FTC Programming Guide

Wen Plotnick Tanay Kane Robert Aburustum

May 28, 2019

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

Contents

Part I
Preface

The goal of this programming guide is for the reader to have a somewhat okay ability to program for an FTC team. However, we are mere mortals, so there are some caveats.

The first is that we assume reasonable knowledge of Java. You do not need to be an expert, but if you are completely new a lot of this might go over your head.

The second is that a lot of FTC programming, and FTC skills in general, come from experience. You can read as many guides as you want, but some knowledge you will need to actually sit down and program for (this is especially true for debugging)

Finally, our teams, in comparison to the upper echelons of FTC teams, did not do anything too special. Sure, we used sensors, encoders, a little bit of PID. But we never really explored the limits of PID, motion profiling, advanced control theory stuff.

When you look at teams like Gluten Free, with their crazy 6 glyph auto in Relic Recovery, and their 4.5 auto cycle auton in Rover Ruckus, you see something that this guide won't be able to teach you. But to get to that level, you need the basics - which this guide can teach.

Part II
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

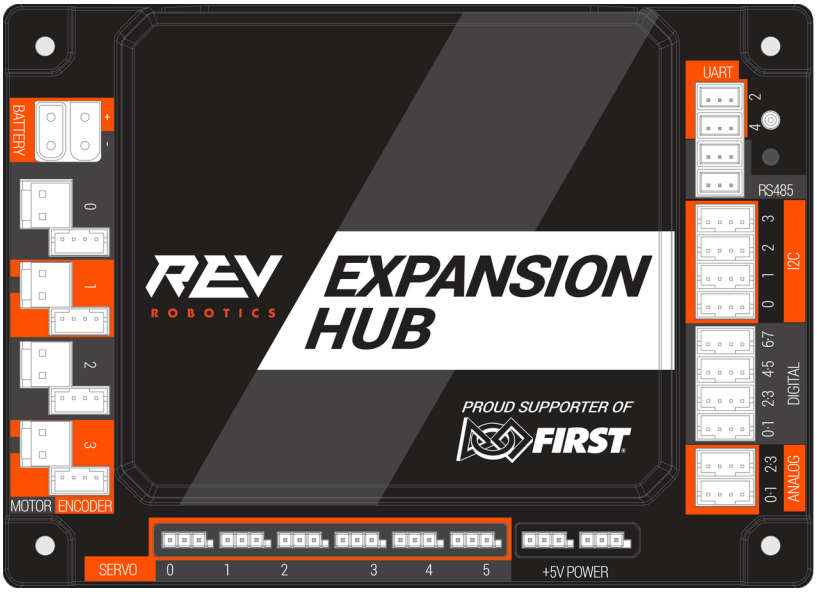


Figure 1.1: Image of the REV Expansion Hub. We will go over the specifics of what each port is for in this section.

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 III
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 IV
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

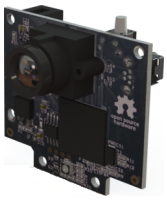


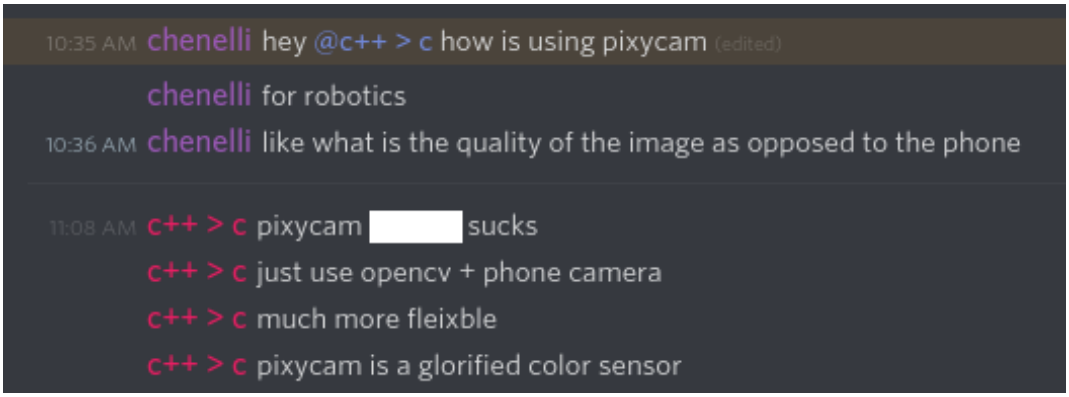
Figure 14.1: A PixyCam, so you know what it looks like.

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



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.

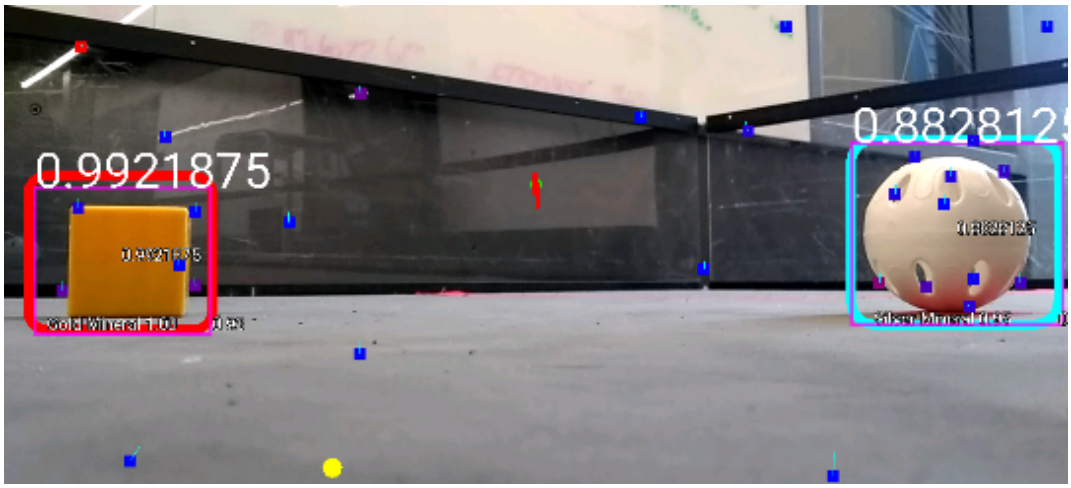


Figure 15.1: Sample output of the Tensorflow. Each object that it believes to be a trained object, in this case the minerals, is boxed. Next to each box is a number, representing how confident the neural network is that the box is a mineral. In the FTC SDK, methods for getting the location of the box,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

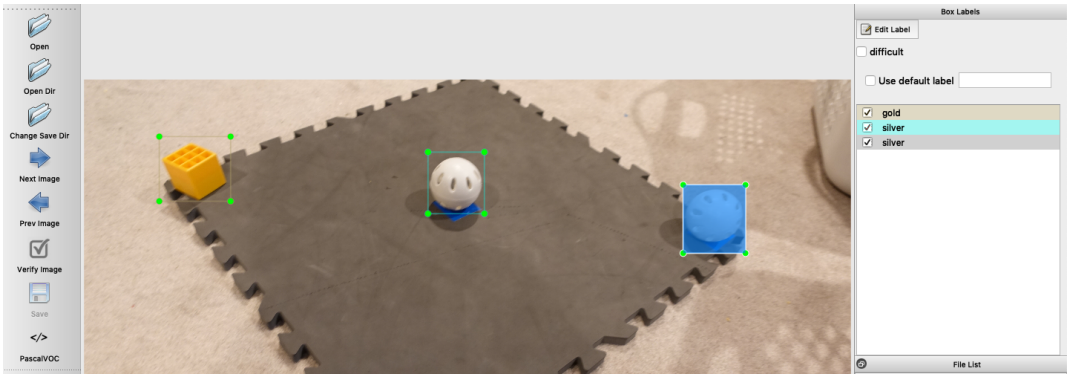


Figure 16.1: Sample screencap of labelling the images.

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 pipeline[1].

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 effect[2]. The importance of this step is to decrease the size of the image, allowing following operations to run must faster - if there is less image to look at, less time is needed.

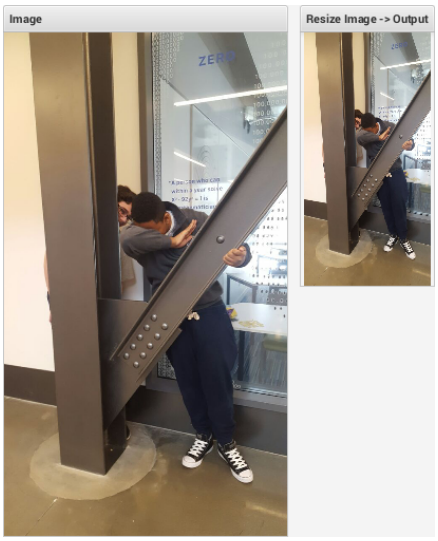


Figure 17.1: Resized picture of Crawley dabbing in GRIP

17.3.2 Color Converting

make sure to mention caveat about the HSV on phones

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

[2] You 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.

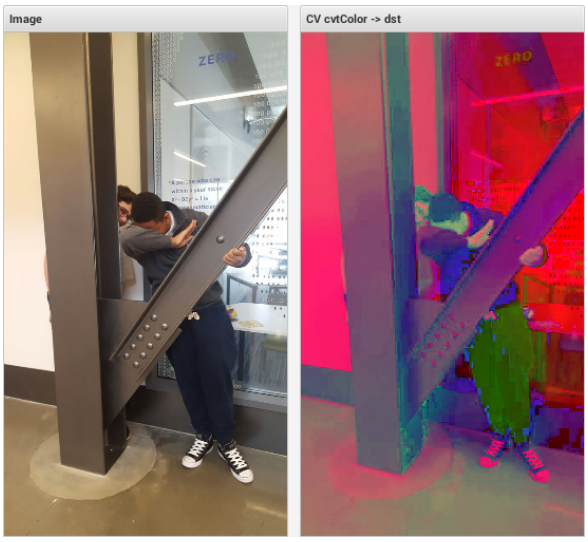


Figure 17.2: RGB Image of Crawley converted to HSV color space in GRIP

17.3.3 Blurring

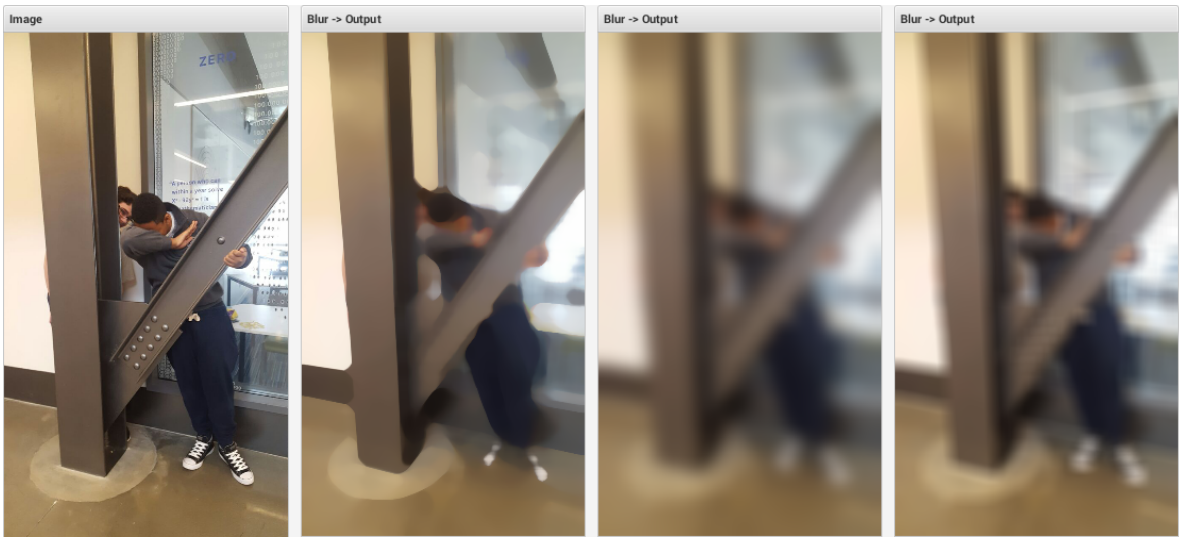


Figure 17.3: 3 types of blurs (Median, Gaussian, Box, in that order) on Crawley

17.3.4 Erosion

Figure 17.4: Heavily eroded image of Crawley

17.3.5 Dilation

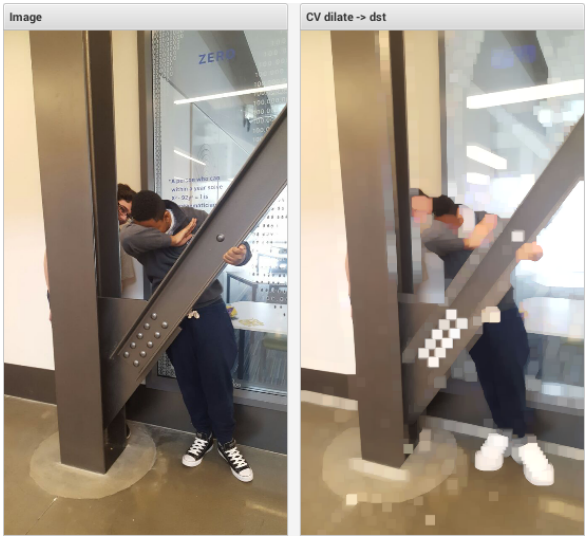


Figure 17.5: Heavily dilated image of Crawley

17.3.6 Color Thresholding

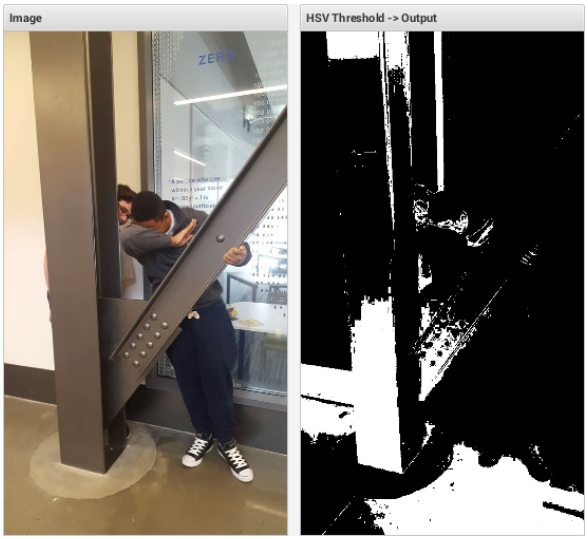


Figure 17.6: Meaningless thresholding of image of Crawley

17.3.7 Finding Contours

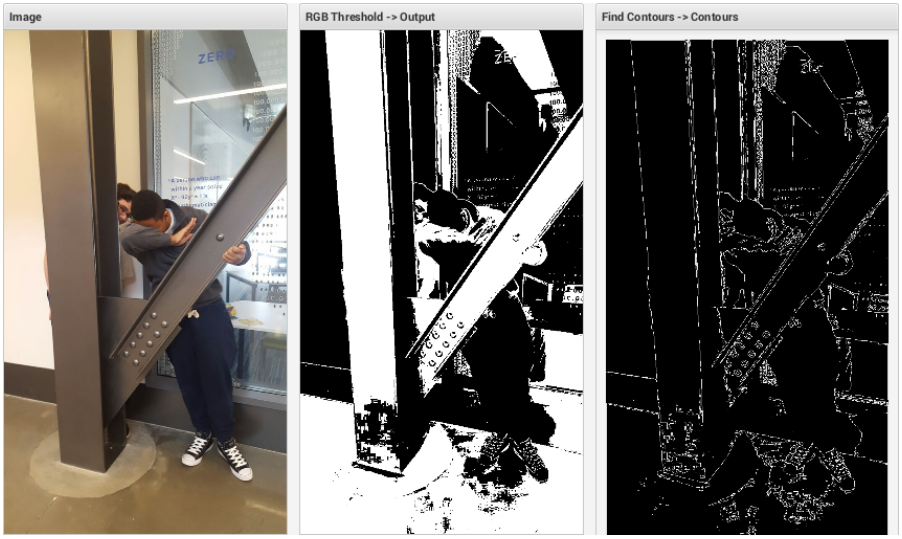


Figure 17.7: Various contours in another pretty meaningless thresholding of the image of Crawley

17.3.8 Finding Blobs

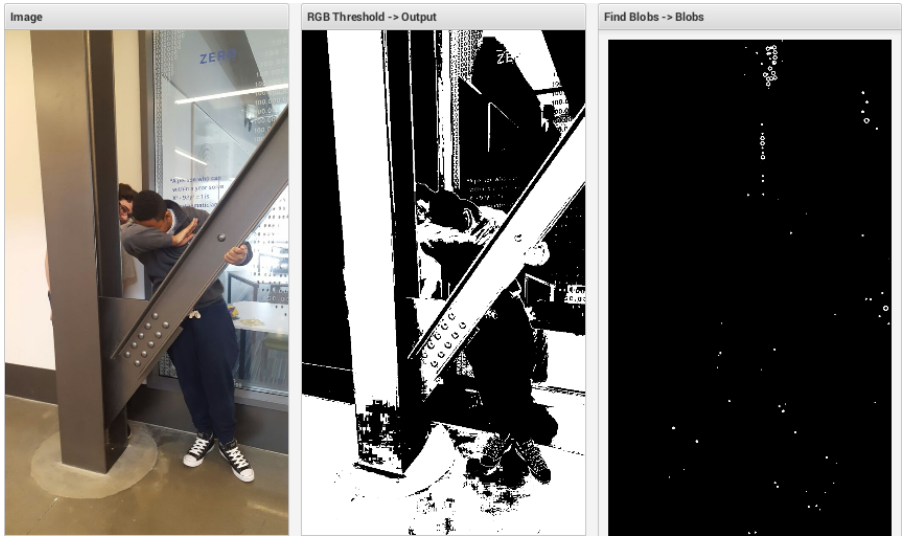


Figure 17.8: Failed attempt at finding blobs in an image that is not properly thresholded for finding blobs

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.

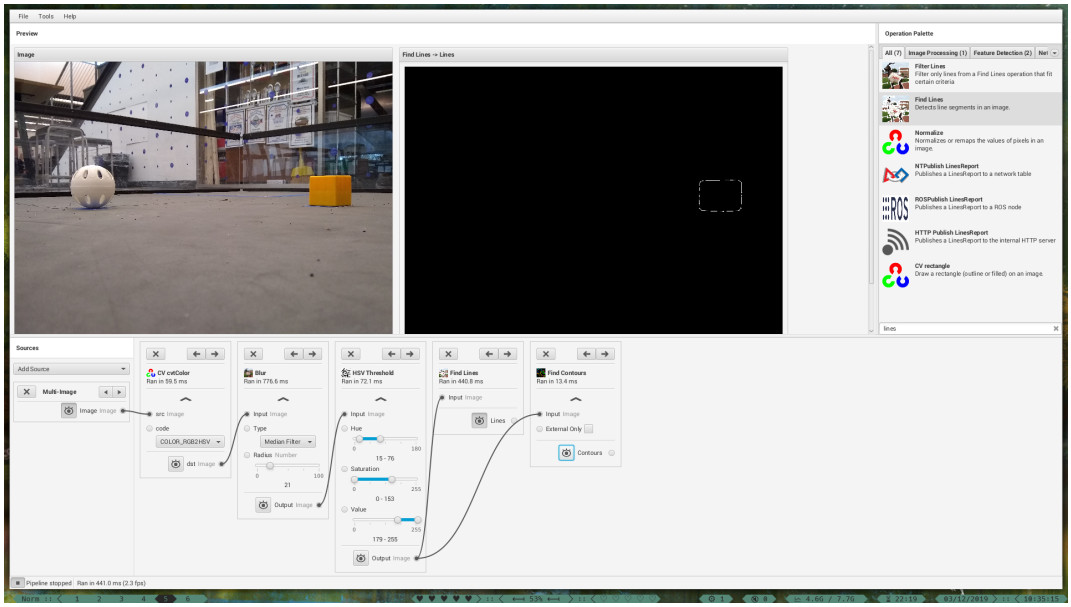


Figure 17.9: The white box is the location of the gold mineral the code can use. This is using a line detection algorithm, as the result from that is easier to see - contour gives a very faint line. The identified location is exactly the same, however. I have also forgone the resizing step since the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

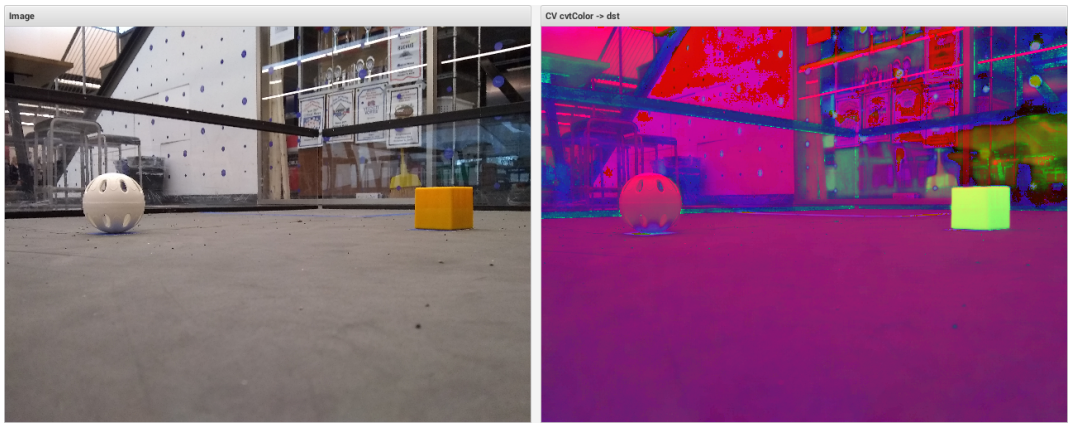


Figure 17.10: Comparison of RGB to HSV image

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.

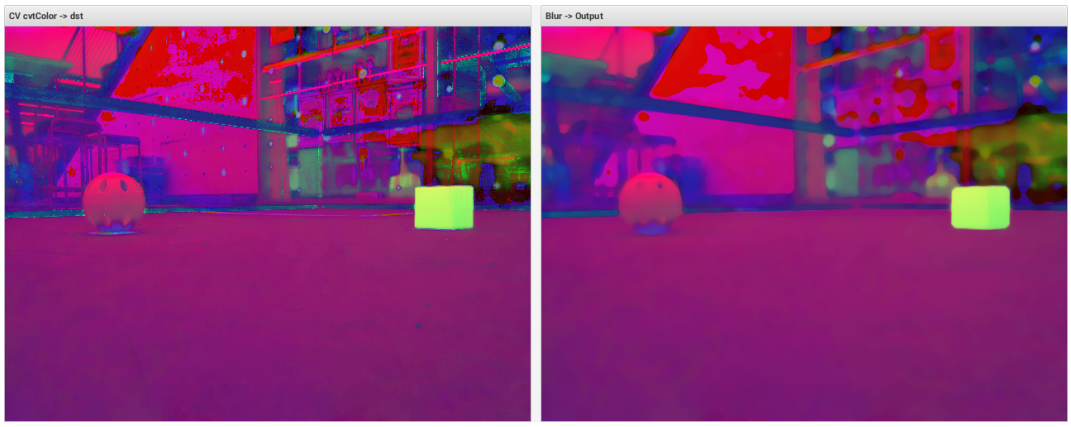


Figure 17.11: Comparison of Pre-Blur to Post-Blur image

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.

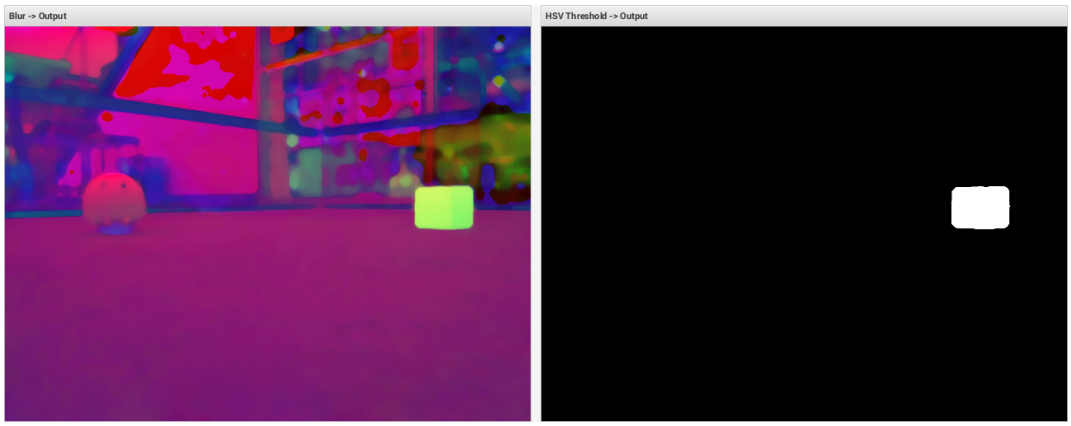


Figure 17.12: Comparison of Source to thresholded image

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 dude 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.

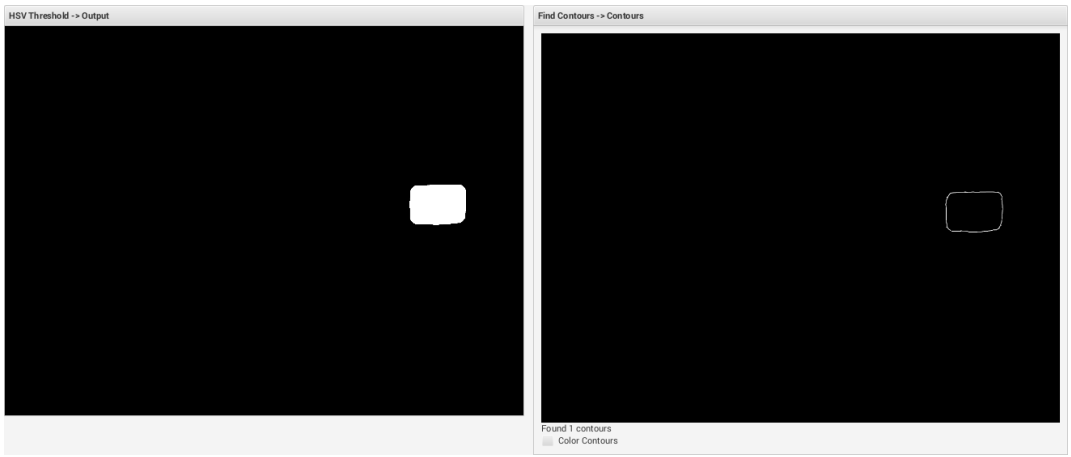


Figure 17.13: Comparison of Source to Identified Contours. The contour line has been drawn over so you can actually see it

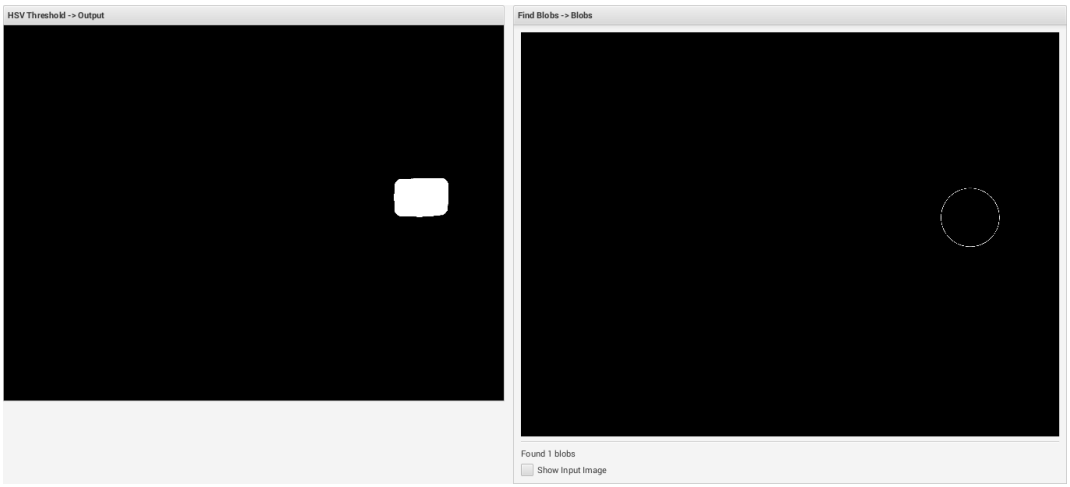


Figure 17.14: Comparison of Source to Identified Blobs

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. Blurring allows you to smooth the edges of the object detected, as well as reduce the chance of detecting noise from the other areas of the image.

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. This means we perform an extra step or so, in order to make sure we identify these separately.

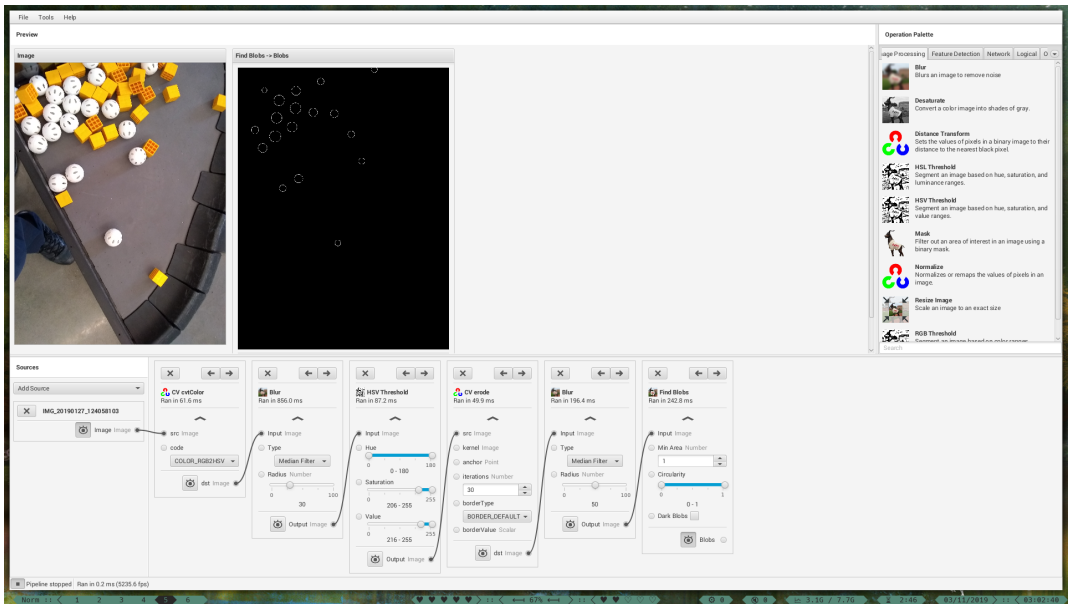


Figure 17.15: Screenshot of the finalized pipeline, and the identified locations of the silver mineral

In our pipeline, we have chosen to not resize the image, mainly because it makes for a nicer looking image. Additionally, the use case for this - detecting where to extend into the crater - is not super time dependent, so if we end up taking 2 seconds to analyze a frame, it is ok.

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

The first class, `GoldDetectorPipeline`, contains the (slightly) modified GRIP generated code. The modifications made are the ones that make the actual code compile (removing of the final line thing) as well as the moving of constants for thresholds, radii, etc.

```
1 public class GoldDetectorPipeline {
2
3     public static final double RESIZE_IMAGE_WIDTH = 320.0;
4     public static final double RESIZE_IMAGE_HEIGHT = 240.0;
5
6     public static final double BLUR_RADIUS = 8.108108108108109;
7
```

```
8       public static final int hsvHueLow = 25;
9       public static final int hsvHueHigh = 55;
10
11      public static final int hsvSatLow = 150;
12      public static final int hsvSatHigh = 255;
13
14      public static final int hsvValLow = 100;
15      public static final int hsvValHigh = 255;
16      //Outputs
17      private Mat resizeImageOutput = new Mat();
18      private Mat cvCvtcolorOutput = new Mat();
19      private Mat blurOutput = new Mat();
20      private Mat hsvThresholdOutput = new Mat();
21      private ArrayList<MatOfPoint> findContoursOutput = new ArrayList<MatOfPoint>();
22
23      //Given a Mat, or image from the phone camerbbbbbbba,
24      public void process(Mat source0) {
25          // Step 1: Resize_Image0:
26          Mat resizeImageInput = source0; // Create a copy of the source imgae
27          int resizeImageInterpolation = Imgproc.INTER_CUBIC; // Set the interpolation
            ↪ algorithm
28          resizeImage(resizeImageInput, RESIZE_IMAGE_WIDTH, RESIZE_IMAGE_HEIGHT,
            ↪ resizeImageInterpolation, resizeImageOutput); // Perform the image resize
29
30          // Step 2: Convert RBG to HSV Color Format
31          Mat cvCvtcolorSrc = resizeImageOutput; //Copy the resized image
32          int cvCvtcolorCode = Imgproc.COLOR_RGB2HSV; // Set the Color Format
            ↪ transformation we will do
33          cvCvtcolor(cvCvtcolorSrc, cvCvtcolorCode, cvCvtcolorOutput); // Convert the Color
            ↪ Format
34
35          // Step 3: Blur the Image
36          Mat blurInput = cvCvtcolorOutput; // Copy the converted image
37          BlurType blurType = BlurType.get("Median Filter"); // Tell the type of blur we
            ↪ will use
38          blur(blurInput, blurType, BLUR_RADIUS, blurOutput); // Blur the image
39
40          // Step 4: HSV Threshold the Image
41          Mat hsvThresholdInput = blurOutput; // Copy the blurred image
42          double[] hsvThresholdHue = {hsvHueLow, hsvHueHigh}; // Set the Hue range we will
            ↪ accept pixels for
43          double[] hsvThresholdSaturation = {hsvSatLow, hsvSatHigh}; // Set the Saturation
            ↪ range we will accept pixels for
44          double[] hsvThresholdValue = {hsvValLow, hsvValHigh}; // Set the Value range we
            ↪ will accept pixels for
```

```
45          hsvThreshold(hsvThresholdInput, hsvThresholdHue, hsvThresholdSaturation,
            ↪  hsvThresholdValue, hsvThresholdOutput); // Threshold the image for the
            ↪  specified colors
46
47          // Step 5: Find the Contours
48          Mat findContoursInput = hsvThresholdOutput; // Copy the thresholded image
49          boolean findContoursExternalOnly = false; //Tell it to find all contours it
            ↪  identifies
50          findContours(findContoursInput, findContoursExternalOnly, findContoursOutput); //
            ↪  Find Contours in the image
51      }
52
53      public Mat resizeImageOutput() {
54          return resizeImageOutput;
55      }
56
57      public Mat cvCvtcolorOutput() {
58          return cvCvtcolorOutput;
59      }
60
61      public Mat blurOutput() {
62          return blurOutput;
63      }
64
65      public Mat hsvThresholdOutput() {
66          return hsvThresholdOutput;
67      }
68
69      public ArrayList<MatOfPoint> findContoursOutput() {
70          return findContoursOutput;
71      }
72
73      private void resizeImage(Mat input, double width, double height,
74                               int interpolation, Mat output) {
75          Imgproc.resize(input, output, new Size(width, height), 0.0, 0.0, interpolation);
76      }
77
78      private void cvCvtcolor(Mat src, int code, Mat dst) {
79          Imgproc.cvtColor(src, dst, code);
80      }
81
82      private void blur(Mat input, BlurType type, double doubleRadius,
83                        Mat output) {
84          int radius = (int) (doubleRadius + 0.5);
85          int kernelSize;
```

```
86          switch (type) {
87              case BOX:
88                  kernelSize = 2 * radius + 1;
89                  Imgproc.blur(input, output, new Size(kernelSize, kernelSize));
90                  break;
91              case GAUSSIAN:
92                  kernelSize = 6 * radius + 1;
93                  Imgproc.GaussianBlur(input, output, new Size(kernelSize, kernelSize),
                    ↪  radius);
94                  break;
95              case MEDIAN:
96                  kernelSize = 2 * radius + 1;
97                  Imgproc.medianBlur(input, output, kernelSize);
98                  break;
99              case BILATERAL:
100                 Imgproc.bilateralFilter(input, output, -1, radius, radius);
101                 break;
102         }
103     }
104
105     private void hsvThreshold(Mat input, double[] hue, double[] sat, double[] val,
106                               Mat out) {
107         Imgproc.cvtColor(input, out, Imgproc.COLOR_BGR2HSV);
108         Core.inRange(out, new Scalar(hue[0], sat[0], val[0]),
109                 new Scalar(hue[1], sat[1], val[1]), out);
110     }
111
112     private void findContours(Mat input, boolean externalOnly,
113                               List<MatOfPoint> contours) {
114         Mat hierarchy = new Mat();
115         contours.clear();
116         int mode;
117         if (externalOnly) {
118             mode = Imgproc.RETR_EXTERNAL;
119         } else {
120             mode = Imgproc.RETR_LIST;
121         }
122         int method = Imgproc.CHAIN_APPROX_SIMPLE;
123         Imgproc.findContours(input, contours, hierarchy, mode, method);
124     }
125
126     enum
        ↪  BlurType {
127         BOX("Box Blur"), GAUSSIAN("Gaussian Blur"), MEDIAN("Median Filter"),
128         BILATERAL("Bilateral Filter");
```

```
129
130         private final String label;
131
132         BlurType(String label) {
133             this.label = label;
134         }
135
136         public static BlurType get(String type) {
137             if (BILATERAL.label.equals(type)) {
138                 return BILATERAL;
139             } else if (GAUSSIAN.label.equals(type)) {
140                 return GAUSSIAN;
141             } else if (MEDIAN.label.equals(type)) {
142                 return MEDIAN;
143             } else {
144                 return BOX;
145             }
146         }
147
148         @Override
149         public String toString() {
150             return this.label;
151         }
152     }
153
154
155 }
```

the seccond class, `GoldDetectorWrapper`, makes the OpenCVPipeline available from the robot class. This requires the installation of EnderCV to use.

```
1  public class GoldDetectorWrapper extends OpenCVPipeline implements Subsystem {
2
3      public static final int minContourArea = 1000;
4      public static final int maxContourArea = 5000;
5      public ImageView imageView = ImageView.THRESH;
6      public GoldDetectorPipeline grip = new GoldDetectorPipeline();
7      // this is just here so we can expose it later thru getContours.
8      public List<MatOfPoint> contours = new ArrayList<>();
9      // To keep it such that we don't have to instantiate a new Mat every call to
       ↪  processFrame,
10     // we declare the Mats upSafe here and reuse them. This is easier on the garbage
       ↪  collector.
11     private Mat hsv = new Mat();
```

```
12      private Mat thresholded = new Mat();
13      private MineralPosition currentState;
14
15      public synchronized List<MatOfPoint> getContours() {
16          return contours;
17      }
18
19      // This is called every camera frame.
20      @Override
21      public Mat processFrame(Mat rgba, Mat gray) {
22          grip.process(rgba);
23          contours = grip.findContoursOutput();
24          switch (imageView) {
25              case RESIZE:
26                  return grip.resizeImageOutput();
27              case HSV:
28                  return grip.cvCvtcolorOutput();
29              case BLUR:
30                  return grip.blurOutput();
31              case THRESH:
32                  return grip.hsvThresholdOutput();
33              case CONTOUR:
34                  Mat contourImageOutput = grip.resizeImageOutput().clone();
35                  Imgproc.drawContours(contourImageOutput, grip.findContoursOutput(), -1,
                    ↪  new Scalar(255, 255, 255), 8);
36                  return contourImageOutput;
37              default:
38                  return rgba;
39          }
40
41      }
42
43      @Override
44      public void init(HardwareMap hwMap) {
45          this.init(hwMap.appContext, CameraViewDisplay.getInstance());
46      }
47
48      @Override
49      public void start() {
50          this.enable();
51      }
52
53      @Override
54      public void reset() {
55
```

```
56      }
57
58      @Override
59      public void stop() {
60          this.disable();
61
62      }
63
64      private void updateState() {
65          try {
66          List<MatOfPoint> contours = this.getContours();
67          for (int i = 0; i < contours.size(); i++) {
68              Rect boundingRect = Imgproc.boundingRect(contours.get(i));
69              if (boundingRect.area() < 50) {
70                  contours.remove(i);
71                  i--;
72              }
73          }
74
75
76          if (contours.size() == 0) {
77              this.currentState = MineralPosition.RIGHT;
78          } else if (contours.size() >= 1) {
79              double areaSum = 0;
80              double xAvg = 0;
81              //double yAvg = 0;
82              for (int i = 0; i < contours.size(); i++) {
83                  Rect boundingRect = Imgproc.boundingRect(contours.get(i));
84                  double areaLol = boundingRect.area();
85                  areaSum = areaSum + areaLol;
86                  xAvg = xAvg + areaLol * (boundingRect.x + boundingRect.width / 2d);
87                  //yAvg = yAvg + areaLol*(boundingRect.y + boundingRect.height / 2d);
88              }
89              xAvg = xAvg / areaSum;
90              //yAvg = yAvg / areaSum;
91              //telemetry.addData("x", boundingRect.x + boundingRect.width / 2);
92              //telemetry.addData("y", boundingRect.y + boundingRect.height / 2);
93              if (xAvg > 160) {
94                  this.currentState = MineralPosition.CENTER;
95              } else {
96                  this.currentState = MineralPosition.LEFT;
97              }
98
99          } else {
100             this.currentState = MineralPosition.NOTVISIBLE;
```

```
        }} catch (IndexOutOfBoundsException e) {

        }
    }

    @Override
    public State getState() {
        this.updateState();
        return this.currentState;
    }

    public List<MatOfPoint> filterContours(List<MatOfPoint> contours) {
        for (int i = 0; i < contours.size(); i++) {
            Rect boundingRect = Imgproc.boundingRect(contours.get(0));
            if (!(boundingRect.area() > minContourArea) || !(boundingRect.area() <
            ↪  maxContourArea)) {
                contours.remove(i);
                i--;
            }
        }

        return contours;

    }

    public enum ImageView {
        RESIZE, HSV, BLUR, THRESH, CONTOUR, DEFAULT;
    }
}
```