Finding Lane Lines on the Road

Writeup Template

Finding Lane Lines on the Road

One of the major thing that we want our self-driving car to have the sense that driver has one while driving the car. One of them being seeing what driver sees. The goal of my very first project in Self-driving car Nanodegree program were the following:

- Detect the lane lines on the road on the series of images that was captured by the front camera of the car.
- To apply the above algorithm for the series of images to the video.

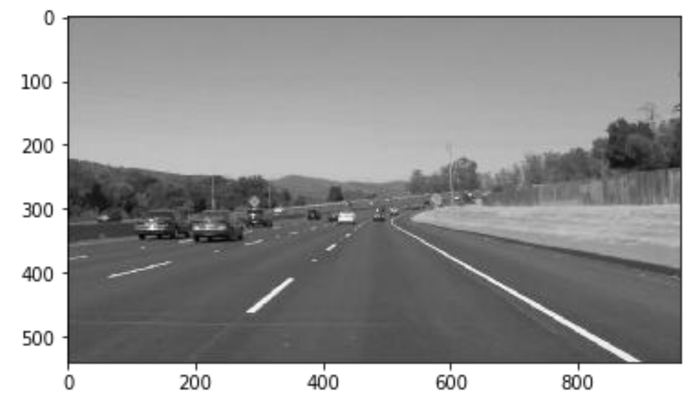
To accomplish this project, we would be using Python, OpenCV package.

1.Pipeline of the Project:

Step One – Grayscale conversion

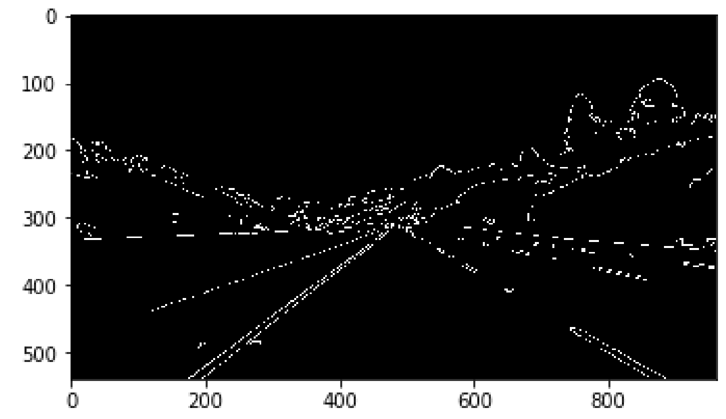
First step was to convert the given series of images to the grayscale. This makes easier for the canny edge algorithm to detect the edges of the lane lines based on the threshold.

We can also apply gaussian blur using kernel size to the grayscale image , if there is a lot of sharp zig-zag edges on the conversion to the grayscale. It is done to detect the lane by eliminating extra noise. Keeping in mind that kernel size should be an odd number.



Step Two – Canny Edge Detection

Using the cv2 package, we can apply the canny edge detection to the output of the gaussian blur function. It is used to detect the edges using the edge gradient and direction for each pixel. It takes in lower threshold and higher threshold as the argument too. Threshold that I used was lower_thres = 50 and higher_thres = 150. It will discard the pixel range below lower_thres and then will look at all the pixels between 50 and 150.



Step Three – Region Masking

This is done in order to discard the object that are not of our interest in the image. I implemented it using a quadrilateral.

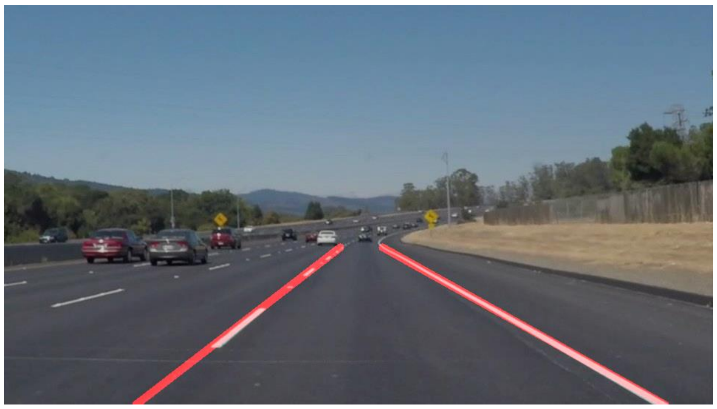
Step Four – Hough Line Transform

I used Hough Line transform function to find the point where maximum lines are intersecting which will give me the line segment that I am interested in. I applied hough transform on the masked edges so that it output the lines with minimum number of intersection as 22 in the hough grid cell, minimum number of pixel that would make up the line as 20 and maximum gap in pixels between line segments as 1.

Step Four – Draw the lines

In order to draw the lines, I defined the draw_lines_avg() in the helper functions. I implemented the following logic in that function:

- Calculated the slope and intercept for each line in the hough output and sorted left and right lane based on the slope. I also calculated the length between the points on each line so that it could be used as weight while averaging out the slope and intercept.
- Next step is averaging out the slope and intercept.
- With y coordinates from the region of interest, I calculated the x coordinates using the mean slope and intercept with a simple one degree polynomial y = mx+b.
- With the calculated x and y points , I drew the line using cv2.line().



2. Identify potential shortcomings with your current pipeline

The current Algorithm successfully detects the straight line. However I tried using the least square method along with ployfit to fit the points in polynomial of degree 2 instead of a linear fit for detecting curve lanes. And it was able to detect the curves but not accurately and also the major issue of vanishing point was showing up constantly which I was not able to eliminate with the above method. So, I have not included the code for that. Also, Currently there are no cars in front of the car in picture in ROI that we have defined which will not be the case in the real world. There might be some car and the ROI will detect those lines too.

3. Suggest possible improvements to your pipeline

I am currently very new to the field of computer vision as I come from the controls system background. Potential improvement would be to find a dynamic ROI as well as a robust algorithm that detects both curved and linear lines.