Project 6 – Traffic Sign Detection

Submitted in partial fulfilment of the requirements for the course of

ENPM673 – Perception for Autonomous Systems

By

Adheesh Chatterjee

Andre Ferreira

Pablo Sanhueza

Course Faculty
Prof. Cornelia Fermuller

A. James Clark
SCHOOL OF ENGINEERING

INTRODUCTION

The aim of this project is to to do Traffic Sign Recognition which is a two-step process of detection and recognition. The concepts learned in class will allow us to write a script that solves the problem of traffic sign recognition. In the Detection stage we aim to extract possible candidates (or regions) which contain a traffic sign. In the Classification stage, we go over each Region of Interest (ROI) extracted previously and try to identify a traffic sign (which might not be in that ROI at all).We use HOG feature detector, MSER feature detector, SVM routines to create the complete pipeline. Below you will see explanations of what we did to accomplish such an arduous task.

DATA PREPARATION

We receive 3 folders containing the frames of the video, the training set and the test set. We first denoise the frames using a Gaussian Blur with a 5x5 Kernel. Since we are dealing only with Red and Blue street signs, we do not necessarily use the green channel. The training and test set are handled as part of the classification.

TRAFFIC SIGN DETECTION

The traffic sign detection is tried using 2 methods – Thresholding in HSV color space and using Maximally Stable External Regions (MSER algorithm)

HSV

One approach tried for the implementation of this project was change the color space of the video frames from RGB to HSV. Once we have the image as HSV we choose a range of red to filter the image.

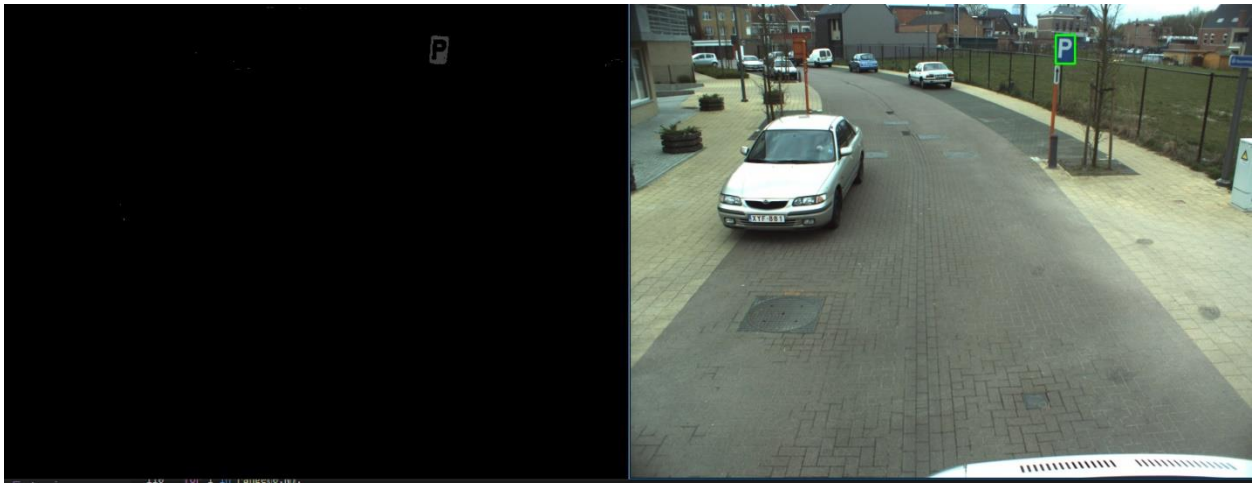
The resulted image gives us a binary image with the blue pixels that are in the range chosen previously, and for this image we apply cv2.findContours in order identify the sign. Since the sign is not the only blue feature in the frame, there are considerable number of outliers which leads to undesired contours and because of that is necessary to filter it in order to get the appropriate ones.
OpenCV findcontours is equipped with many features to filter the contours, for this approach we used the following ones:

Area: There are many tiny and large undesired contours, to filter them we choose a minimum and a maximum area for the contours.

Aspect Ratio: Cv2.boudingRect allow us to get the height and width of the bounding box, with this information we can calculate the aspect ratio of each contour.

Extent: Since we have the area of the contour and the area of the bounding box(height*width) we calculate extent which is given by the ratio between the area of the contour and the area bounding box.

This methodology has given good results for the blue signs as you can see in the following Image:



For the red signs the results were not as good as the ones from the blue signs , the reason for that is given because the red color is more prevalent in the environment which leads to a higher number of outliers .

This outlier were harder to filter and even dough we were able to filter a lot of it, there was still some outliers that were interfering on the results of some frames of the video. On the other hand It was possible to identify all the red signs.

But these outliers were given as result some false positives and because of that we chose to try MSER in order to get better results.

MSER

Noise Removal

As stated in data preparation, first a Gaussian Blur is applied with a 5x5 kernel is applied on each individual RGB plane.

Contrast Normalization

Contrast Normalization is done using the imadjust function created. Here the image intensities in a grayscale image are adjusted so that 1% of the data is saturated at low and high intensities. This step is done to increase the contrast of the red color in the red channel and the blue color in the blue channel.

The intensities are then normalized as suggested by [1] using the formula –

$$C = max(0, \frac{min(R-B, R-G)}{R+G+B})$$

$$C = max(0, \frac{min(B-R)}{R+G+B})$$

Where the first formula is used to detect the red signs and the second one detects blue signs

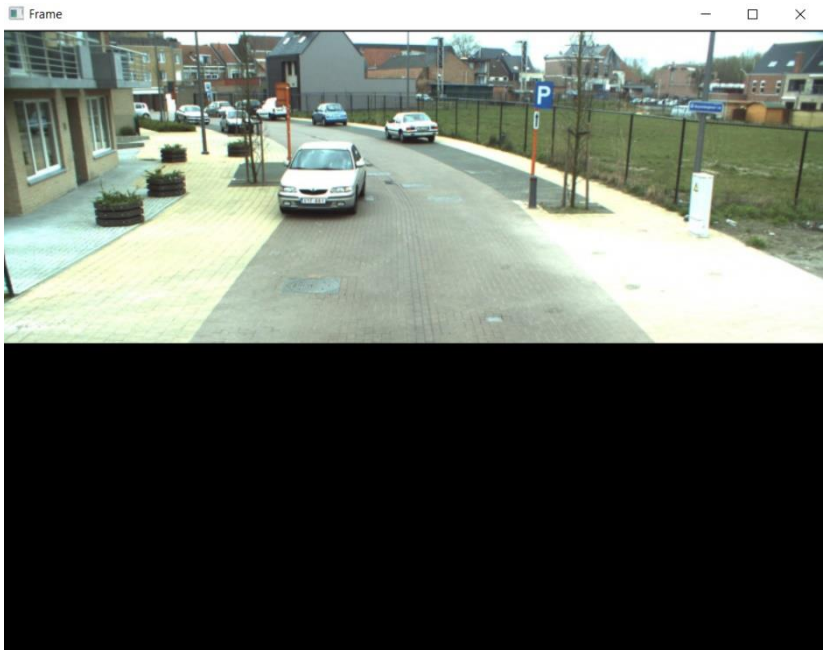
The outputs are shown below –

Red

Blue

Extract MSER Features

A black mask is placed on the bottom half of the frame as traffic signs cannot be detected near the ground. The size of the bounding box of the detected sign also reveals information about the possible location of the sign i.e., if the bounding box is quite small then it could be that the sign is really far away and it should be closer to the horizon (or top) in the image. Else if it is bigger, then it is closer to the center line of the image. The output is shown below



The cv2 function for MSER create is now used to detect features in the red and blue normalized images. A convex hull is found for all the features. And we tried plotting the contours directly on the image. But on further investigation, we found that multiple contours are plotted in the same spot.

We use the MSER create class to get the best detection of traffic signs by changing parameters. More description of the parameters can be found in [3]

§ create()

```
static Ptr<MSER> cv::MSER::create ( int       _delta = 5,
                                    int       _min_area = 60,
                                    int       _max_area = 14400,
                                    double    _max_variation = 0.25,
                                    double    _min_diversity = .2,
                                    int       _max_evolution = 200,
                                    double    _area_threshold = 1.01,
                                    double    _min_margin = 0.003,
                                    int       _edge_blur_size = 5
                                  )
```

Here a trick was thus used. We created a black mask of the same size as frame. We then plotted all the contours detected as filled contours of white color. Thus making it a binary image. We then did contour detection on this binary image formed to get only one contour for every individual detection. The output is shown below

Blue at frame 70

The MSER was created using different parameters for both red and blue depending on what gave better results. Even when separating contours we used multiple fine-tuned tricks.

For the blue contours, we realized that we have to just detect rectangular and circular signs. Thus the aspect ratio of every of contour was found, and only if it was <=1, it was plotted, this helped reduce false positives.

For the red contours, since we had triangular signs and an octagonal sign, the aspect ratio was made to be >=1 and an area threshold of between 1000 and 3000 was used. Another thing we realized was that for triangles, the centroid lies 2/3 from the top. So the width and heights of the bounding boxes were found. They were normalized between 0 and 1, and the difference between the normed values was evaluated as a way to separate the contours. This improved the detection considerably. We also used cv2 match shapes to get the best octagonal shape for the Stop sign.

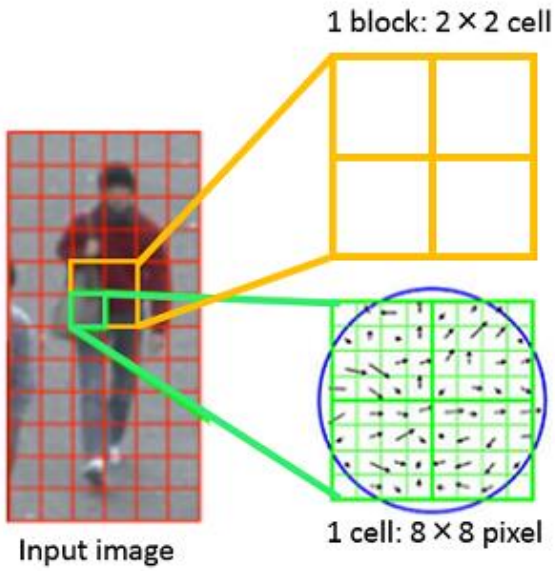
We were thus able to detect almost all the signs with very few false positives.

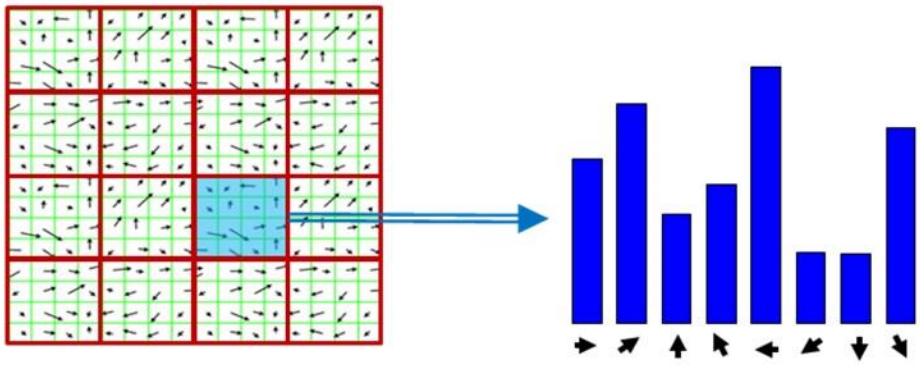
CLASSIFICATION

One of the most important sections in this project is the one of image classification. For this we used histogram of oriented gradients (HOG) and a Support Vector Machine (SVM) model. To use the feature descriptor we resize the training images to a 64 x 64 x 3. From here, the HOG extracts features of the image and converts these to a vector of 1 x n shape where these are then given to the SVM. Please see below a more in depth view of what was done to train the model.

HOG

The HOG descriptors are derived from histograms and direction of gradients. The gradients (x and y) are useful in this case to be able to detect edges and corners, which are great for specific object detection. The HOG descriptors were obtained from the OpenCV function called cv2.HOGDescriptor(). As for most functions, there are some restrictions and in this case the size of the image has to be of a certain aspect ratio. For example a 1:2 ratio. The OpenCV function first takes the gradients in the vertical and horizontal directions of the image and then from these it calculates the gradient magnitude and direction. From here, the function breaks up the image into blocks and cells and creates a histogram of 9 bins. These bins represent the corresponding angles such as 0, 20, 40, 60, 80, 100, 120, 140, and 160. The resulting vector of the HOG descriptor is basically the represented by its own name, which gives back the histogram of the oriented gradients. In other words the values of the resultant vector is the normalized population of each bin in the histogram.





SVM

The second half of this section is to classify the images based on the information given by the HOG descriptors. For this we used a Supper Vector Machine that takes as input the HOG descriptor with the images respective label.

First, we classify images based on what type of sign they are and give them a different label. For this we labeled the given training images their corresponding folder names. For example images in folder '0001' has a label of integer 1 and images from the '0009' folder had a label of integer 9. All of this is done in a for loop for all images of type '.ppm' in the classifierSVD.py file.

To train the model we first need to initialized it and set several different parameters. The 4 main parameters that need to be set are the type of SVM, C, gamma, and the type of kernel that will be utilized. In this case we opted to use the values and types below.

$$\boldsymbol{Type\ of\ SVM} \rightarrow cv2.ml.SVM\_C\_SVC$$

$$C = 1$$

$$Gamma = 0$$

$$\boldsymbol{Kernel} \rightarrow\ cv2.ml.SVM\_LINEAR$$

Here the type of SVM that was used is an n-class SVM. In this case we opted to use all the training set and therefore we have 62 classifications (labels range from 0-61). The kernel type is linear, which means that there is no mapping and we use a linear regression for the original feature space.

To train the model we stack all the HOG descriptors of each training image vertically. We then use the svm.train() function to obtain and save the model to later use in the main code file to detect traffic signs.

Since the OpenCV SVM prediction function doesn't give us a probability of success for each image being compared, we opted to used all the training dataset to filter as many false positives as possible.

SVM Testing

To test our model we run a for-loop that reads all the Testing set images. In this for-loop we obtain the HOG descriptor of each image and use the svm.predict() function to obtain the predicted label. To verify the accuracy of our model we count how many times it accurately detects the right label, and also count the times if doesn't match the traffic sign being analyzed.

Please see small section of our code below.

```
svm = cv2.ml.SVM_load('SVM_TEST.xml')
testingImgs = glob.glob('Testing/*/*.ppm')
test_labels = []
results = []
count1=0
count2=0
for i in testingImgs:
    img = cv2.imread(i)
    testResponse = svm_predict(img, svm)
    results.append(testResponse)
    test_labels.append(int(i.split("\\")[1]))
    print('Original Label: ', int(i.split("\\")[1]))
    print('SVM Prediction: ', testResponse)
    print('==========================')
    if (int(i.split("\\")[1])==int(testResponse[0])):
        count1+=1
    else:
        count2+=1
print(count1/(count2+count1))
```

This model gives us a **96%** accuracy.

FINAL STEPS AND ISSUES FACED

After the classification was done, we were able to compare every detected sign with the trained model. Only if it matched the trained model, were we then drawing a bounding box around the sign and displaying a true output.

One of the few problems with this was, that the Parking sign (No. 45) was very similar to the Parking signs (No. 46 and No. 47) which is why there was a mismatch at times. Giving both false positives and false negatives.

In the beginning of the video, if you see, the parking sign blinks a lot. This is because the matched output is 46 and not 45. Thus the SVM training technique even though very good with an accuracy of about 95% isn't enough to differentiate the 3 types of signs properly.

Towards the end of the video, the same issue arises where the parking sign No.47 is being detected as 45.

This could maybe be fixed by using a different training technique or larger datasets to get better results.

(a) 45 **(b)** 21 **(c)** 38 **(d)** 35

(e) 17 **(f)** 1 **(g)** 14 **(h)** 19

SUMMARY

So to summarize the pipeline used –

1) We first denoise the image using a Gaussian Blur
2) Contrast Normalization is performed over each channel
3) Intensity is normalized in the image.
4) MSER feature extractor is used to detect regions from image for blue and red with different parameters
5) Every contour detected is plotted as a filled white contour on a black mask creating a binary image.
6) The contours of the binary image are found
7) The contours found are compared with the classification model and if there is a match, the bounding box is placed in the original frame
8) The true output of the sign is displayed next to the detection.

REFERENCES

[1] S. Salti, A. Petrelli, F. Tombari, N. Fioraio, and L. Di Stefano. A tra_c sign detection pipeline based on interest region extraction. In The 2013 International Joint Conference on Neural Networks (IJCNN), pages 1-7, Aug 2013.

[2] http://www.micc.unifi.it/delbimbo/wp-content/uploads/2011/03/slide_corso/A34%20MSER.pdf

[3] https://docs.opencv.org/3.4/d3/d28/classcv_1_1MSER.html