Finding Lane Lines On The Road.

Reflections:

Finding lane lines on the road is the first step in making self-driving car bot to know its position on the road and restrict itself inside the lane lines region. This task involves applying certain series of image pre-processing software for the desired result. Following software pipeline is being followed for this project.

1. Gray Scaling Image:

   In this step I Gray scaled the image this is important because this indirectly helps to consider only less values for computations like calculating the gradients etc.

2. Gaussian Blur:

   In this step I used the inbuilt function of open “cv2.Gaussianblur” to smoothen the image and this also removed some of the noise that when left without smoothing can make edge detection algorithm to detect these high frequency noise as also edges thereby causing false edge detections. This is by default applied by the Canny algorithm though.

3. Canny Edge Detection:

   In this step I passed the smoothened image to the “cv2.canny”, Open-cv’s inbuilt function for implementing canny algorithm, this algorithm detected edges in the image and returns the respective line segment’s vertices. I experimented with different values for the hysteresis thresholding parameters at the end I got better results with values, low_threshold being 100 and high _threshold being 200.

4. ROI masking:

   In this step I masked the entire image except the polygon region where it is highly likely for lanes to fall in. This helps to specifically apply the

"Hough Transform" to this region. I have experimented different values of vertices for the polygon , at the end I got values that exactly got the region containing the lanes , gave some padding in left and right side of polygon so as to allow any movements in the lane lines beyond their average position.

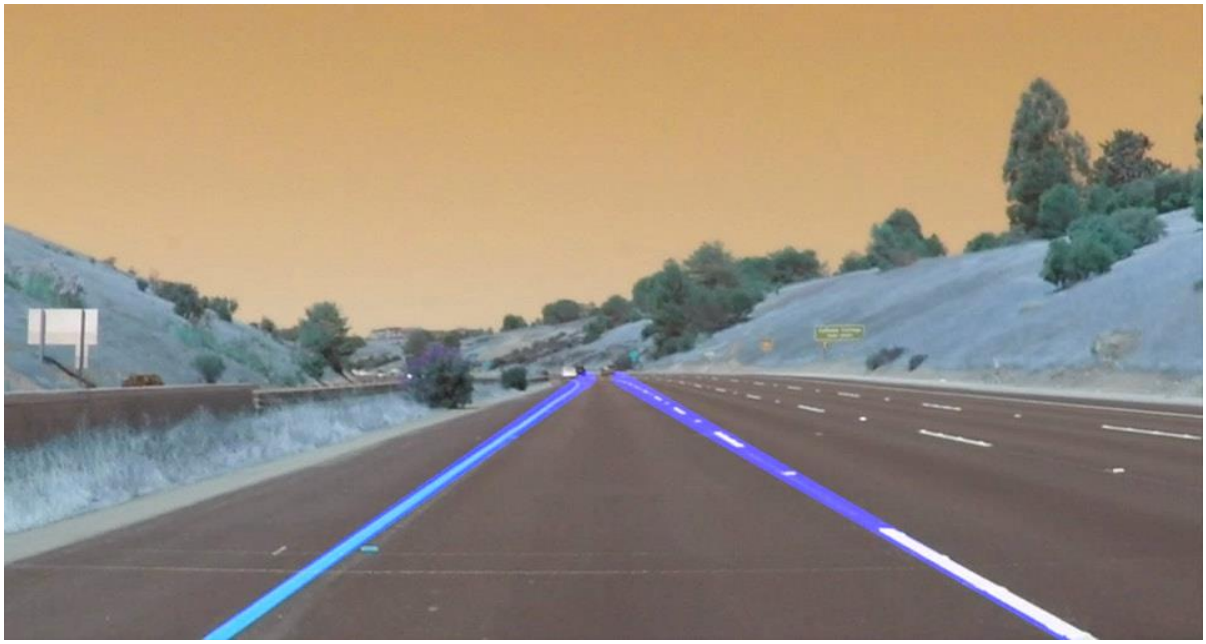
5. Hough Transform:

   In this I performed the Hough transform over the ROI masked image.
   I have experimented different values for the parameters of the Hough transform like grid spacing of rho and theta coordinate system, threshold, etc. This was a difficult task because of the fact that there are more than million combinations of parameters, but I was able to remove most of the possibilities by intuitively thinking about the values parameters take that makes sense and at the end got the parameter values that gave significant output.

6. Blending original image and image that contained lines in ROI:

   In this I blended the original image and image that contained lines in ROI using Open-cv's method "cv2.addweight" this method blends images based on weights. This overlays the lines onto the original image.
   I have experimented with different values of alpha, Beta, Gamma weights for perfect blending.

Above was the pipeline I followed, and the following is the outcome for a certain test image:



For eliminating multiple individual small lines on left and right lanes and to draw only single line respectively, I modified the draw_lines() function .

My very first approach was, I have done a regression type line fitting problem by separately fitting a line for left and right lanes by segregating the points based on the slope of the individual lines given by “Hough Transform” operation. This worked well but this would be computationally expensive task when dealing with a video stream because this operation depends on error calculation and iteratively changing parameters.

My second approach was, to segregate the lines into right lane lines and left lane lines based on the sign of slope calculated. Then also accumulating the respective intercepts and slope values. Then calculated the respective averages of intercepts values and average of slope values for both right and left lines. Then calculated the x values of top and bottom coordinates of respective lines by using the maximum and minimum y coordinate of the vertices of ROI polygon, this restricts the line from being stretched and limits line to a certain portion of the lane only. Then I have used Open-cv’s cv2.line function to draw line segments over image with specified end points.

Potential Short Comings Of This Lane Line Detection Algorithm

The potential shorting coming of my lane line detection algorithm that I would think of are:

1. This doesn't work well for lanes that are at different lighting conditions.

2. When lanes are curved this algorithm fails because this does not use any polynomial fitting to correctly fit the lane curve

3. For this task we assumed that there was no distortion in images, this might cause a false lane detection that would cause the car to be disoriented to other direction and may potentially be leading to accidents.

Possible Improvements to The Pipeline

Some of the possible improvements over the existing pipeline is to overcome above shortcoming by including some additional pre-processing before detecting the lane lines , that includes by adding a software component that does some image pre-processing that can make canny detect the lane lines in different lighting conditions. We can modify the draw_lines() helper method's code to detect curved lane lines using techniques like regression analysis with higher degree of independent variable. By adding a software component that takes in the captured image / frame and removing possible distortions in the images using some computational photography techniques.