UNIVERSITY OF MARYLAND, COLLEGE PARK

Lane Detection & Prediction for Self-Driving Cars

Pranav Inani

pranavinani94@gmail.com

This report is a commentary on the steps involved in systematically performing Lane Detection and Prediction for Self-Driving Cars. We begin by discussing some image processing techniques like filtering and edge detection to initially prepare each frame. Then, the use of Hough Transform to find significant lines in region of interest to find lanes is analyzed. Next, the concept of vanishing point utilized to perform lane prediction is explained. The software implementation implementation on MATLAB®[1] is discussed wherever deemed necessary. Finally, the results and conclusions of the project will be stated. The report will then conclude by touching upon possible future work. It should be noted that the dataset was obtained from Udacity[2].

Keywords: Lane Detection, Lane Prediction, Self-driving Cars, Sobel Operator, Hough Transform, Lane Departure Warning.

CONTENTS

1 Lane Detection **3**

1.1 Noise Removal . . . 3
1.2 Conversion to Grayscale & Thresholding . . . 3
1.3 Edge Detection . . . 4
1.4 Region of Interest (Masking) . . . 4
1.5 Hough Lines & Peaks . . . 5
1.6 Extrapolate Line . . . 6

2 Lane Prediction **6**

2.1 Calculate vanishing Point . . . 6
2.2 Buffer of Vanishing Points . . . 7

3 Results, Discussions & Future Work **7**

1 Lane Detection

1.1 Noise Removal

First, image smoothing was performed to reduce noise reduction. Denoising the image ensures that any future image processing is not hampered by the noise in the image. A standard Gaussian Filter was used with the default sigma value of 0.5 using the MATLAB function **imgaussfilt3()**. Below is a comparison of before and after filtering:

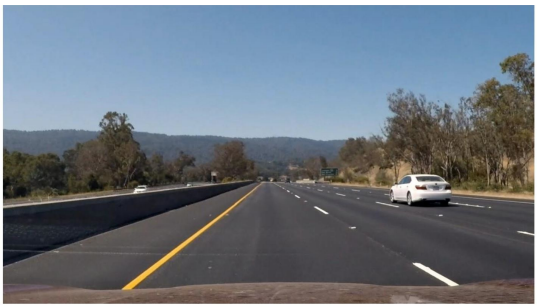

Figure 1.1: Before Filter

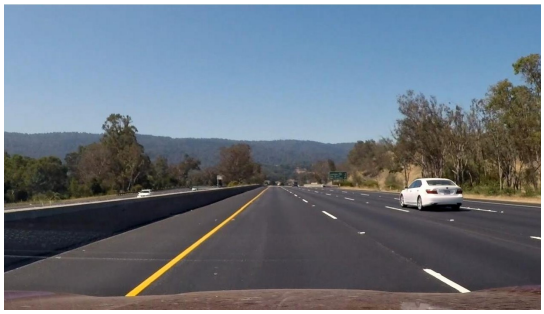

Figure 1.2: After Filter

1.2 Conversion to Grayscale & Thresholding

Next, we take the filtered RGB image and convert it to a grayscale image using the MATLAB function **rgb2gray()**. Once we have grayscale image, we perform thresholding to create a binary image. This is done as the input to the edge detector needs to be an intensity image or a binary image. We set any value above pixel intensity of 180 (value obtained from trial and error) to 0(white). Thresholding especially helps in the sections of the road made of concrete.

Figure 1.3: Grayscale Image

Figure 1.4: After Thresholding

1.3 Edge Detection

After the initial image processing, we are now ready to find edges in our image. There are several robust edge detector implementations obtainable in MATLAB but we choose the Sobel Edge Detector[3]. The advantage of using Sobel Operator, besides the fact that it is computationally inexpensive, is that it can look for edges in a specified direction. This comes in handy to us since our lanes are vertically oriented in the image. The MATLAB function used was **edge()**, the operator type was *sobel*, the direction field was set to *vertical* and the threshold was set to *0.025* (obtained by trial and error).

Figure 1.5: Sobel Edge Detector Output

1.4 Region of Interest (Masking)

Applying edge detection on the entire image results in several extraneous edges which may may hamper the lane detection when we later apply the Hough Transform. To deal with this we select a polygonal region right in front of the car as that's the region which we are interested in. We create a mask such that all the values within this are set to 0 and all values outside it are set to 1. The MATLAB function used was **roipoly** which takes in the x and y coordinates of the desired ROI (manually) and also the dimension of the image (720x1280).

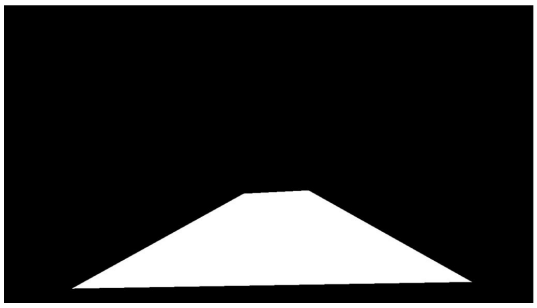

Figure 1.6: Region of Interest

Now we apply this mask on our edge detection output to obtain edges only in our region of interest. Following, this we obtain edges only in our region of interest.

Figure 1.7: Output after Masking

1.5 Hough Lines & Peaks

Once we have our edges in the region of interest we can go ahead and find lines in that area using the Hough Transform[4]. The corresponding MATLAB function is **hough()**. It takes in the binary image as input and returns the ρ and θ values for all the lines obtained in our ROI and also the Hough matrix containing the votes received for each possible line on the image. Next we use the MATLAB function **houghpeaks()** this uses the Hough matrix previously obtained to get the peaks in the Hough matrix i.e., the top ranked lines. For our purposes we set the number of outputted lines to 10. Finally, we use the MATLAB function **houghlines()** to obtain the line segments of the top 10 peaks previously found.

Now we divide the lines based on if the θ value is negative or positive. These are our candidates for left and right lanes respectively. Once we have segregated the lines based on their slopes, we find the longest line with a negative slope and positive slope respectively. These will our left and right lane for the particular image.

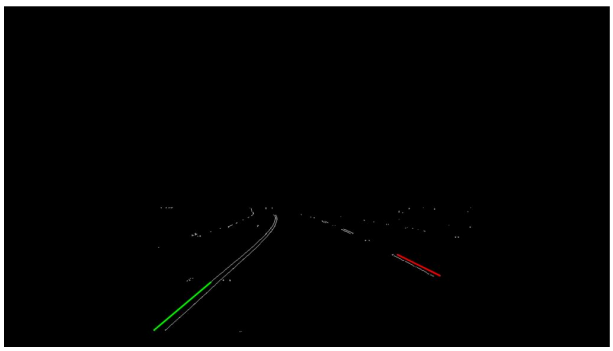

Figure 1.8: Final left and right lane

1.6 Extrapolate Line

Using the endpoints of each line and their respective slopes, we use simple geometry to obtain the equations of each line. Consequently, we extend the line to the desired length by substituting the desired y coordinate and obtaining the resultant x coordinate by using the line equations of left and right lane candidates respectively. Now we use these updated coordinates to plot the lanes onto the original RGB image. We also use these new coordinates as endpoints of the polygon which recognizes the lane in front of us. We plot this region using the **patch()** function of MATLAB.

Figure 1.9: Line extrapolation and highlighting lane

2 Lane Prediction

2.1 Calculate vanishing Point

We utilize the concept of vanishing point to predict if the lane is curving towards right or left, or going straight. We calculate the vanishing point by finding the point of intersection of our two lanes. Note that we only need the x coordinate of the vanishing point. This is done again by using simple geometry. By finding the equations of lines in point-slope form and equating them to obtain x. Our strategy will be to analyze the position of the vanishing point of the image. If the vanishing point is close to the center of the image then the road ahead is straight, if it is on the left half on the image the road is curving right and if it is on the right half of the image the road is curving left. There's of course some thresholding to be applied as the car might not always be traveling at the center of the road. After trial and error, the following scheme was developed:

```
if vanishingPoint > 530
    lanePrediction = "Right Curve Ahead"
elseif vanishingPoint < 515
    lanePrediction = "Left Curve Ahead"
else
    lanePrediction = "Straight Ahead"
end
```

2.2 Buffer of Vanishing Points

Simply using the vanishing point calculated at each frame to do lane prediction may not always give desirable results as there maybe some frames where the lane prediction is a little tilted which hampers the calculation of vanishing point. To do this, a circular buffer was implemented which stores the vanishing points of the last 25 frames and calculates their average. This average value of vanishing points is then used to predict the direction of the lane. However, lane prediction on the patch where the road is not dark takes a hit.

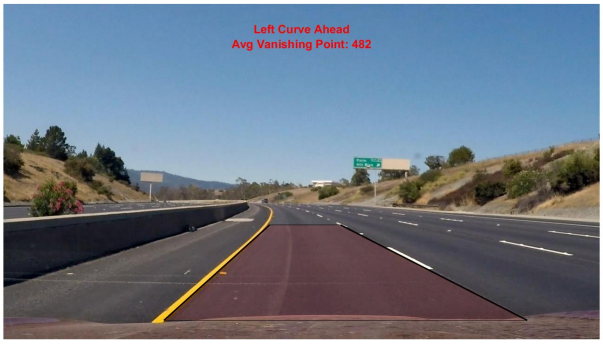

Figure 2.1: Final output of a frame

3 Results, Discussions & Future Work

The lane prediction and detection methodology described in the previous sections was applied to each frame. These output frames were stitched together to form the final output video.

The use of thresholding after initially converting the RGB image to grayscale proved to be very beneficial as it enabled edge detection on frames where the road was made of concrete.

Figure 3.1: Edges without thresholding

Figure 3.2: Edges after thresholding

The implementation of a buffer of vanishing points to calculate the average vanishing point over the last 25 frames did a significantly better job of predicting lanes.

The usage of the longest lane on each side Vs fitting a polygon on all the obtained left and right candidates can be further evaluated. We have used the former. One advantage of employing this technique is that it makes the calculation of the vanishing point much easier.

Finally, the present scheme can be extended to find curved lanes by using the constrained Hough Transform for curve detection. Working in a different color space like (HSV or L*A*B) may aid with with edge detection on the concrete patches of the road. These are left to be explored as future scope of this project.

REFERENCES

[1] MATLAB, *version 9.3.0 (R2017b)*. Natick, Massachusetts: The MathWorks Inc., 2017.

[2] Udacity, "self-driving-car." `https://github.com/udacity/self-driving-car/`, 2018.

[3] I. Sobel, "An isotropic 3× 3 image gradient operator," *Machine vision for three-dimensional scenes*, pp. 376–379, 1990.

[4] P. V. Hough, "Method and means for recognizing complex patterns," Dec. 18 1962. US Patent 3,069,654.