Project 4 Report

Ashwin Goyal; UID: 115526297

May 16, 2018

Introduction

In this project we aim to do Traffic Sign Recognition. This has been accomplished in two steps, detection and classification. Images from a driving car, and training/testing images for various signs have been provided.

Methodology

Traffic Sign Recognition can be staged into two sections, Traffic Sign Detection and Traffic Sign Classification. In the Detection stage, aim is to extract possible candidates/regions which contain some traffic sign. In the Classification stage, aim is to go over each Region of Interest (RoI) and identify what sign it represents.

Traffic Sign Detection

There are a lot of ways of detection. Traffic Signs can be broadly classified into two categories, RED (Danger & Prohibitory) and BLUE (Mandatory) based on color. In this section, the objective is to extract the Region of Interest (i.e., around a traffic sign). This can be accomplished with simple thresholding in an appropriate color space. But, this method is light sensitive. In HSV color space, though it is light insensitive, this method selects many regions which are not of interest (i.e., not a traffic sign). Another method is using MSER regions, a more robust method, to identify regions of similar intensity. This method, though robust, is not very productive when directly applied to the input image (Figure 1). So, images are pre-processed before extracting MSER regions.

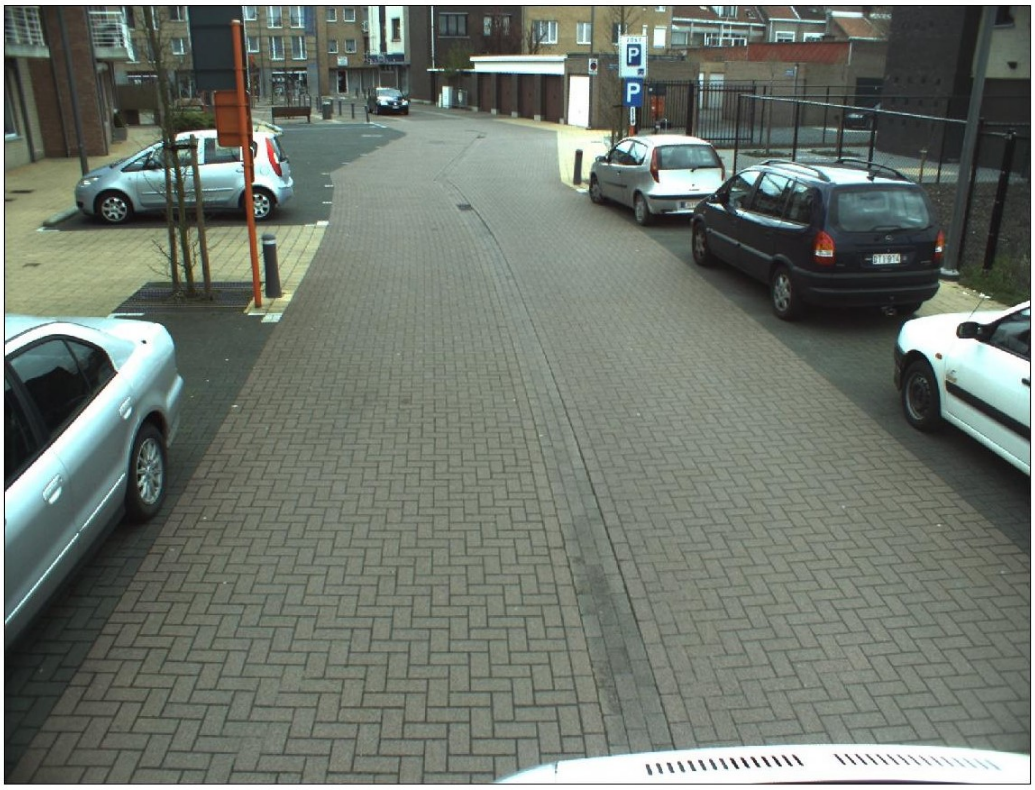

Figure 1: Frame 190 from the Data-set

In computer vision, maximally stable extremal regions (MSER) are used as a method of blob detection in images. A trivial intuition is that MSER gives regions of similar intensity given a grayscale image. This method of extracting a comprehensive number of corresponding image elements contributes to the wide-baseline matching. Thus, it is a good solution for this project as a traffic sign is mostly a uniform intensity region. But, as stated before, MSER regions does not produce a very effective solution when applied to the input image. So, the first step in the algorithm is **Noise Removal**. There is no unique way to accomplish this task. For this project, several filters were used before finalizing the algorithm. Currently, the algorithm uses two filters, Gaussian Filter `imguidedfilter` and Local Laplacian Filter `locallapfilt`, in conjunction to produce an output. First, guided gaussian filter is used to preserve the edges (Figure 2), and then local laplacian filter is used to remove unimportant features from the image (Figure 3). The difference in the filtered image and the input image can be clearly seen. In the filtered image, only the features with prominent edges are left.

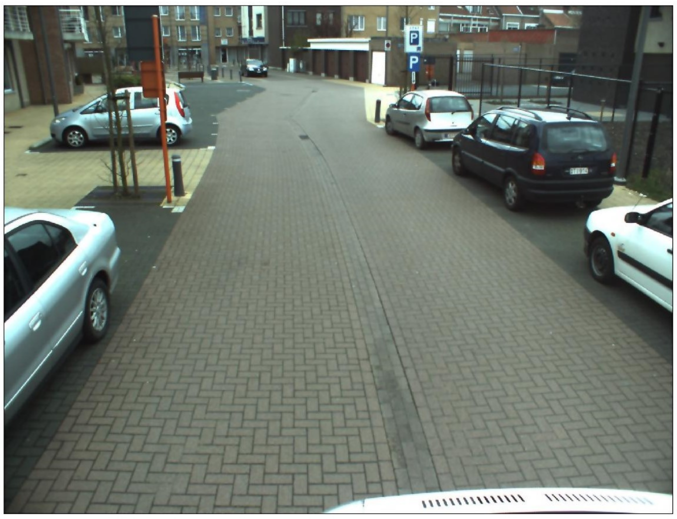
Figure 2: Image after Guided Gaussian Filtering

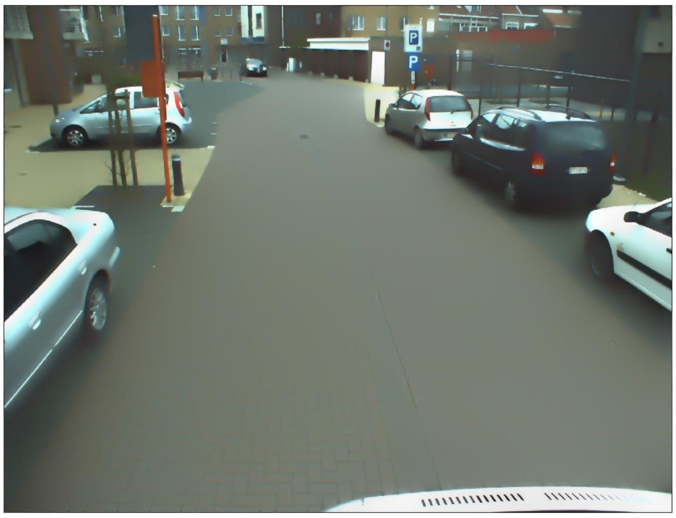
Figure 3: Image after Local Laplacian Filtering

Now, the image only has the necessary features. But, this image is still not good for the MSER region to work on. So, the next pre-processing step is **Normalization**. In this step, the image intensity is normalized. This step is significant as at this location the two color segments are separated. The equations used to normalize the image intensity are

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

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

The significance of this step can be seen from Figure 4 and Figure 5. The processed images only show the regions which have red component only and blue component only, respectively.

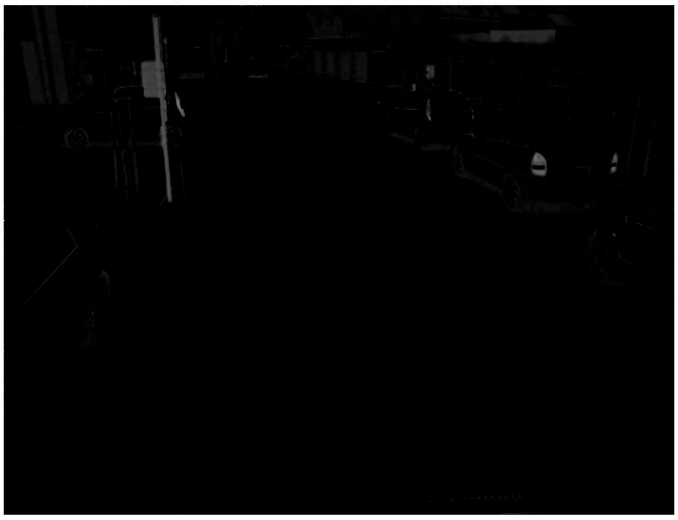
Figure 4: Image after Normalization for Red Signs

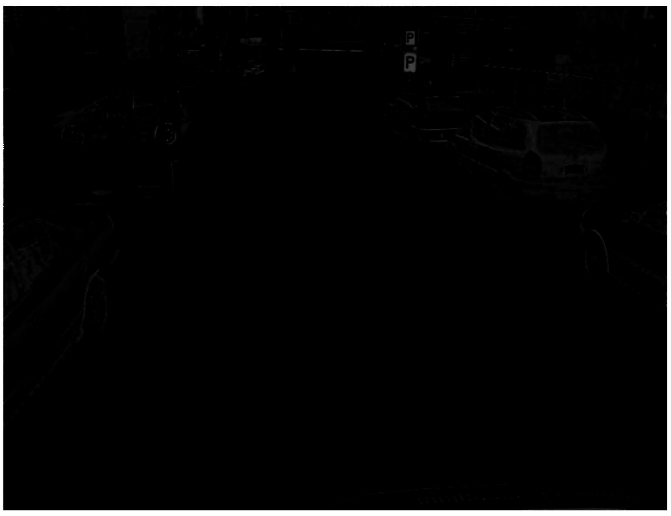
Figure 5: Image after Normalization for Blue Signs

After this step, the focus is to highlight the regions of interest. This was meant to be accomplished by stretching the image by setting the meadian of the image at 255, for unsigned integer type image. The algorithm finds the median of intensities and maps it to 128. That is, if the median is 80 then the algorithm maps the intensities from $0 - 80$ to $0 - 128$ using $\frac{128x}{80}$, and maps the intensities from $80 - 255$ to $128 - 255$ using $255 - \frac{127(255-x)}{255-80}$. But, this method highlighted regions other than the traffic sign as well. Also, when I tried to reverse the order, that is, first stretching then normalizing, it gave the same output as normalizing alone. So, the final algorithm, do not use stretching.

Now, the image is ready to extract MSER regions. But, there are several parameters in MSER that need to be tuned for the two sets of signs, red and blue. First parameter is *ThresholdDelta*. it is the step size between intensity threshold levels which has a range (0,100]. It was varied from 0.4 to 8 and the outputs after pre-processing were almost the same. So, it was kept at the default value of 2. Second parameter is *RegionAreaRange*. It is the size of the region in pixels. This resulted to be an important parameter as it helped to remove very small and very big regions. After some tuning, the range was set from 100 to 14000. Third parameter is *MaxAreaVariation*. It is the maximum area variation between extremal regions at varying intensity thresholds. This parameter is based on the knowledge that stable regions are very similar in size over varying intensity thresholds. This value was varied from 0.1 to 1.0, that is the entire range, but the outputs had not much improvement. So, it was kept at the default value of 0.25. The last parameter is *ROI*. It is the rectangular region of interest. It was the key to eliminate false positives. From the data set, it was observed that the blue signs only occur in the top-half of the image, and the red signs only occur in the top-right-quarter of the image. So, the ROI were set accordingly for the two sets. The outputs from MSER are shown in Figure 6 and Figure 7 respectively.

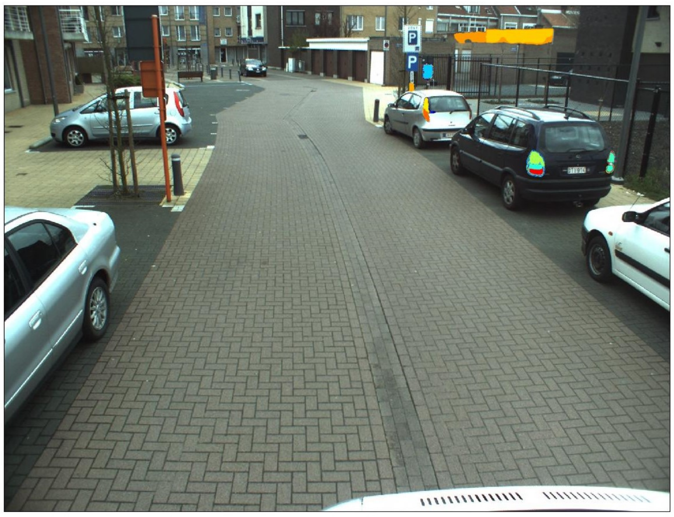
Figure 6: MSER regions for Red Signs

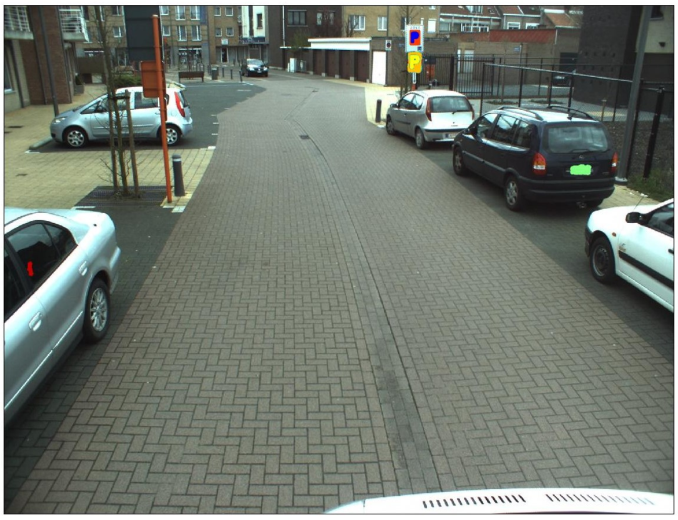
Figure 7: MSER regions for Blue Signs

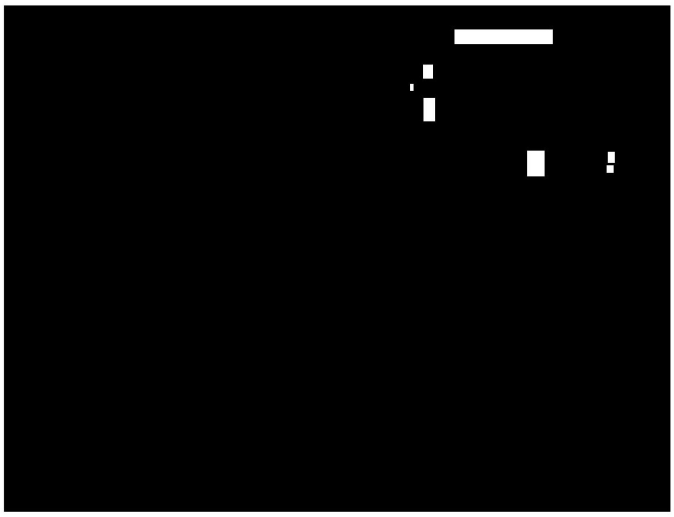
Figure 8: Binary Image using MSER for Red Signs

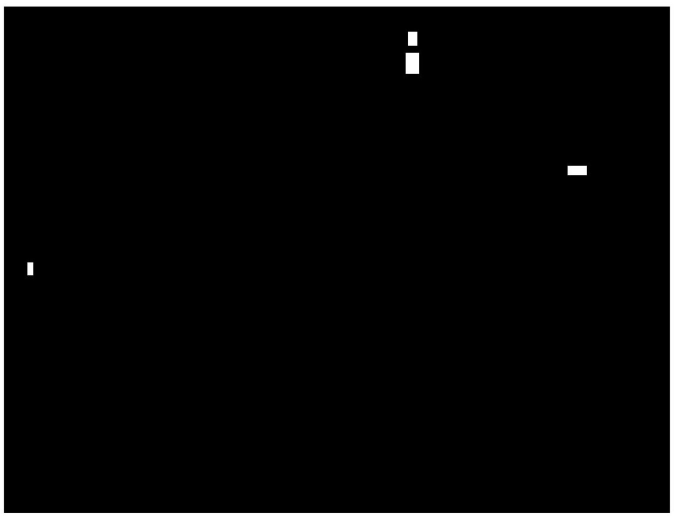
Figure 9: Binary Image using MSER for Blue Signs

From Figure 6 and Figure 7, we can observe that the selected regions are not only the traffic signs but other, red or blue, regions as well. So, in order to remove these unwanted regions, some post-processing was conducted. The first step in post-processing was to convert the images into binary on the basis of the MSER regions selected. This was accomplished as shown in Figure 8 and Figure 9, respectively. Then, to remove false positives a mask was created using the HSV color space. Using simple thresholding on the saturation and brightness values, the thresholds were effective to remove most of the unwanted regions as shown in Figure 10 and Figure 11 respectively.

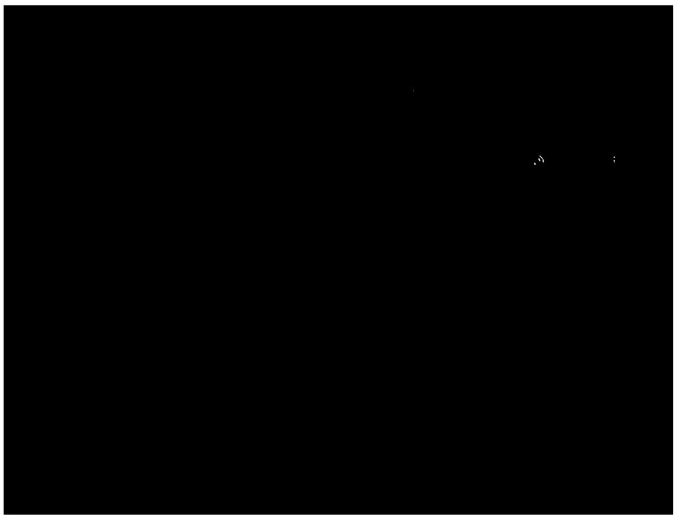

Figure 10: Final Binary Image for Red Signs

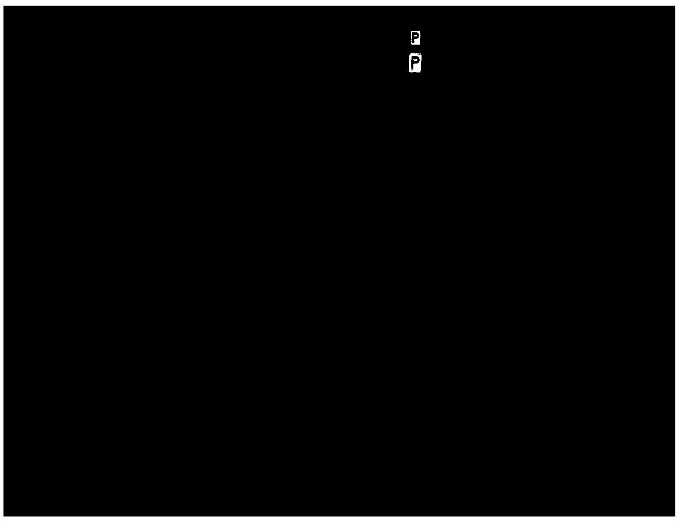

Figure 11: Final Binary Image for Blue Signs

The binary images generated were used to create bounding boxes around the traffic sign. Now, ideally, this step should be enough to detect the sign. But, as a fail safe, some other properties of traffic signs were exploited such as the area of the bounding box should be above a minimum value, set at 100, and the aspect ratio of the bounding box should not be more than a maximum value, set at 1.5. After this, though a redundant step, the algorithm checks for overlapping bounding boxes and keeps only the ones with higher area.

Now, the algorithm for traffic sign detection is complete. The last step of the algorithm is to make the bounding boxes in the original image (Figure 12).

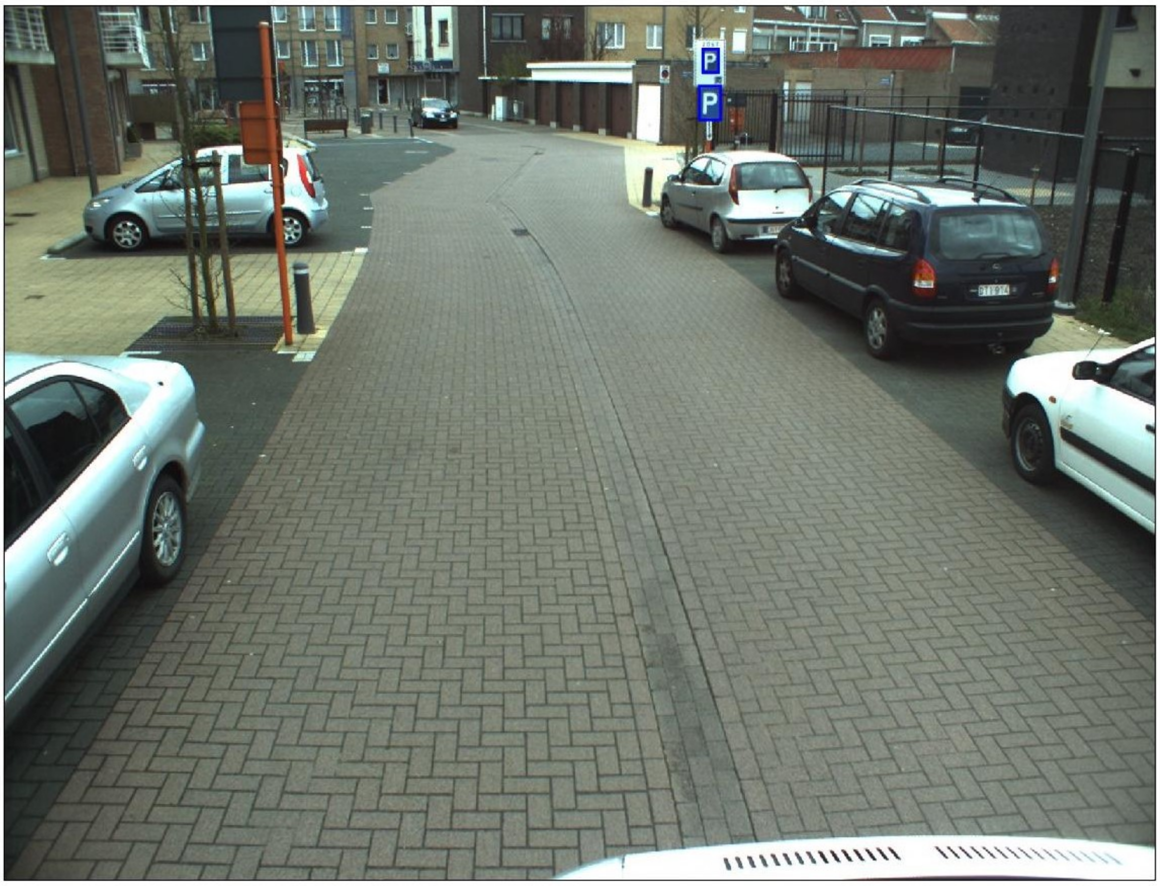

Figure 12: Image with Bounding Boxes

Traffic Sign Classification

Sample images for different signs have been provided to train the classifier. The training step is straight forward. The first step is to read all the images and create a labeled data set. As the images are of different sizes, all the images are resized to 64 x 64 before extracting features. The size of the image was set using the mean width and height of all the images provided in the data set. Next step in the training was to find features in the image for which the classifier would train on. These features were generated using `extractHOGfeatures` MATLAB function. This function takes *CellSize* as one of the parameters which was set, after multiple iterations, to [8 9].

Now, the labeled data set is used to train a SVM classifier using `fitcecoc` MATLAB function. This function takes care of non-binary classification, multi-class SVM. Some of the parameters of this function were modified. First parameter specified the use of parallel computing to increase the training speed. The other parameter was *Coding*. This is an important parameter as it determines how the training will be conducted by the function. There are seven different ways of training out of which "sparserandom" coding has been used to generate the given results. This was chosen as it gave the best result in the shortest time.

Then, to confirm that the training has been done correctly, the model was tested using test data. The accuracy of the model ranges from 97% to 99.5%. It should be noted that these steps are independent of Traffic Sign Classification. None of the above steps in this section were performed on the input images.

To achieve a complete Traffic Sign Recognition pipeline, the algorithm, first, resizes the image inside the bounding box to 64 x 64 and then extracts HOG features using a cell size of [8 9]. After the features are extracted, the model trained above is used to predict the sign. According to the prediction, the algorithm places a label image beside the bounding box to show that the sign has been classified correctly. As a fail safe, some conditions have been placed in the algorithm, which prevents the algorithm from making a wrong classification. The highlight of these condition is the prediction score. After studying the results, it was found that the score is always greater than -0.15, for a correct prediction. So, the algorithm only classifies a bounding box as a particular label if its score is greater than -0.15.

The final result after the completion of the pipeline is shown in Figure 13.

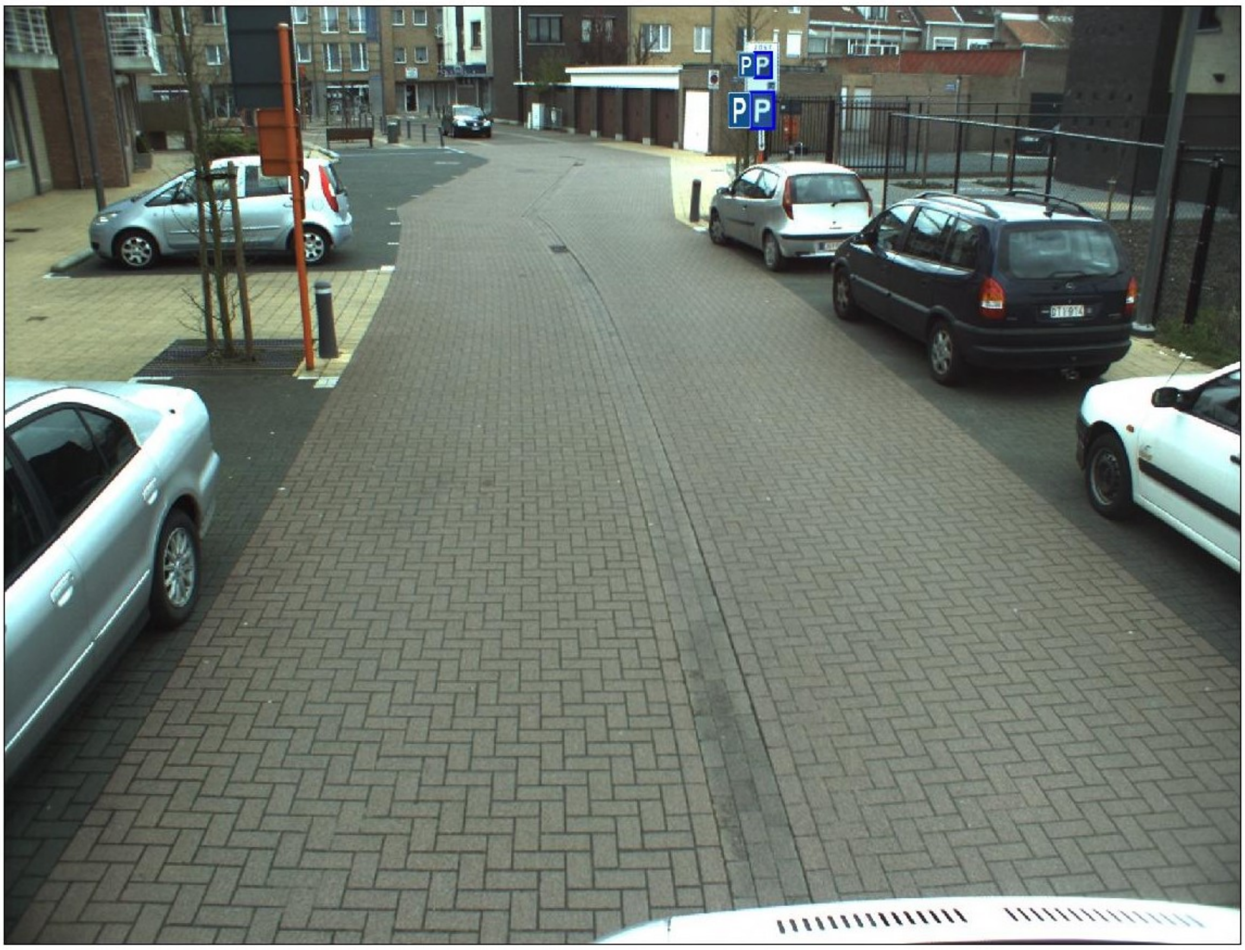

Figure 13: Final Output for Traffic Sign Recognition

Discussion

In this project, the major portions that took time were the tuning of the parameters, and the creation of video.

The tuning of parameters was conducted for, mainly, two functions, that is, to detect MSER regions and to extract HOG features. The latter was comparatively easy as the code was simply run to train on all possible cell sizes for all possible coding. This way, score matrix was generated for all posiible training parameters. Finally, the one with the highest accuracy was selected. The tuning for MSER regions was harder as it couldn't be done in a single code. It had to be tuned manually by running the algorithm for several frames. It was time consuming as well as took a lot of energy and computation power.

The creation of video was, by far, the most time-consuming part of this project. This is because the algorithm takes a lot of memory and time for each iteration. The concern for time increases as the code progresses because the memory used by MATLAB to create videos increases after every frame. The algorithm slows down exponentially with every iteration. So, in the end the algorithm was broken down into 6 sets of frames and ran on each set individually. Finally all the videos generated were merged to get the final video.

As can be observed from the generated video, the algorithm is not perfect. There are several false detections. It might need more tuning to improve the algorithm. Another way to improve the recognition is to train the SVM classifier on all possible traffic signs. This solution came across after observing that the algorithm detects all the traffic signs and classifies them into one of the trained categories. These proposed solutions have not been applied in the submission due to time constraints.

References

[1] MATLAB Documentation, *https://www.mathworks.com/help/matlab/*

[2] ShareLaTeX, Online LaTeXEditor, *https://www.sharelatex.com/*