** Finding Lane Lines on the Road **

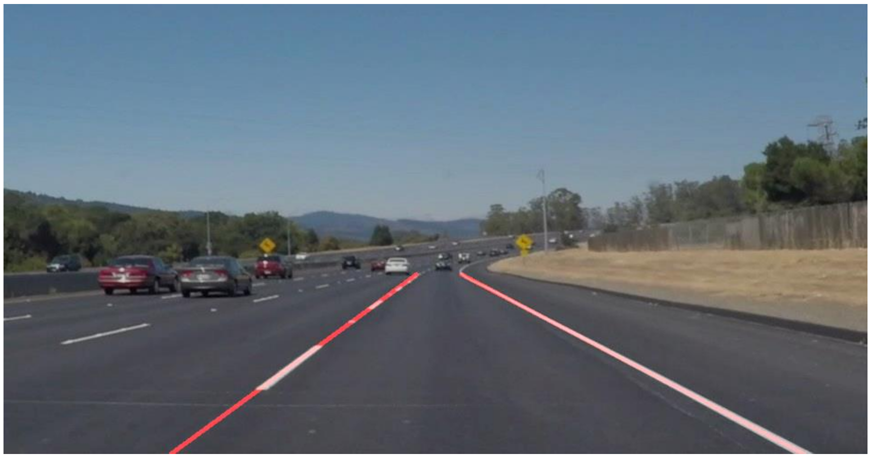
Pipeline Description:

My pipeline includes the following steps.

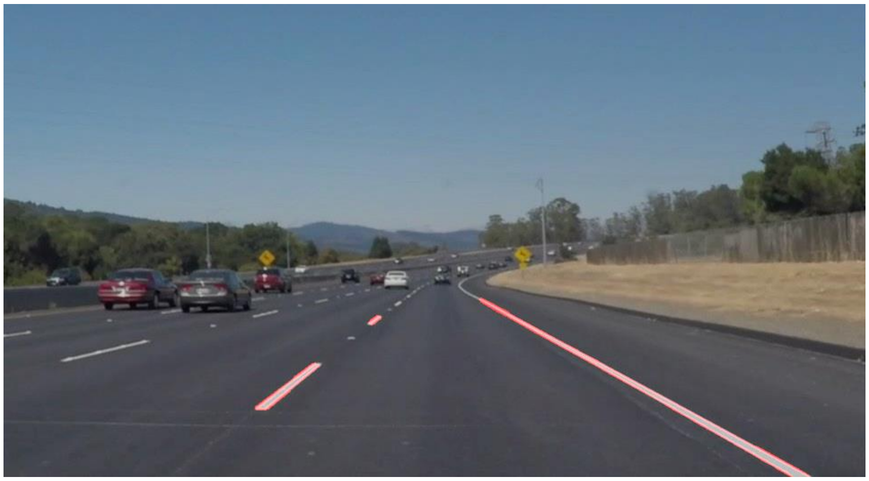
1. Get the image input to which we need to find the lane lines.
2. Use select_colors method:
    a. It takes the image as input and returns the another image which selects only the white and yellow pixels and black out everything else.
    b. Define the lower and upper range for white color and then apply masking using cv2.inRange method.
    c. Convert the image to get the hsv format of the image to identify yellow color.
    d. Define the lower and upper range for yellow color and then apply mask as we did for white color.
    e. Combine both the white and yellow masked image and return it to further processing of the pipeline.
3. Convert the image to grayscale.
4. Define a kernel size and apply Gaussian smoothing.
5. Define the parameters for Canny and apply Canny transformation on the output of the Guassian smoothing image.
6. Define Vertices of the polygon as numpy array for region selection of the lane line.
    a. In my case, I have used a quadrilateral.
    b. I have chosen the bottom post two points for region selection.
    c. The top two points was chosen with the view of focusing on the lane lines upto 40% of the overall length of the sight of the image from the car and restricting it to particular lane in which the car is running.
7. Using cv2 fillpoly and bitwise_and methods, mask the region of interest.
8. Define the Hough transform parameters and apply them on the region selected image.
9. Use HoughLinesP method to find the line coordinates and then pass all the lines to draw_lines function
10. I have done the following in draw_lines function:
    a. Before changing the draw_lines functions, the final output will be different. It will have raw lines. I have attached both the images in the end.
    b. Find the lines which are right and left.
    c. Calculate the slope and compare the current value of x with that of the mid value of the x in the image.
    d. Base on the positive or negative values and greater or lesser values of x, separate the left and right lane lines.
    e. Find the m and b values for the right and left lines that we have identified using polyfit function of numpy.
    f. As I did in the region masking, I now repeat the same. I chose the y boundary values of the lines to bottom of the image and 40% of the image from bottom.
    g. We then find all other x values using the formula x = (y – b)/m derived from y = mx + b.

h. Now we draw left and right lane line using the open cv line function.

11. After drawing the lane lines, combine that line with the original image to get the lane lines marked on the original image which is inputted.
12. Return the final image.
13. The final image looks something like below.



Lane Lines Extrapolated



Lane Lines – Raw Lines Before extrapolating

Potential Shortcomings of the Pipeline:

The pipeline holds good for most of the cases but not all the cases. For example,

1. Too much curvy roads, where the road has too much twists and turns.
2. When the Lane markings are not very clear or the color used is other than yellow and white.
3. If we are travelling over hills which has ups and downs.

Possible Improvements to the Pipeline:

The possible improvements can be as follows,

1. For tackling the curvy roads, instead of fixing the lane line finding region to 40% of the original image, we should formulate some algorithm which handles the curve and identifies it pretty well.
2. We can collect and use data from gps to foresee the road structure and compare it with our visuals and combine both these data to arrive at a robust image to find the lanes pretty well.
3. We can classify the road types based on the geographical location and difficulty of manual driving in those roads and create a separate pipelines for each of those to tackle location specific problems like ups and downs in hills.