Finding Lane Lines on a Road

This write up explains the approach used to code a lane finding program and compares it with the objective rubric.

The write includes:

1. *Project Approach and Reflection*
2. *Pipeline Shortcomings*
3. *Possible improvements*

Project Approach

The project required a program that could detect lane lines in different road images, road videos and annotate the road lanes with a continuous straight line. I created a function 'detect_lane_lines' that has the following functions:

1. *Load the image or video*
2. *Detect white, yellow pixels in the image and mask other colored pixels*
3. *Convert the image to grayscale and apply Gaussian Blur using kernel = 3*
4. *Detect edge pixels in the image using Canny Edge Detection*
5. *Draw the region of interest and mask all other pixels not inside the region*
6. *Apply hough transform to detect pixels that form a noticeable continuous line and color the lines red*
7. *Average and extrapolate the lines to get a single solid lane marking line*
8. *Tune the canny edge detection and hough transform parameters to coincide the marking lines with the actual lane lines*

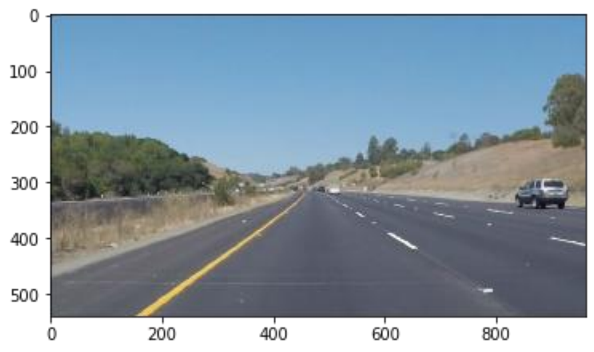
Figure 1: Original Image

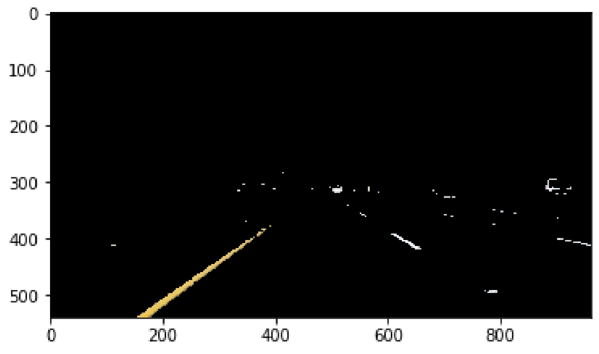
Figure 2: Masked image with white and yellow pixels

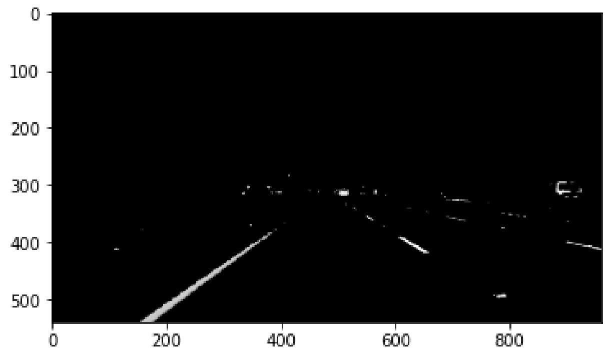
Figure 3: Grayscale image with Gaussian Blur

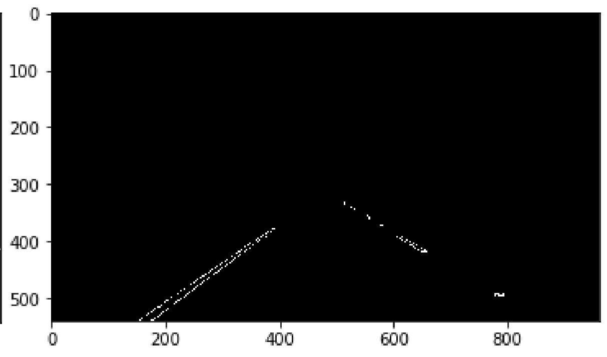
Figure 4: Canny Edge Detection with Region Masking

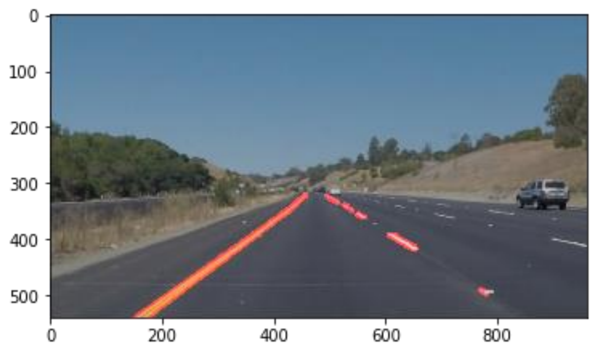

Figure 5: Lane lines after Hough Transform

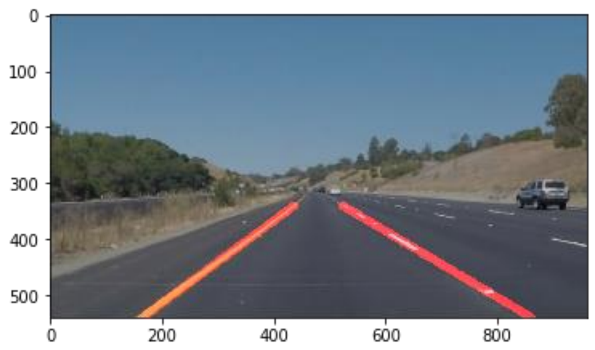

Figure 6: Average & Extrapolated Lane Lines

Draw_lines() function
I modified the draw_lines() function to separate the left, right hough lines, to sort the hough lines based on their slope; and to average, extrapolate the hough lines to get single, continuous right, left lanes. The function has the following steps:

- a. *Initialize the lowest and the highest y co-ordinate value*
- b. *Calculate the gradient of a hough line*
- c. *Compare the y co-ordinates of the line to stored low y value*
- d. *Update the stored value to the smallest of the three values: 2 y co-ordinates and 1 stored low y value*
- e. *Reject the line if the gradient of the line is less than a threshold to avoid horizontal lines and store the gradient if it is above the threshold*
- f. *Sort the stored line into left and right lane lines based on its gradient and x position relative to the center of the image*
- g. *Repeat steps b, c, d, e, f until all lines in the image are sorted*
- h. *Calculate the average gradient, x and y co-ordinates for both left and right lane lines*
- i. *Use the mean values to calculate the intercepts of the lines*
- j. *Solve the equation of the lines to get the x co-ordinates of the end points of the lines*
- k. *Use these x values along with the 2 y values to draw a single line in red*

Detect_colors() function
This function was designed to optimize the detect_lane_lines() function to detect lines in the challenge output. Due to a lot of gradient changes close to the lane lines caused by other colors, it was necessary to separate the yellow and white colors from the image to stabilize the canny edge detection, hough transform and draw_lines() function.

Pipeline Shortcomings
The pipeline was designed with a specific output based on the given images, videos and was not tested on several instances that could arise in the real world when it comes to detecting lane lines. The pipeline will not function under diminished visibility due to bad weather, road with changing elevation, curved hilly roads or if the curvature of the road is large enough to not form a single continuous line either by parts of the line under the gradient threshold or parts of the left line being on the right half of the image. The detect_colors() function is designed just for natural sunlight or white light. It will not work under any other shades of light or in low light conditions such as night. The detect_colors() function might cause some discrepancies if a white or a yellow car is very close in front of the image. The lane lines might not be detected if the lanes move out of the region of interest due to vehicle movement or if the lanes are faded, missing.

Possible Improvements

The pipeline is very rudimentary and can be made more robust to tackle every anomaly discussed above. The detect_colors() function could be more robust to concentrate in the region of the lanes and to adjust to light conditions. Under bad weather conditions, it should work with other sensors to drive through a safe path. The draw_lines() function can be modified to draw higher order functions based on the curvature of the lane lines. The lane lines in conjunction with cars in front and around if possible could be used to detect a region in which the vehicle is allowed to travel. In retrospect, rather than concentrating on a particular region of interest or color masking, the program should be able to separate different sections of the image using canny edge detection, hough transform; detect features of each section such as color, position, shape, etc and classify to detect just the lane lines.

Rubric	***Progress***
Include Project Files with Submission	*Done*
A pipeline for line identification that takes road images from a video as input and returns an annotated video stream as output	*Done*
A pipeline been that uses the helper functions and / or other code to roughly identify the left and right lane lines with either line segments or solid lines	*Done*
Filter / Average / Extrapolate detected lines to map out the full extent of the left and right lane boundaries	*Done*
Reflection on the project	*Done*