Udacity Self-Driving Car Nanodegree
Term One: **Computer Vision & Deep Learning**
Project One: **Finding Lane Lines**

Nishan Srishankar
Worcester Polytechnic Institute

2017

1 Introduction

One of the fundamental aspects in implementing a self-driving car is perception and deals with how a car would identify aspects of interest that are necessary for its functioning or could impede its performance. This could include traffic signs, signals, neighboring vehicles, lane lines, and even chaotically behaving humans or cyclists. This project deals with identifying lane lines on a highway, and overlaying necessary information on this line.

2 Methodology & Implementation

This section deals with the algorithm utilized for simple vision and perception of lane lines. For simplification, this algorithm is tested on a highway with minimal traffic (no bumper-to-bumper traffic), on roads with little curvature or incline during daytime (with high levels of ambient light). The algorithm is tested on some of these edge cases later on to identify possible improvements later on. One of the main aspects that had to be taken into consideration were the different lane colors i.e. both yellow and white lane lines of varying thicknesses/lengths had to be detected and marked.

2.1 Algorithm

The main dependency of this algorithm is OpenCV with basic dependencies on numpy, matplotlib, and math for array manipulation and plotting images. The techniques used for this project are Color-Palette selection, Canny Edge Detection, Region-of-Interest selection, and Houghline feature transformation.

2.1.1 Color Plane representation

An initial 3-channel image is used as an input and is taken from the perspective of camera placed on the hood of the car. The Red-Green-Blue color image is then changed into a different colorspace. The different, valid colorspaces were either grayscale, HSV (Hue-Saturation-Value) or HSL (Hue-Saturation-Luminosity). The last two are cylindrical coordinate representations of an RGB color model as seen in Figures 1,2

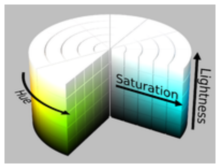

Figure 1: Hue-Saturation-Lightness polar representation

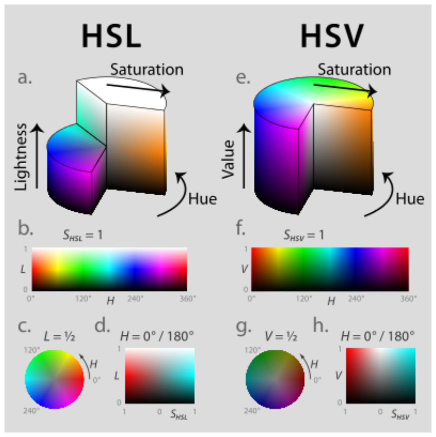

Figure 2: HSL and HSV comparison

Colorspace Choice Grayscale converts a RGB image to a single channel image where each pixel can take a value between 0 to 255. HSV, and HSL colorpsaces were explored as it allows better classification of yellow lane lines, while white lane lines can be decently detected using grayscale image itself. HSL seems to have a better classification than HSV for yellow lanes compared to the surrounding asphalt as can be seen in Figures 4, **??**.

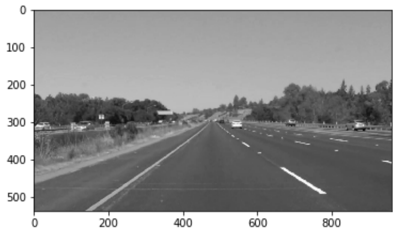

Figure 3: Grayscale Image of a snap that consists of yellow and white lanes

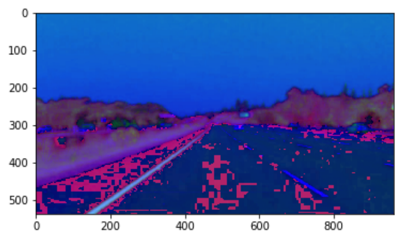

Figure 4: HSV Image of a snap that consists of yellow and white lanes

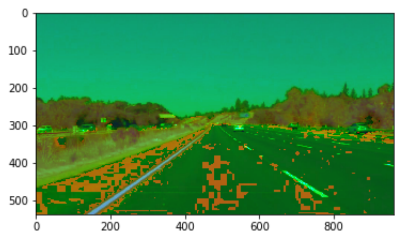

Figure 5: HLS Image of a snap that consists of yellow and white lanes

Blurring The converted image then is smoothed/blurred using a gaussian filter to eliminate noise. Noise is random jumps in in intensity values compared to neighbors, and can mean that a pixel could be mistakenly classified as an edge. The kernel chosen for this needs to be positive and odd and a large kernel size means that potentially important information is lost during heavy blurring.

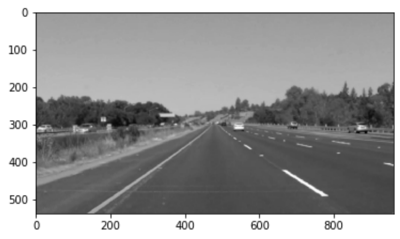

Figure 6: Gaussian Blur with a 5*5 square kernel

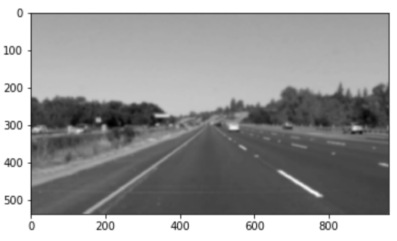

Figure 7: Gaussian Blur with a 15*15 square kernel

2.1.2 Thresholding & Feature extraction

Once an image is converted to the needed colorspace and blurred, it needs to undergo further processing to obtain the necessary lanelines.

Edge Detection This is done by Canny-edge detection which is a multi-stage algorithm to detect color gradients between a specified threshold.

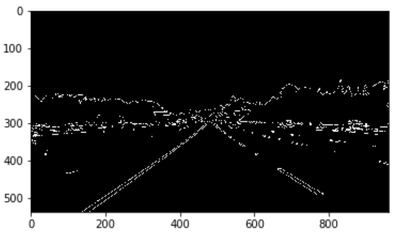

Figure 8: Canny Edge detection on a grayscale image

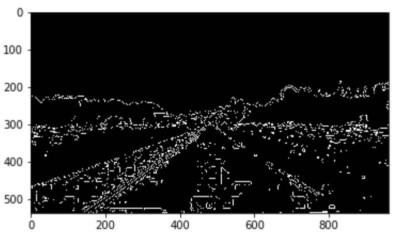

Figure 9: Canny Edge detection on a HLS image

Pixels with gradients higher than the upper threshold is chosen as an edge. Pixels with gradients lower than the lower threshold is rejected as an edge. Pixels that have a gradient between the lower and upper thresholds are only selected if these are neighbors with pixels with gradients above the higher threshold. As can be seen in Figures 8 and 9, the HLS image detects a lot of unwanted gradient-edges that need to be removed later on.

Region of Interest Despite the above processing, unnecessary edges on outlying regions (skies, surrounding land that isn't a highway) doesn't need to be considered as this might throw erroneous data. As the camera is mounted in the same location, and the highway is usually in the same location in an image, a region of interest can be defined to focus further processing. A trapezoidal region-of-interest is selected on the center lower-half of the image as can be seen in Figure 11 as there is greater focus on the region than a triangular mask in Figure 10.
The region of interest mask is applied to the complete canny image, to now output coordinate points localized to the highway.

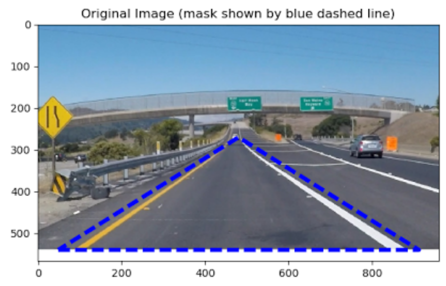

Figure 10: Overlay of Triangular Region of Interest

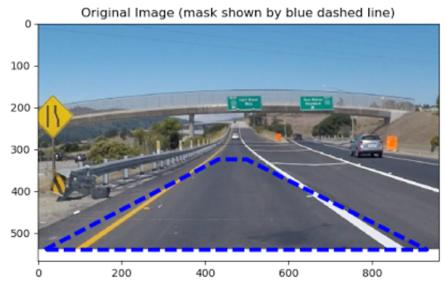

Figure 11: Overlay of Trapezoidal Region of Interest

Feature Extraction Following the masking, features (coordinate points) are extracted and converted into line segments. This s done using a hough transform where each input measurement contributes to a globally consistent solution.
The resolution of the transformation is set using distance and angle resolutions in pixels and radians respectively. Accurate houghlines are obtained by tweaking parameters that control the minimum number of votes in a cell for a candidate line, the minimum line length (to reject segments shorter than a given pixel length), the maximum line gap (to connect points on the same line).

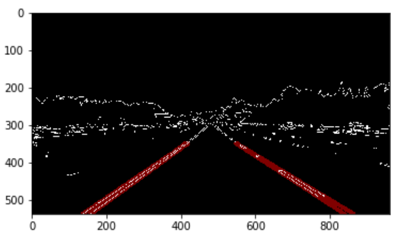

Figure 12: Displayed Hough features

Layering lines The houghlines needs to be layered onto a blank image properly. The return of feature extraction is two sets of coordinates (x,y) which represent endpoints of a line. A valid coordinate set for drawing is one that doesn't represent a vertical line or one in which the absolute value of the gradient is above a certain threshold. Using this, and comparing x_1 and x_2 to the x-midpoint of the image will ensure that the lines can be properly distinguished into left and right lanes.

3 Results

A lot of time and effort was spent in tuning parameters for every aspect of the algorithm. The simplistic algorithm above works very well for each of the images, and the two test videos. In the challenge video, it starts to fall apart, however still functions. The algorithm was tested on a night-time driving sample video however it was unable to detect any lane lines due to extremely low ambient lights, blinding street lights, and headlights over-illuminating lane lines. The outputs for each of the test images are shown below.

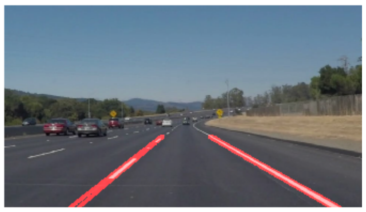

Figure 13: Solid White Curve

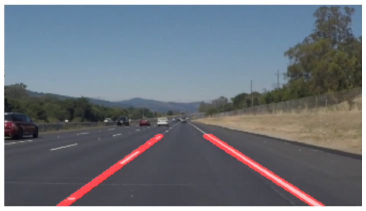

Figure 14: Solid White Right

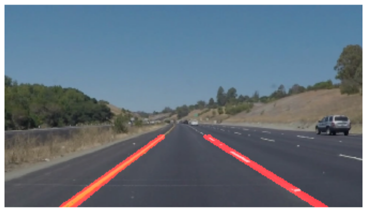

Figure 15: Solid Yellow Curve

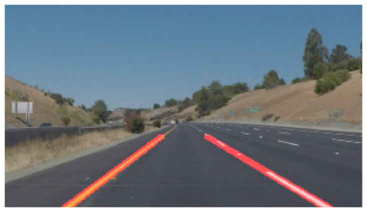

Figure 16: Solid Yellow Curve II

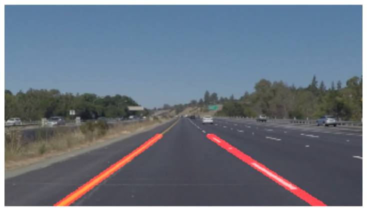

Figure 17: Solid Yellow Left

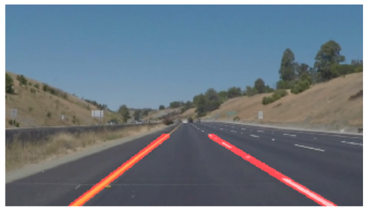

Figure 18: White Car Lane-Switch

The obtained videos can be seen on Youtube.

4 Conclusion & Further Improvements

The algorithm is a step in the right direction to detect lane lines, however, it can definitely be improved much more. There can definitely be a more optimum selection of parameters that have not been explored in this project.

This algorithm will fail in the case of low light, heavy snow/rain conditions, as well as moderate-to-significant levels of traffic. In addition to this, should the car drive on a steeply-curving or a steeply-inclined lane, the algorithm would start to glitch. As in the test challenge case, significant lane-markings or color-gradients on the road would cause a false positive. Testing on other roads that are not highways should also be done, as the region of interest would now be slightly different.