Project 6 ENPM 673

Rene Jacques and Zachary Zimits

May 18, 2019

1 Sign Detection

1.1 Blue Signs

The process for detecting blue signs begin by converting the current frame into HSV colorspace. Once converted we apply a simple threshold operation to filter out the majority of the objects in the scene. We then convert the new image to gray-scale and find contours on it. The first check we preform on the contours is to see if they have an area greater 300 pixels. This filters out any small false positive like decals on the side of the truck or signs in shop windows. We then calculate a bounding box for the contours and compare the ratio of width to height. Since none of the blue signs are oriented horizontally we use the constraint $width < 1.2 * height$ to filter out any contours on objects like the road name signs. The remaining ROIs are extracted from their bounding boxes and fed into our sign identification pipeline to determine if they are a sign we are looking for.

Figure 1: Left Final Output, Right HSV Color Segmentation and Contours

1.2 Red Signs

Detection of red signs follows a similar track by beginning with a color threshold in the HSV space and an initial check to make sure that the area enclosed in the contours is large enough.

1.2.1 Triangular Signs

Of the five red signs we are looking for in this project four are triangles and three of those are oriented right side up. Again a bounding box is constructed for these countours but since all the triangles should fit within a square box we use the constraints $width < 1.2 * height$ and $height < 1.2 * width$ to filter out non-signs. Now that we know the bounding box is the right size we look at the actual shape of the contour. We calculate the centroid of the contours using *cv2.moments()* and find centroid_dist, the distance from the top of the bounding box(y) to the centroid(cy) using Equation 1

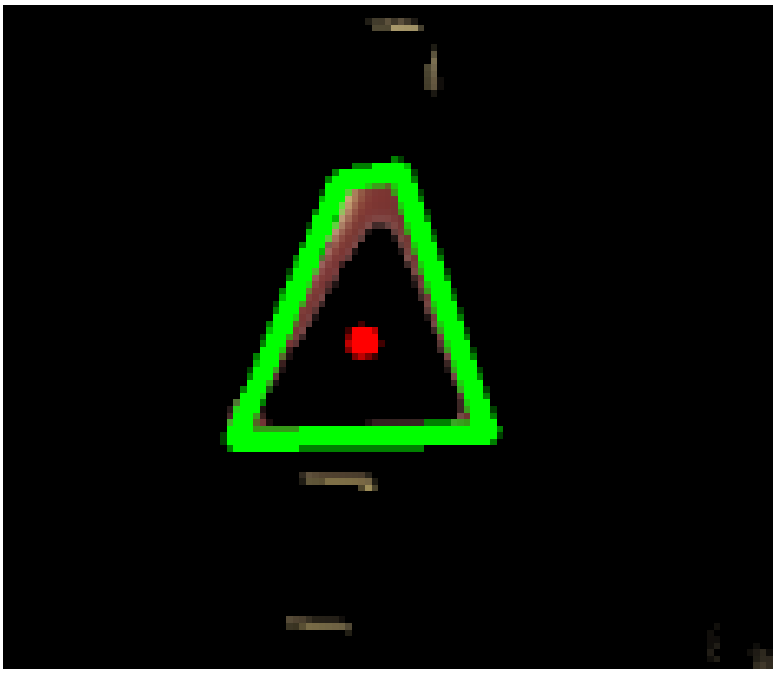

Figure 2: Centroid Location in Triangular Contour

$$centroid\_dist = cy - y \tag{1}$$

For triangles that are right side up we use the conditions $centroid\_dist > 2 * h/3.5$ and $centroid\_dist < 2 * h/2.5$ since triangle have their centroid 1/3 of the way from their base to their tip. We also checked to ensure that the centroid was in the center of the contour in the x direction with the conditions $cx > 0.8 * (x + w/2)$ and $cx < 1.2 * (x + w/2)$ For the yield sign that uses an inverted triangle the same equations are used exact the equations that check the centroid height are divided by 2.

1.2.2 Stop Signs

To detect stop signs we use a slightly different threshold operation. Firstly since the stop sign has the bright white letters processing the image with the same threshold as the other red signs resulted in two separate contours being detected which the rest of our pipeline was not able to handle. Instead we changed the threshold parameters to capture a larger portion of the stop sign. While this did do a good job of highlighting the stop sign it also captured a lot of other dark colors and shadows. To filter these out we used a second threshold operation on the BGR image of the frame to find the areas that we bright enough the be the sign.By combining these two masks together we were able to achieve good

detection of the stop sign. The only other check that we did for these signs was to ensure that it fit in the same bounding box ratio as the triangles.

2 Sign Identification

2.1 Histogram of Oriented Gradients

The first step to identifying signs was to break down the sign images into descriptors. We used histograms of oriented gradients (HOG) to create the descriptors, specifically by using OpenCV's *HOGDescriptor* class. This class takes in several initialization parameters, the most important of which are window size, block size, block stride, and cell size. Window size is the size of the image that is being processed into descriptors, and was set as $(64, 64)$. Block size counters variations in illumination and was set to $(16, 16)$. Block stride determines the overlap between blocks and was set to $(8, 8)$. Cell size determines the number of descriptors for the image. One cell contains one descriptor, so the smaller the cell the more descriptors there are of the image. The cell size was set as $(4, 4)$. These parameters serve to initialize the *HOGDescriptor* class that will process sign images into the data that will be used to train and test the support vector machines (SVM) described in the next section.

2.1.1 Processing Training Images

Each of the images in each of the class folders withing the training folder for the given data were read in and resized to be $64x64$ images. This is to match the size that the *HOGDescriptor* class is expecting. The descriptors for each resized image were then computed using the *HOGDescriptor*'s *compute*() function, and the descriptors were appended to a list. At the same time, the corresponding image labels (the names of the folders the training images were in) were saved to another list. Both of these lists were then used to train the SVM in the next section.

2.2 Support Vector Machines

2.2.1 Training

The SVM used for training and detection was created using $cv2.ml.SVM\_create()$. Its type was set as SVM_C_SVC, This allows the variable C to be set, which controls the bias used in classifying data. C was set to a default value of 1, and in testing this seemed to produce the desired results. The kernel type for the SVM was set as SVM_LINEAR, making the model a linear one, and therefore reducing the number of additional parameters that would need to be set if the model was non linear. Finally, training was completed using the function $svm.train()$, where the descriptor list and the label list from Section 2.1.1 were inputs to the model. The training took around 20 seconds, and once complete,

the model was saved so that training would not need to be completed every time the video was processed. The model accuracy on test images was 95%.

2.2.2 Identification

Once a potential sign has been identified, as described in Section 1, it needs to be extracted and processed before the SVM that we trained can predict the sign's class. After computing a bounding box around the contours that have been identified as potential signs, the coordinates, width, and height of the bounding box are used to extract the image data from the frame and create a sign image. This sign image is then resized to be $64x64$, and is then fed into the *HOGDescriptor* class described in Section 2.1. The computed image descriptors are then input into *svm.predict*(), which predicts the sign class based on the provided training data. This prediction is then used to display a corresponding sign next to the sign in the frame, thus indicating which sign has been predicted. Since the assignment only calls for the detection of signs 1, 14, 17, 19, 21, 35, 38, and 45, these are the only signs that are shown in the video if they are detected and identified. If any other sign predictions are returned they are not displayed.

3 Results

Our sign detection algorithm is capable of finding every frame the only issue we had with not finding a sign is the speed bump sign at the top of the pole right before the van turns into the driveway. The reason that if misses this last one is because by the time that the sign is close enough to the car for the contours to capture a complete triangle the top portion of the sign is off the frame causing the contour area to be to small. While we were able to detect every sign we did have a few problems when trying to identify them. The first error is when the truck drives by while the van is stopped at the stop sign. There is a blue mark on the side of the truck that gets identified as directional arrow. The other major problems come with another directional arrow which it can only identify for a couple of frames and one of the road narrowing signs which it also has trouble identifying.

Figure 3: False Positive on the Truck

4 Video Output

The processed video showing sign detection and identification can be found at this link:
https://drive.google.com/drive/folders/1L7Pyt41PDXRmsFJT$_1$q9$JRf22yoLk5O_?usp = sharing$)