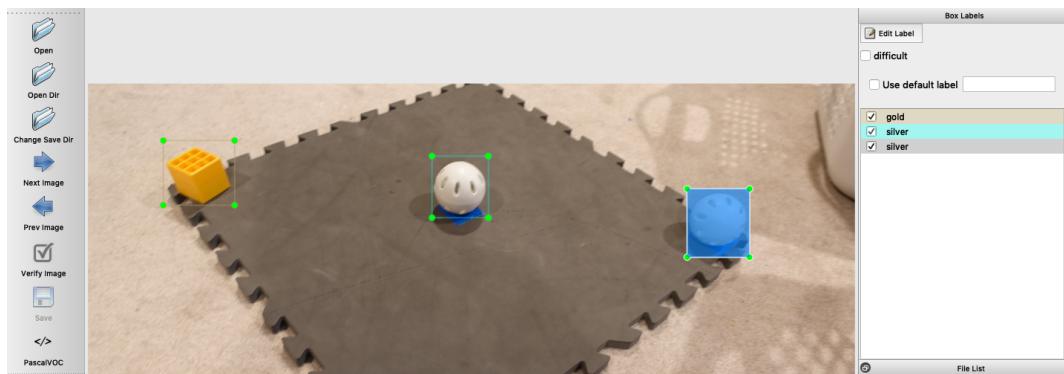


Figure 16.1: Sample screencap of labelling the images.

17 OpenCV

17.1 Overview

Computer Vision is the field of getting data from images that a computer can use. While the aforementioned TensorFlow does count in this regard as Computer Vision, typically Computer Vision refers to more direct algorithms - not trained neural nets. OpenCV is a collection of the various algorithms people have developed for various languages, including java. OpenCV allows for a much more flexible method of detecting and classifying objects in an FTC game.

The algorithms, or operations, provided by OpenCV can be chained together into a pipeline. To this pipeline an image is fed in, and out some information about that image is returned. Thus, when developing an OpenCV pipeline, one will first want to experiment with different operations to perform the desired action.

17.2 GRIP

Provided by WPI and found at <https://github.com/WPIRoboticsProjects/GRIP/>, GRIP is an extremely useful program for creating an OpenCV pipeline¹.

17.3 Operations

17.3.1 Image Resizing

This operation is pretty self explanatory. The input image from the phone is resized. There are a bunch of methods of interpolation - how the computer knows what colors to make the pixels in the new image - but you can keep this on the default, cubic, since it really doesn't have much of an effect². The importance of this step is to decrease the size of the image, allowing following operations to run faster - if there is less image to look at, less time is needed.



Figure 17.1: Resized picture of Crawley dabbing in GRIP

17.3.2 Color Converting

make sure to mention caveat about the HSV on phones

¹A short tutorial on how to use GRIP can be found at <https://wpilib.screenstepslive.com/s/currentCS/m/vision/1/463566-introduction-to-grip>

²You typically end up messing up the image with a blur, erosion, or dilation, so any work the phone does to interpolate data will be destroyed.

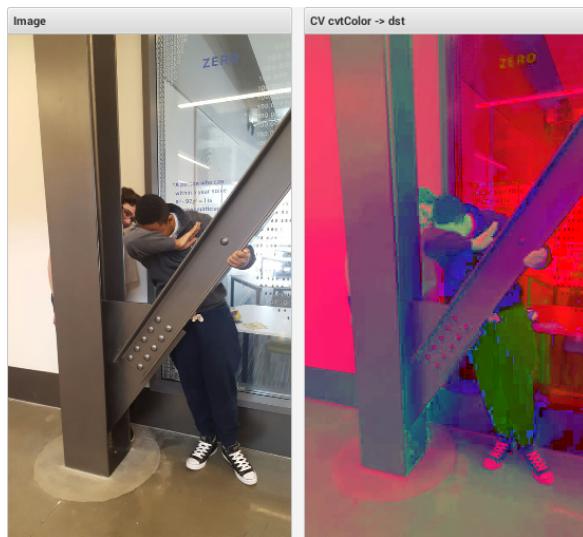


Figure 17.2: RGB Image of Crawley converted to HSV color space in GRIP

17.3.3 Blurring

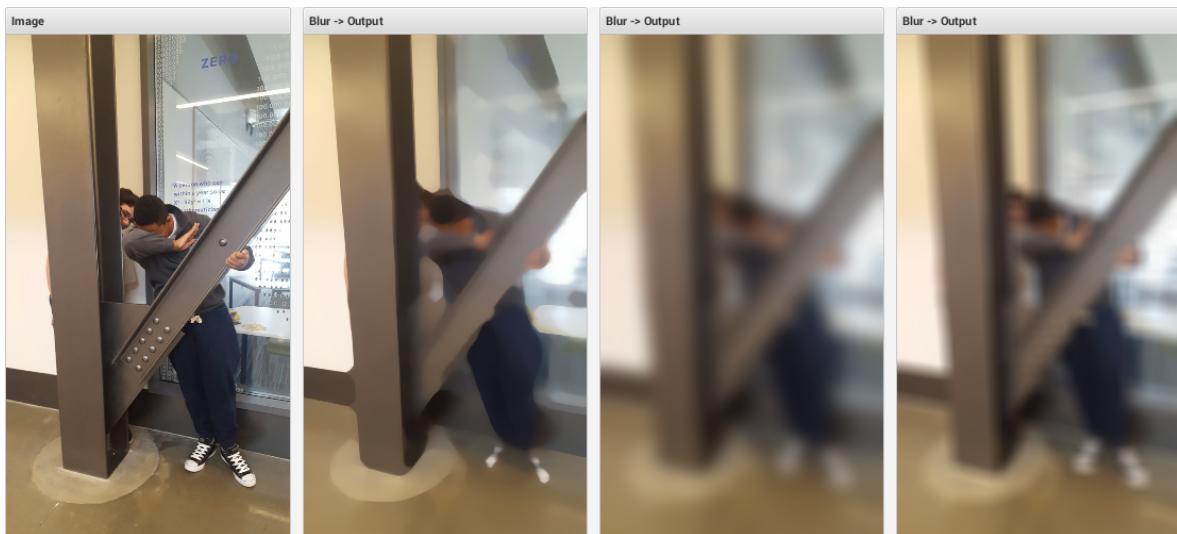


Figure 17.3: 3 types of blurs (Median, Gaussian, Box, in that order) on Crawley

17.3.4 Erosion

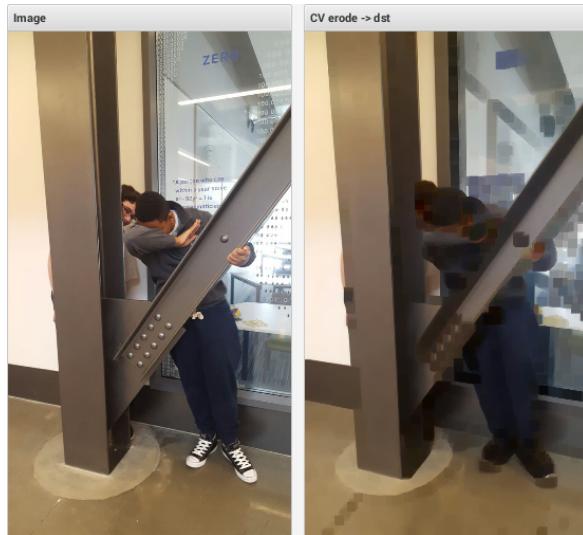


Figure 17.4: Heavily eroded image of Crawley

17.3.5 Dilation

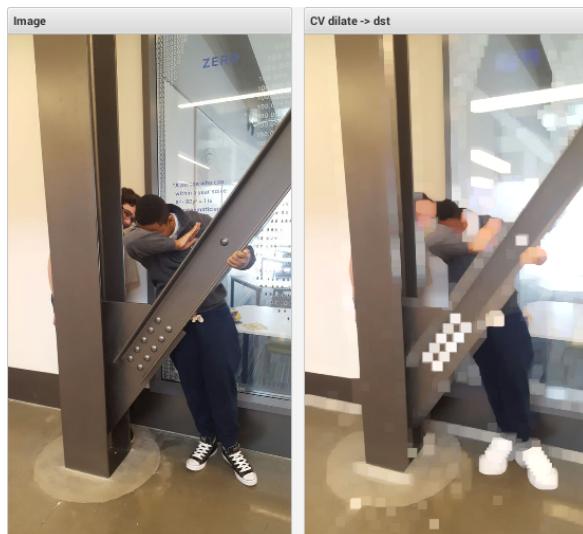


Figure 17.5: Heavily dilated image of Crawley

17.3.6 Color Thresholding

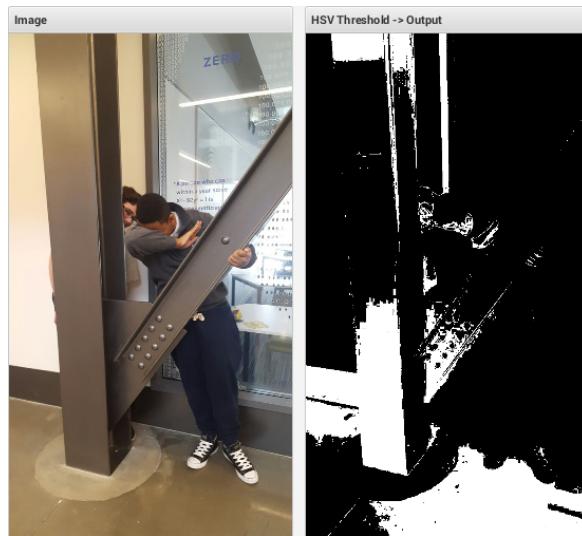


Figure 17.6: Meaningless thresholding of image of Crawley

17.3.7 Finding Contours

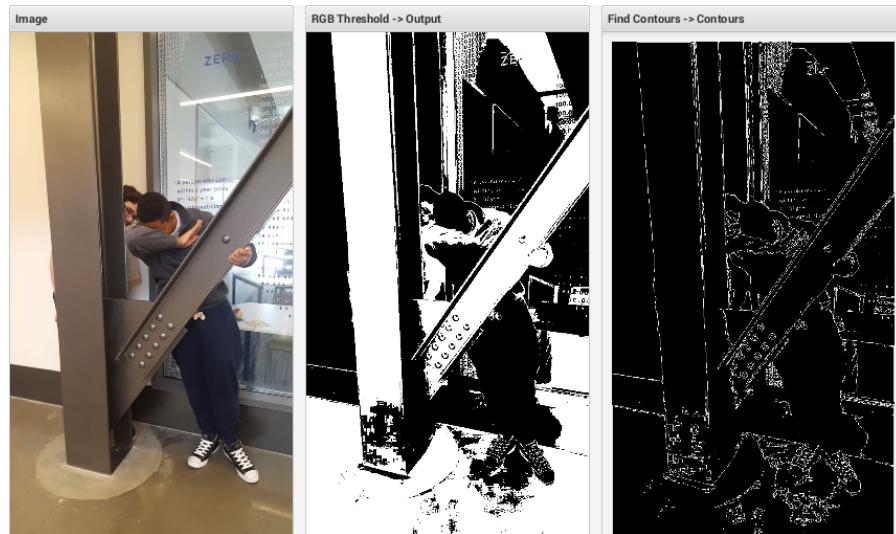


Figure 17.7: Various contours in another pretty meaningless thresholding of the image of Crawley

17.3.8 Finding Blobs

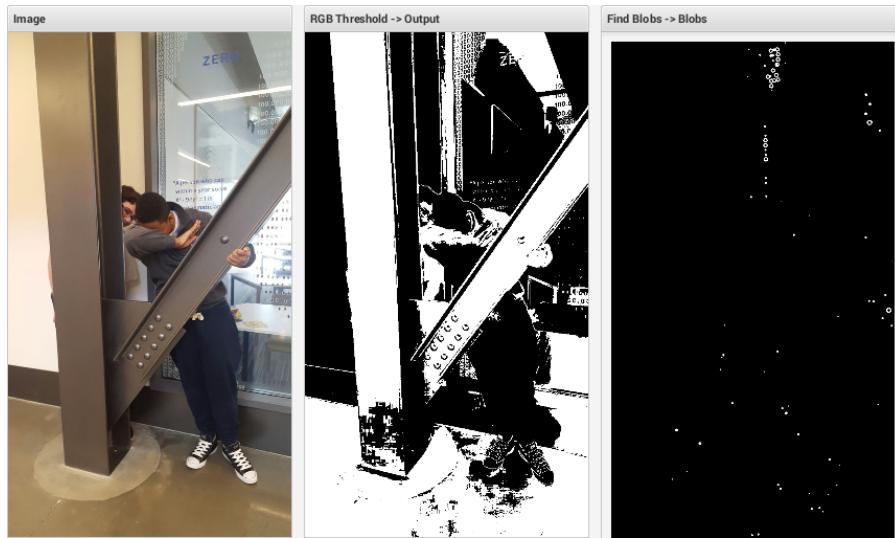


Figure 17.8: Failed attempt at finding blobs in an image that is not properly thresholded for finding blobs

17.4 Example Pipelines

17.4.1 Gold Mineral Location Detection for Sampling in Rover Ruckus

This pipeline is the (nearly) exact pipeline 207 used at NJ states in the Rover Ruckus season, with much success. First, a demonstration of the final product.

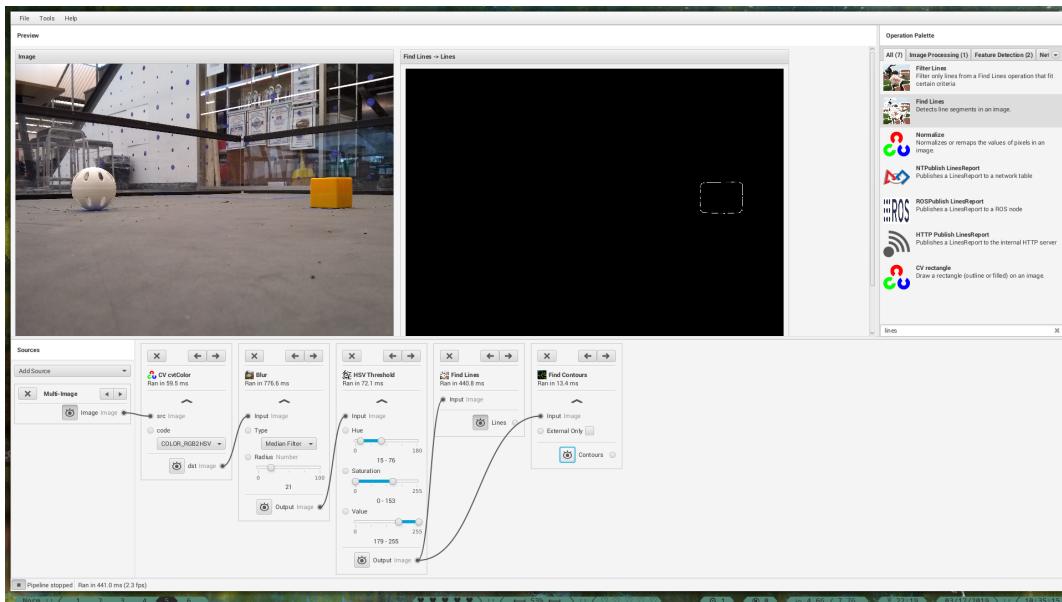


Figure 17.9: The white box is the location of the gold mineral the code can use. This is using a line detection algorithm, as the result from that is easier to see - contour gives a very faint line. The identified location is exactly the same, however. I have also forgone the resizing step since the images are then too small to see clearly.

So, having shown that it works for at least one case, let's go over how the pipeline was made and works. The first (omitted in the example) step is to resize the image. We do this because we do not need a whole lot of detail in our detection - we only really care about the two big mineral pixel areas. Thus, by resizing, we can speed up our identification, with no detriment to the algorithm efficacy. The next step is to convert the image color space to HSV.

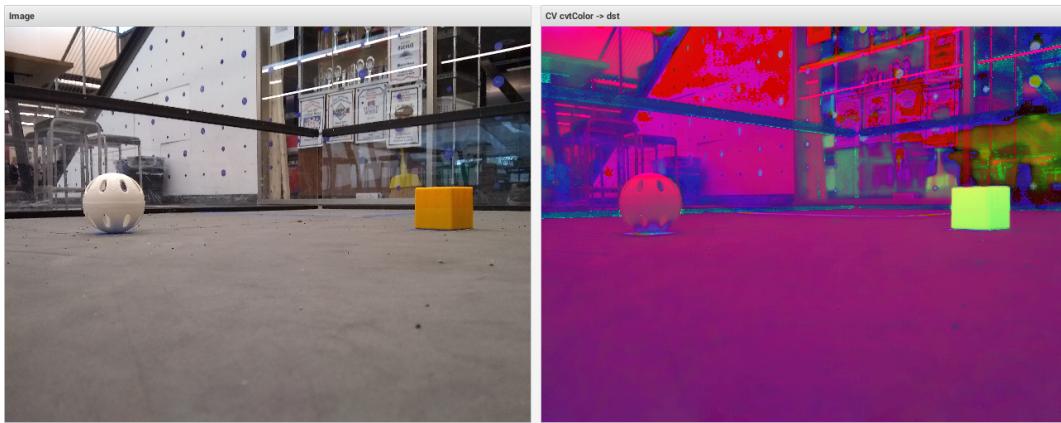


Figure 17.10: Comparison of RGB to HSV image

This is a fairly standard step, especially useful for the gold mineral as can be seen in the neon green color. The next step that we are going to perform is a blur. We do this to ensure that the edges are smoothed out.

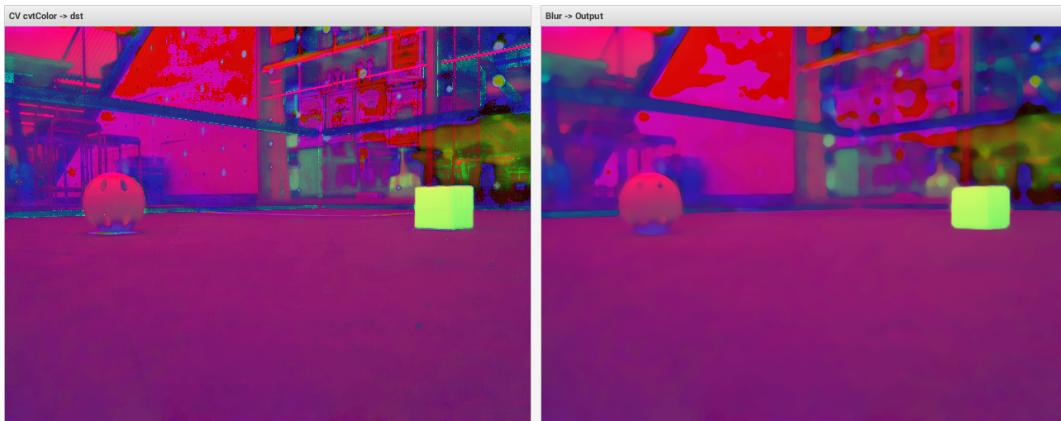


Figure 17.11: Comparison of Pre-Blur to Post-Blur image

The next step is the “meat and potatoes” of the algorithm. Experimentally, we determine the range in which the color values of the gold mineral lay. Once this range is found, we perform a threshold on the image, selecting pixels which lie in that range exactly.

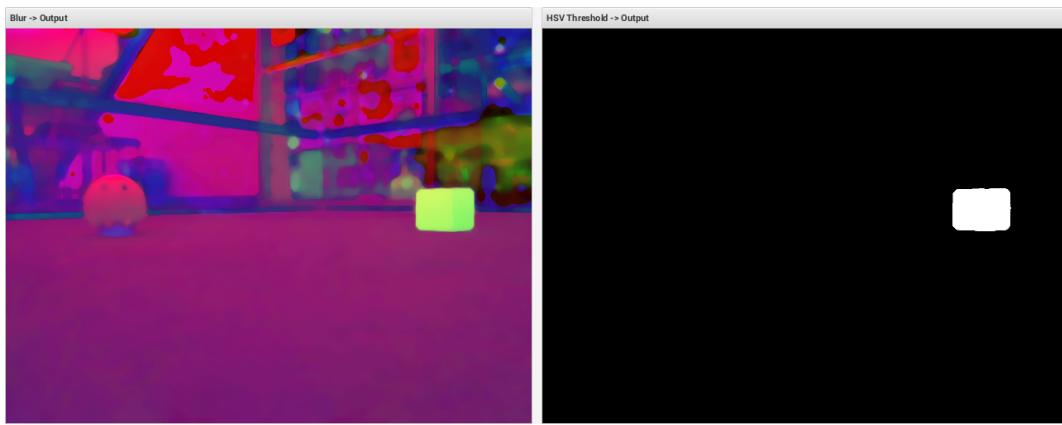


Figure 17.12: Comparison of Source to thresholded image

Once we have that binary image, we can run either contour detection, or blob detection. We personally recommend contours - they are faster, as well as more robust due to the tendency of blobs to detect only blobbish objects. In this specific image, both work. However, other images in testing did not work with blobs.

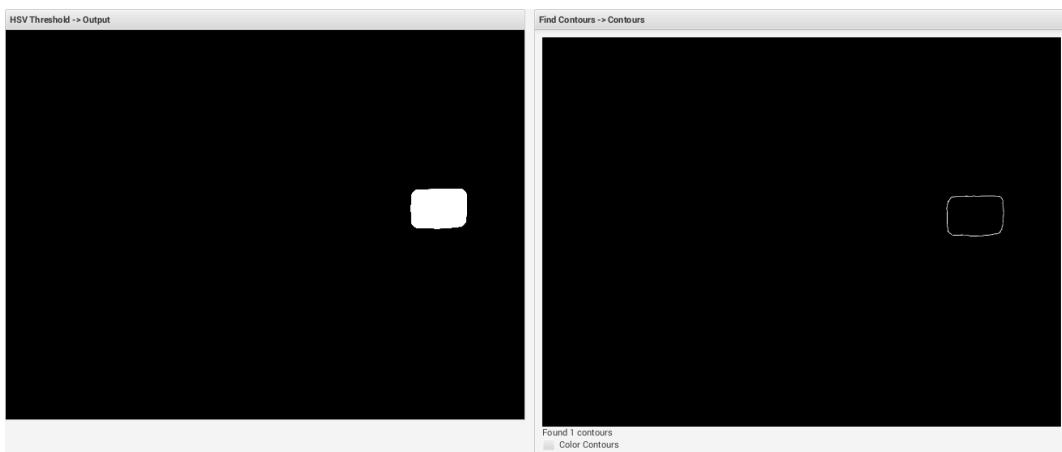


Figure 17.13: Comparison of Source to Identified Contours. The contour line has been drawn over so you can actually see it

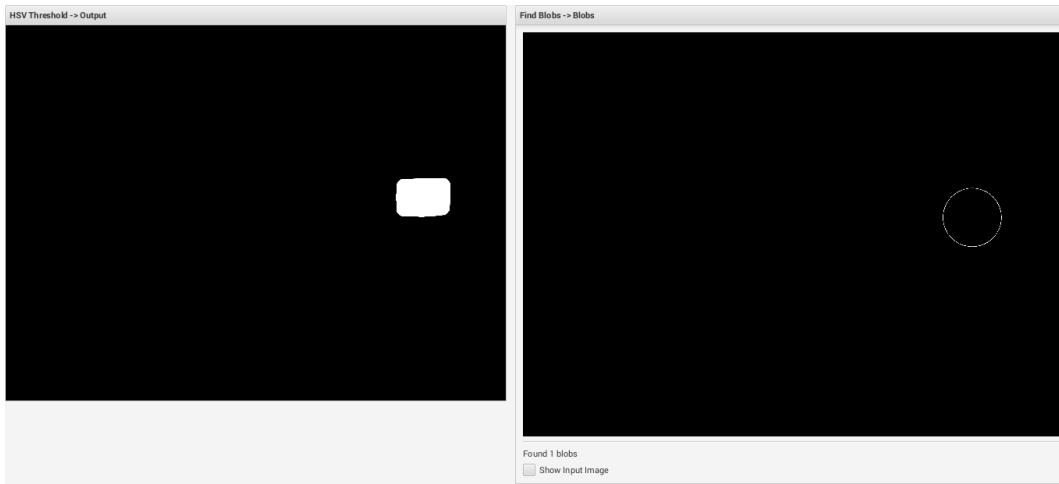


Figure 17.14: Comparison of Source to Identified Blobs

The pipeline described above can work for a wide variety of objects. The jewels in Relic Recovery could be found in a very similar way. This pipeline is useful for identifying a small object from an image, with a uniform color across the object. When developing these types of pipelines, blurring and thresholding are very easy to mess around with to increase the efficacy of the algorithm.

17.4.2 Silver Mineral Location Detection in the Crater in Rover Ruckus

In this section, we will explore a OpenCV pipeline that can identify the location of silver minerals in the crater. The important note here is that the minerals are very close together, which is different from the sampling pipeline.

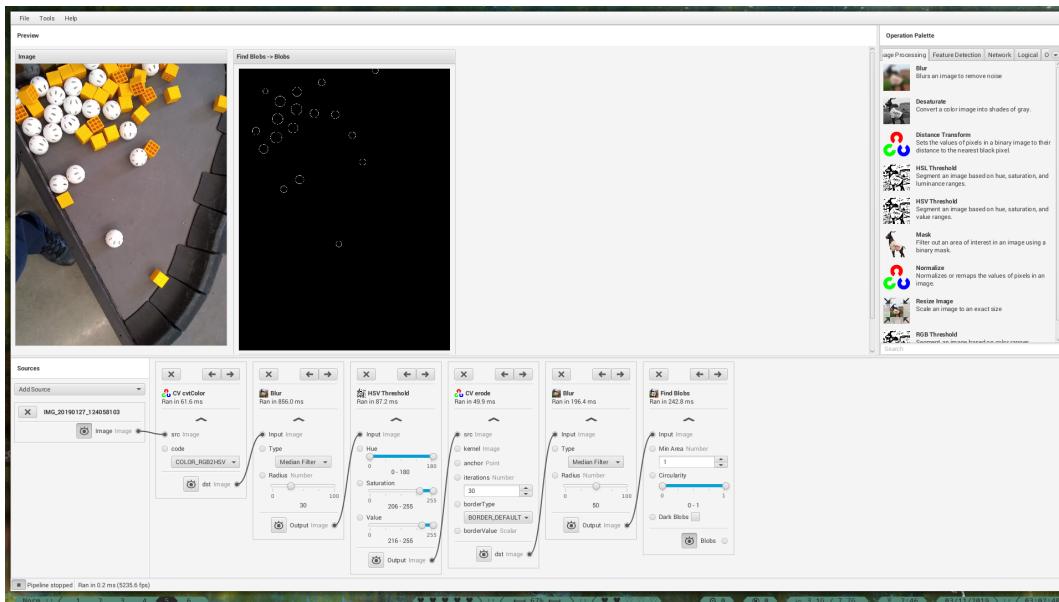


Figure 17.15: Screenshot of the finalized pipeline, and the identified locations of the silver mineral

17.4.3 Glyph Detection in the Pit for Relic Recovery

17.5 Programming OpenCV

17.5.1 DogeCV

17.5.2 EnderCV

17.5.3 Autogenerated GRIP Code

17.5.4 Example Implementation