ENPM673 Project2 Report

Bhargav Kumar Soothram
UID: 117041088
bsoothra@umd.edu

4 April 2022

1 Problem-1: Histogram Equalization

Most of the Computer Vision pipelines for lane detection or self-driving tasks require good lighting conditions and color information for detecting good features. Since we cannot expect such scenarios to start with in most processing pipelines, there is a need to accommodate for non-ideal circumstances. Here, our aim is to improve the quality of the image sequence provided.

We use the Histogram Equalization methods to enhance the contrast and improve the visual appearance of the video sequence. For the purposes of this project, we develop two kinds of histogram equalization: traditional and adaptive methods.

Below are the results for each of the methods.

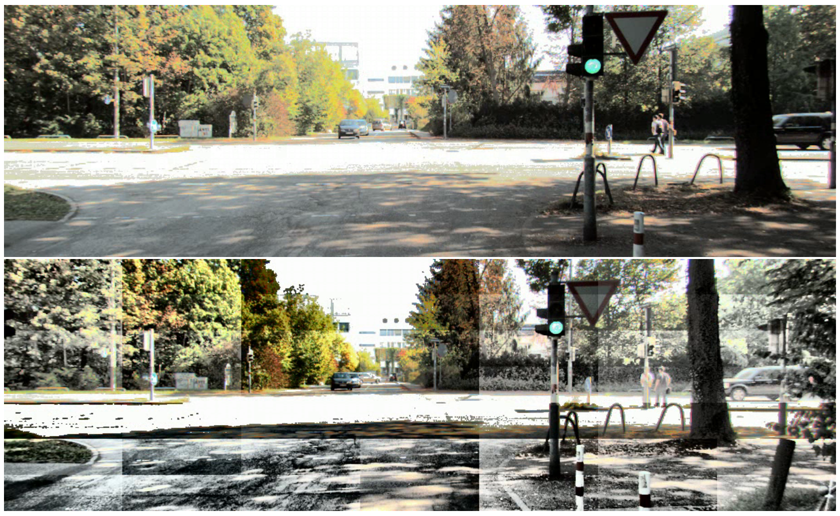

Figure 1: Top - Traditional method, Bottom - Adaptive method

We can clearly see that although the adaptive method provides us with better contrast, in our case, since there was no clipping done on values, we observe a considerable amount of noise. Also, we see rectangular artefacts because of our tile-like selection. Since the border values were not adjusted using interpolation, the artefacts are visible.

2 Problem 2: Lane Detection

Lane Detection is a ubiquitous problem that has gained a lot of traction lately. Here, we present a simple implementation of this popular problem using Hough Lines. Firstly, let us understand Hough Lines and Hough Transform.

Lines can be represented uniquely by two parameters. Often the form with parameters a and b or r and θ is used. The Hough space for lines has therefore these two dimensions; θ and r, and a line is represented by a single point, corresponding to a unique set of parameters (θ_0, r_0).

The algorithm for detecting straight lines using Hough Space can be divided into the following steps:

- Edge detection, e.g. using the Canny edge detector (I have used FFT in my approach).
- Mapping of edge points to the Hough space and storage in an accumulator.
- Interpretation of the accumulator to yield lines of infinite length. The interpretation is done by thresholding and other constraints.
- Conversion of infinite lines to finite lines.

The finite lines can then be superimposed back on the original image. Once the Hough lines are obtained, filtering is done on which Hough lines to preserve and which to discard. This differs from one use case to another. The results are shown in Figure 2.

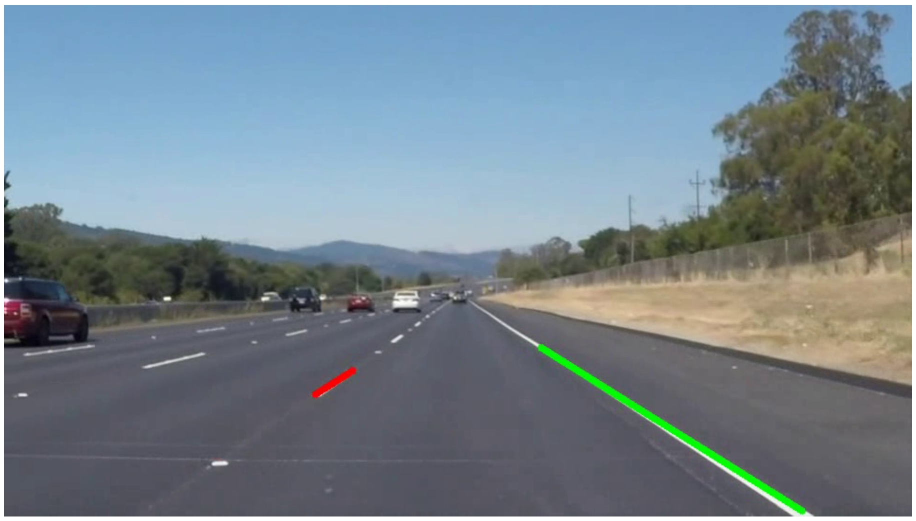

Figure 2: Lane Detection Applied

The major issue faced during the implementation was the segregation of Hough lines into dashed and solid lines. After getting the line parameters using cv2.HoughLines(), the slope was found out and then based on the slope, the line parameters were assigned to left or right lanes (positive - left, negative - right).

In case this approach fails, there is another check where in the highest detected y coordinate value is noted and this generally represents the solid line (since dashed lines are not that far off in all the cases due to the applied edge detection).

Then, to make it work when the image is horizontally flipped, I have used a simple parameter: the number of lines in each of the lanes. Whichever list count is greater, represents the solid line and other represents the dashed lines.

I believe that this pipeline would work for most similar scenarios, provided the edge detection is good. For example, I have tried the same pipeline on another video that has worse resolution and it did not produce good results. So, wider application highly depends upon how well the edges are detected.

3 Problem 3: Turn Prediction

In this section, the goal is to detect curved lanes and predict the turn depending on the curvature: left or right. For this, we need compute the radius of curvature of the road by fitting a curve through the detected lane lines. The pipeline used to solve the problem is shown below:

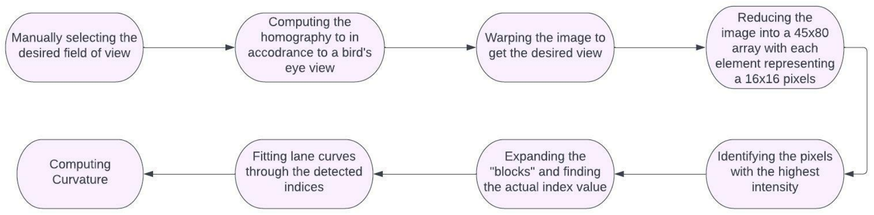
Manually selecting the desired field of view
Computing the homography to in accodrance to a bird's eye view
Warping the image to get the desired view
Reducing the image into a 45x80 array with each element representing a 16x16 pixels
Identifying the pixels with the highest intensity
Expanding the "blocks" and finding the actual index value
Fitting lane curves through the detected indices
Computing Curvature

Figure 3: Pipeline used for turn prediction

The resulting turn prediction is shown in Figure 4.

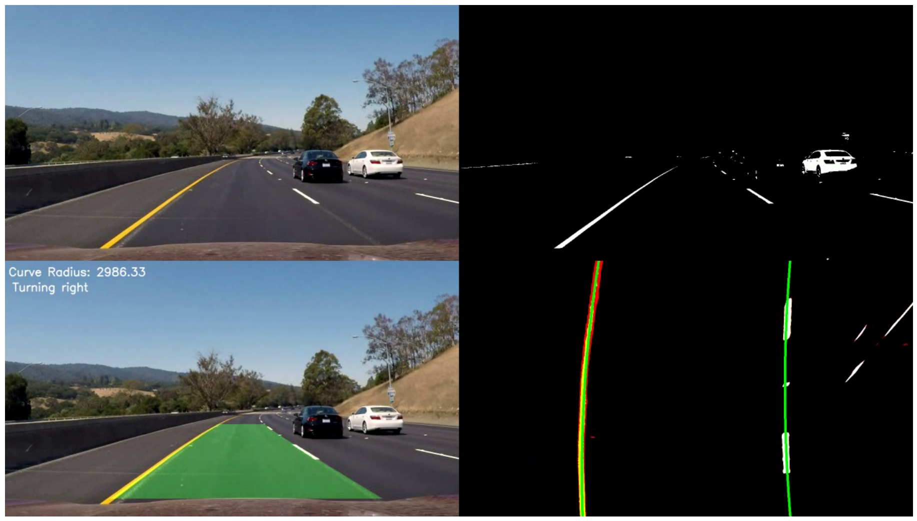

Figure 4: Sample frame from the results obtained

The biggest issues faced here were the exposure (shadows and highlights) and the resulting road colour. Since line detection is difficult in these cases, I have used a weighted average approach here, where the curve fitting parameters that were historically stored weigh in heavily and predict the lane lines. A sample frame is shown below:

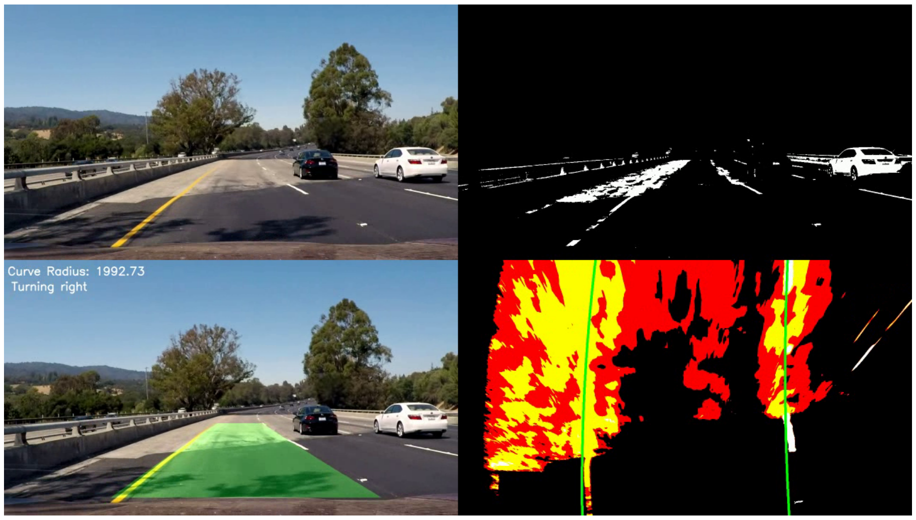

Figure 5: Predicted lanes when the lane detection is improper

The pipeline presented works even when the image is horizontally flipped and its wider use depends upon the same factors that were discussed in problem 2, with the most important being the quality of edge detection.

4 Video links

- Histogram Equalization - Regular method
- Histogram Equalization - Adaptive method
- Lane Detection
- Turn Prediction