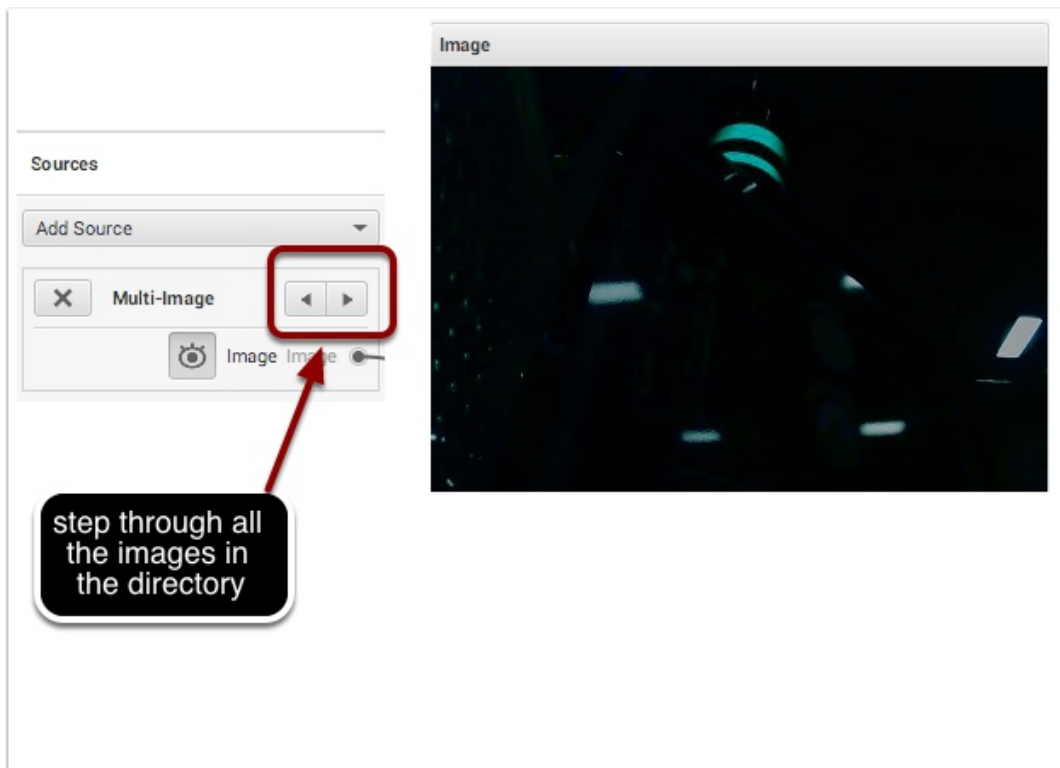


Processing images from the 2017 FRC Game

Processing images from the 2017 FRC Game

One strategy for detecting the retroreflective vision targets is to use an LED ringlight around the lens on your camera and setting the camera brightness so only the target is visible. Then you can filter for the LED ringlight color and find contours (connected sets of points) in the image. There are a set of images and the GRIP save file provided by FIRST that were taking this way [here](#). By starting the GRIP pipeline with a multi-image (files) set of sources you can use the left-right arrows to cycle through all the images in a set and make sure that your code is properly tuned to detect just the vision targets.

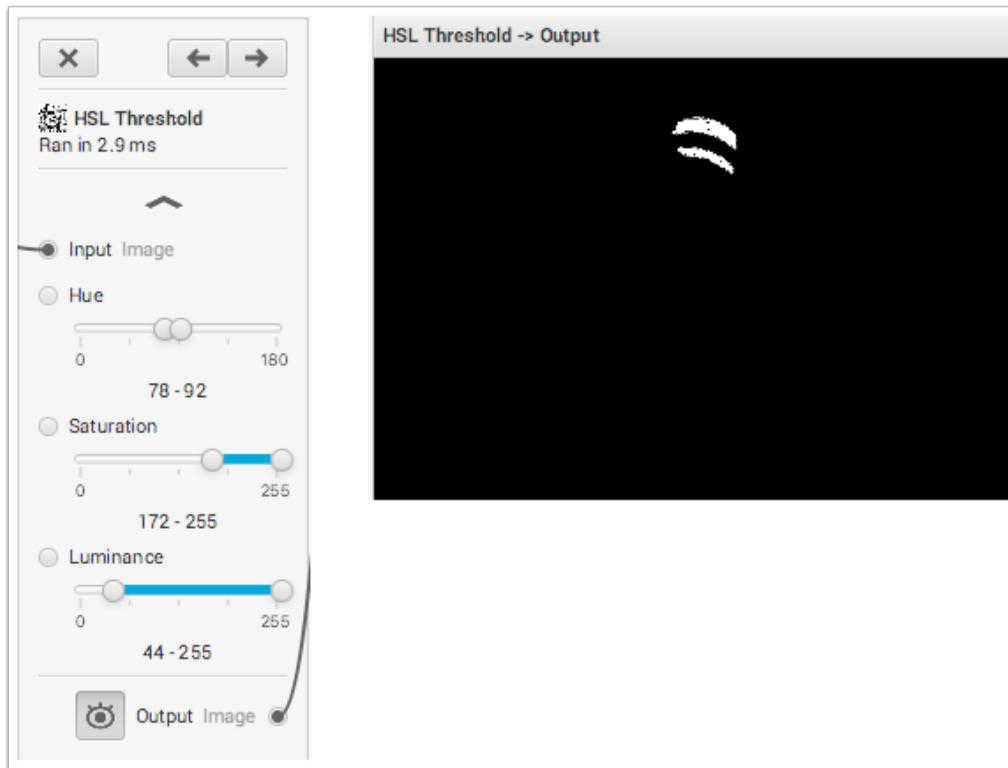


In this image you can see that everything that is not brightly lit is black and the only things visible are the reflected vision targets and some lights that were in the room. But the target is distinctly green (the LED color) and can be differentiated from the ceiling lights.

Filter out everything but the vision targets

Using an HSL Threshold operation you can filter out almost everything except the vision targets as shown in the following image.

Processing images from the 2017 FRC Game

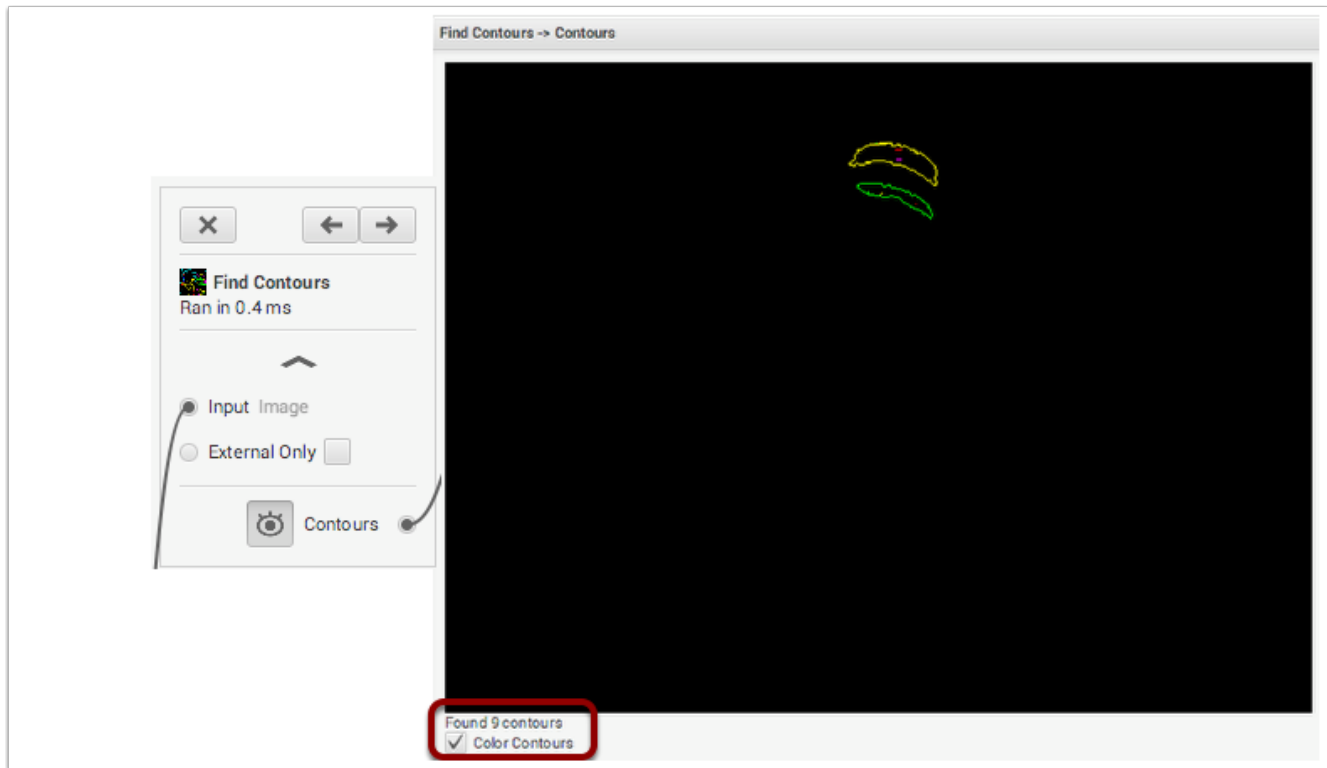


The HSL Threshold lets you specify an accepted range of Hue, Saturation, and Luminance values that will make up the pixels in the resultant binary (one bit per pixel) image.

Finding Contours in the image

The next step is to identify the selected areas that make up the targets. This is done using a Find Contours operation. Contours are contiguous areas of pixels that are lit in the image.

Processing images from the 2017 FRC Game

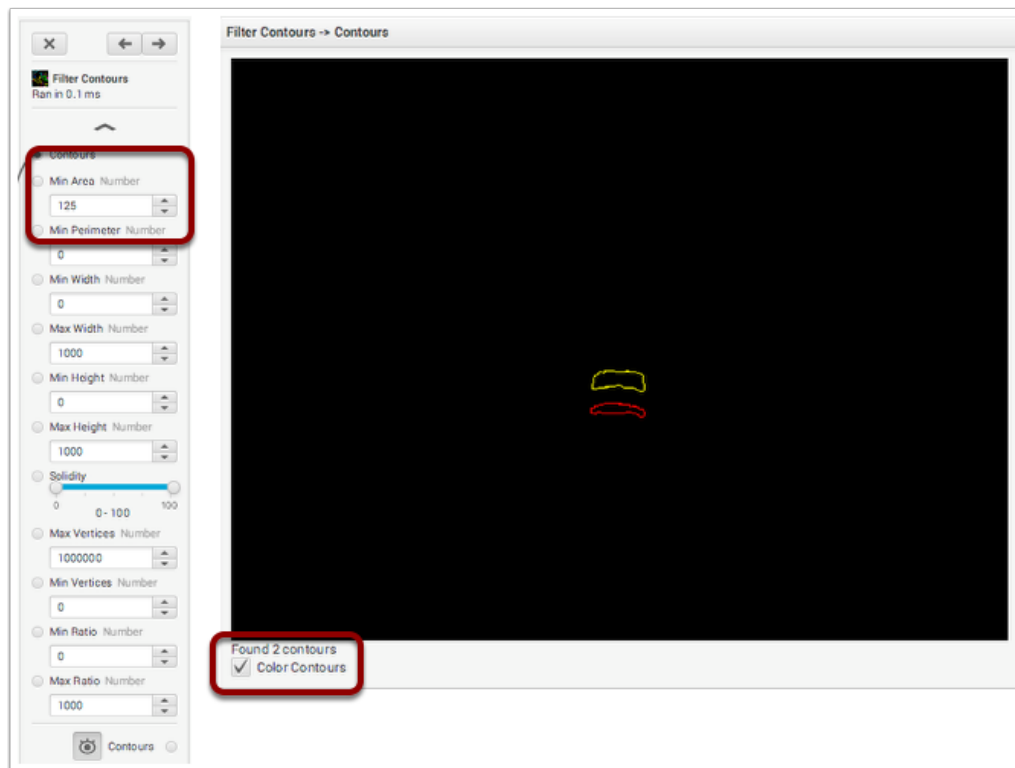


Notice that 9 contours were found in the image because of noise tha other other very small areas that were selected. The idea is to come up criteria to reduce th number of false positives.

Filtering out noise

You can reduce the number of extraneous contours found a number of ways. One is to filter on the area of the contour. In the following example, a minimum area of 125 pixels was shown. The filtered contours are shown in the next image.

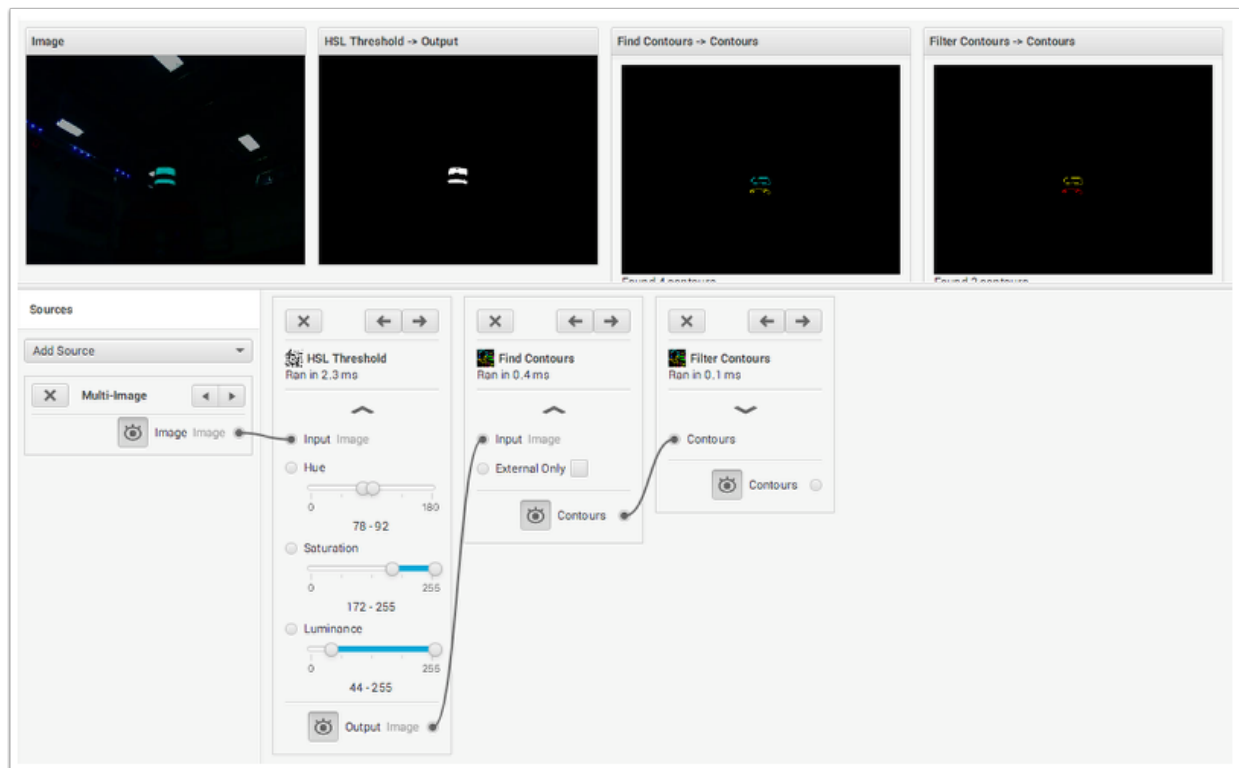
Processing images from the 2017 FRC Game



And now there are only 2 contours remaining representing the top and bottom piece of retroreflective tape.

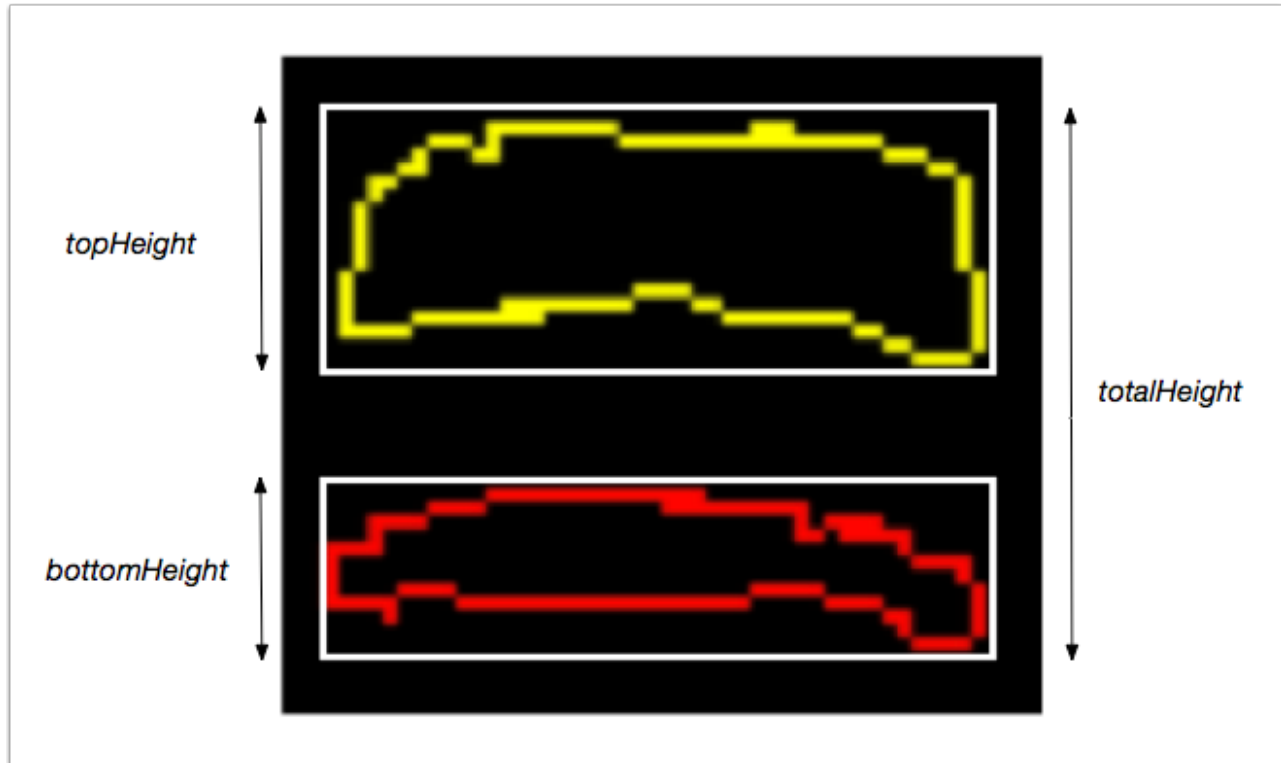
Here is the full GRIP pipeline that is described in this article.

Processing images from the 2017 FRC Game



Processing images from the 2017 FRC Game

Further processing



From there you may wish to further process these contours to assess if they are the target. To do this:

1. Use the `boundingRect` method to draw bounding rectangles around the contours
2. Use the following sample heuristics as a starting point that you can apply to score your found targets.

Each of these ratios should nominally equal 1.0. To do this, it sorts the contours by size, then starting with the largest, calculates these values for every possible pair of contours that may be the target, and stops if it finds a target or returns the best pair it found. In the formulas below, 1 followed by a letter refers to a coordinate of the bounding rect of the first contour, which is the larger of the two (e.g. 1L = 1st contour left edge) and 2 with a letter is the 2nd contour. (H=Height, L = Left, T = Top, B = Bottom, W = Width)

// Top height should be 40% of total height (4in / 10 in.)
 $\text{Group Height} = 1H / ((2B - 1T) * .4)$

// Top of bottom stripe to top of top stripe should be 60% of total height (6in / 10 in.)
 $\text{dTop} = (2T - 1T) / ((2B - 1T) * .6)$

Processing images from the 2017 FRC Game

```
// The distance between the left edges of contours 1 and 2 should be small relative
// to the width of the 1st contour (then we add 1 to make the ratio centered on 1)
LEdge = ((1L - 2L) / 1W) + 1
```

```
// The widths of both contours should be about the same
Width ratio = 1W / 2W
```

```
// The larger stripe should be twice as tall as the smaller one
Height ratio = 1H / (2H * 2)
```

Each of these ratios is then turned into a 0-100 score by calculating: $100 - (100 * \text{abs}(1 - \text{Val}))$

To determine distance, measure pixels from top of top bounding box to bottom of bottom bounding box

distance = Target height in ft. (10/12) * YRes / (2 * PixelHeight * tan(viewAngle of camera))

You can use the height as the edges of the round target are the most prone to noise in detection (as the angle points further from the camera the color looks less green). The downside of this is that the pixel height of the target in the image is affected by perspective distortion from the angle of the camera. Possible fixes include:

1. Try using width instead
2. Empirically measure height at various distances and create a lookup table or regression function
3. Mount the camera to a servo, center the target vertically in the image and use servo angle for distance calculation
4. Correct for the perspective distortion using OpenCV. To do this you will need to calibrate your camera with OpenCV as described [here](#).

This will result in a distortion matrix and camera matrix. You will take these two matrices and use them with the `undistortPoints` function to map the points you want to measure for the distance calculation to the "correct" coordinates (this is much less CPU intensive than undistorting the whole image)