Finding Lane Lines on the Road

The goals / steps of this project are the following:

- Make a pipeline that finds lane lines on the road
- Reflect on your work in a written report

Pipeline Description

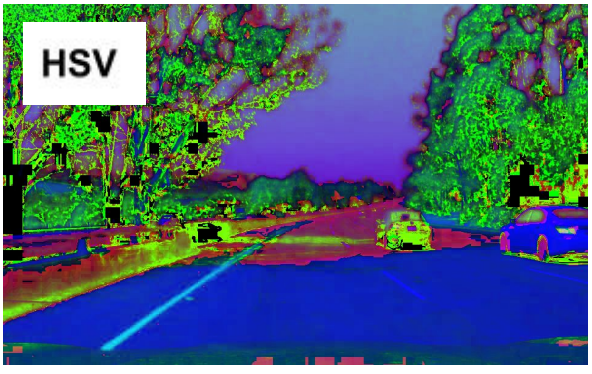
My pipeline consisted of a few steps and they are described as below:

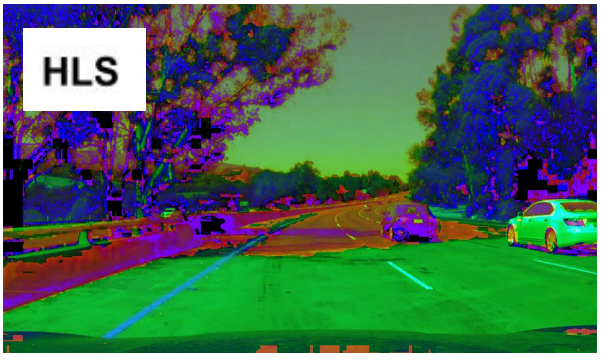
Step 1:

First, I imported all images from the folder “/test_images”.

Step 2:

Then, I converted the images into HLS scale and I will explain it after this. Taking the image “challenge.jpg” as example, results of RGB, HSV and HLS, also their respective white and yellow masks are shown.





Comparing HSV and HLS, we can clearly see that the white lines in HSV image (right lines) are almost blended with the color of lane. The upper right lines are completely indistinguishable as well. Therefore, I chose to not use HSV.

As for potential of using RGB to detect the lane lines, it is possible. However, I chose to not use it due to the fact that I tried with RGB and it was tough to filter the lane lines consistently. Hence, I proceeded with HLS.

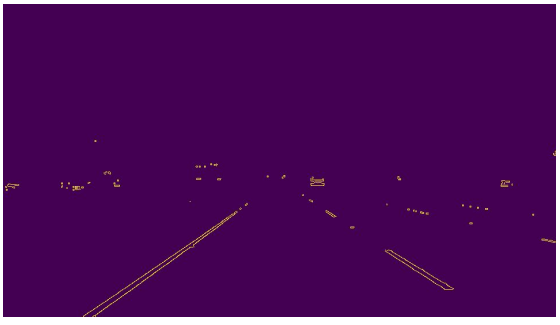
Next, I made a color mask of yellow and white color on lanes using the HLS image. For white, I used lower bound HLS of (0, 210, 0) and upper bound of (255, 255, 255). As for yellow, lower and upper bounds were (20, 0, 110) and (45, 255, 255). Here is the result.

Step 3:

After that, I first converted the color mask to grayscale. The grayscale image can be seen on the left.

Then, I proceeded with Gaussian smoothing. This was to smoothen out the random dots that occured around the lines.

I used a kernel size of 11 for my Gaussian filter.



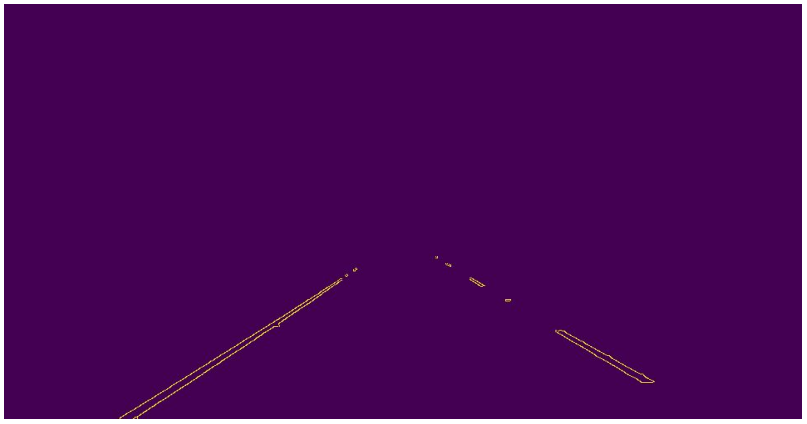
Next, I detected the edges using Canny edges method. My low threshold and high threshold for Canny edges detection were 100 and 200 respectively.

Step 4:

I created my region of interest (ROI) which helped me to only capture what I needed from the road. I made a polygon shaped ROI with vertices depending on the size of image. The vertices consisted of four points or coordinates on the image.

Taking x and y size of the image, my left and right bottom points were (0.1x, y), (0.9x, y). And my left and right top apexes were ($\frac{4}{9}$ x, 0.6y) and ($\frac{5}{9}$ x, 0.6y).

Note that only the Canny edges detected inside this ROI were taken for further processing, rest were eliminated.



Step 5:

Next up, I did a Hough Transform on this ROI. Technically, I used Probabilistic Hough Transform. The parameters used for Hough Transform are:

1. rho (p): 2
2. theta (θ): pi/ 360
3. threshold: 30
4. min_line_length: 40
5. max_line_gap: 80

Step 6:

Moving on, I collected all the hough lines and calculated the slope, m and y-intercept, c for each of the lines. Also, I classify them into left or right lines.

The Hough lines actually consisted of two coordinates, which were the first and last coordinates that made up the each of the lines. How did I actually do that is I was comparing the first x coordinate, x1 collected from each line.

We know that the top left corner is our origin (0, 0), as the hough transform scanner went from left to right (started from 0 until the end on x-axis), it passed through x1 of each line before meeting the second x coordinate of the line.

So, we can understand that if x1 is less than the middle of image horizontal size, it is a left line and otherwise. In addition to that, I also calculated their line length and it was used for averaging all the lines after this.

Other than that, I created interquartile range for each of the lines. I only considered x1 in this case as it is a robust value to rely on. The first and third quartile were made to be 25 and 75. Hence, any lines with their x1 out of this range were eliminated.

After that, I made a dot product of all the slope and y-intercept of each of the left or right lines with their respective line length. This was then divided by the total of the lines length. This can help to average out the lines by making the lines with longer length to be more significant. The results of this were the average slope and y-intercept for left and right lines.

Step 7:

After all the processing, it was time to draw the lines. To draw this, first I made the maximum and minimum height that I wanted to draw. They are y_max = image vertical size, and y_min = 0.6*image vertical size.

Then, I collected the average slope and y-intercept that calculated from the previous step. There were three cases to be considered when the car is moving.

1. Case 1: Car is turning left

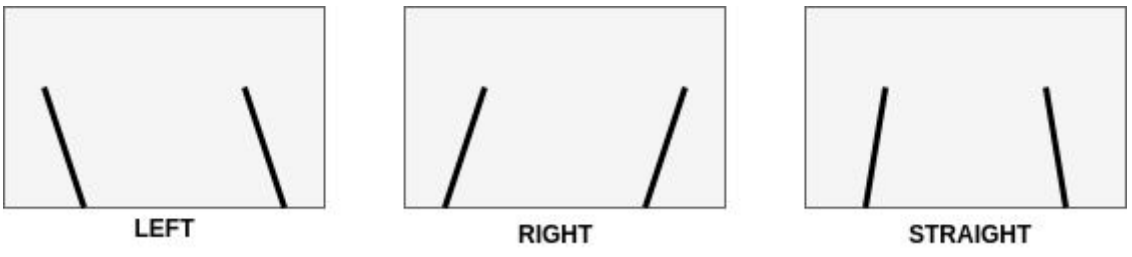
   In this case, its left slope has to be positive and the same goes to right slope.

2. Case 2: Car is turning right

   In this case, its left slope has to be negative and the same goes to right slope.

3. Case 3: Car is moving straight

   In this case, its left slope has to be negative and right slope has to be positive

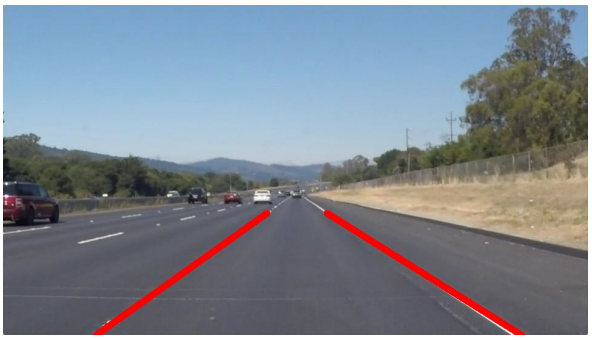


Next, the x coordinates at y_max and y_min were calculated with x = (y - c)/ m, derived from the formula, y = mx + c, where m is the slope and c is the y-intercept.

Then, the (x, y) were all gathered, and lines were drawn with red colour (255, 0, 0) with thickness of 10. And all images were processed with the pipeline, shown below.








Reflection

Potential Shortcomings

1. One potential shortcoming would be what would happen when there are a lot of shadows covering the lane or lines.
2. Another shortcoming could be the imperfect lane conditions for example, faded lines and random dots or objects present on the lane.
3. Last shortcoming would be the time when driver is making a sharp turn.

Possible Improvements

1. One possible improvement would be implementing a more robust outliers detection algorithms as outliers greatly affect the outcome of my pipeline.
2. Also, implement a cache to memorize the last few lines of slope so it could be used to eliminate the current lines with abrupt change of slope, the outliers.
3. Instead of only using the first and last coordinates of Hough lines for processing, consider their median also.
4. Instead of averaging the lines with their length, averaging with the number of points touches the lines could be a good approach.
5. Drawing the lines with maximum position based on the changing of slope.
6. Train a CNN model to detect lane lines instead of solely depending on color-based detection.

Credits:

1. Naokishibuya - https://github.com/naokishibuya/car-finding-lane-lines
   The repo inspired me to try with HLS instead of HSV which is what I did at first. Also, I learnt the idea of averaging using lines length.