SELF DRIVING CAR ENGINEER NANODEGREE / TERM – 1

PROJECT – 1
FINDING LANE LINES ON ROAD

The goals / Steps of this project are as follows:

- Make a Pipeline that finds Lane lines on the road
- Reflect the overall structure of the work in this report

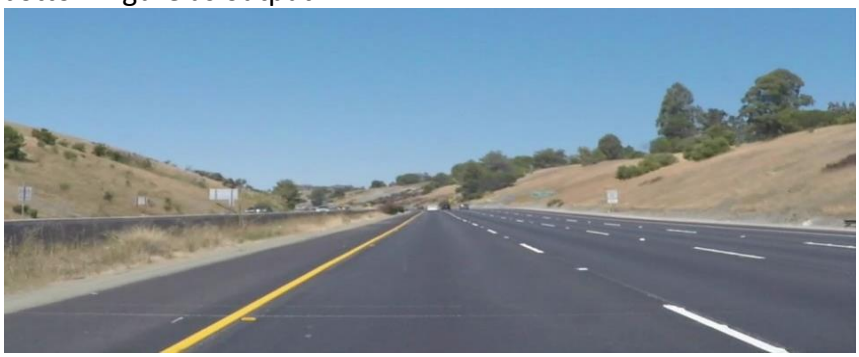
REFLECTION:

PIPELINE DESCRIPTION:

This pipeline consists of 5 steps – to read and process the image or frame of a video clip to detect the lane lines on the road.

1.) Converting the image to grayscale:
   a. First as I read the image of the road with different colored lines – I convert it into gray scale to have an image with single color channel:

   The top figure is the original RGB image and once converted into grayscale we have the bottom figure as output.



2.) Apply a Gaussian mask to filter any noise from the gray scaled image. The size of the noise kernel is defined as an input using cv2 functions defined in the helper functions:

```
def gaussian_blur(img, kernel_size):
    """Applies a Gaussian Noise kernel"""
    return cv2.GaussianBlur(img, (kernel_size, kernel_size), 0)
```

3.) Now for the filtered image I run the canny edge detection function with defined minimum and maximum threshold limits to detect spikes in gradient across pixels and return points distributed over the edges. The following conditions worked fine for the initial part of the logic to detect lane lines.

a. LOW_THRESHOLD = 50 and
   HIGH_THRESHOLD = 150

Output from Canny edge detection algorithm is as follows:

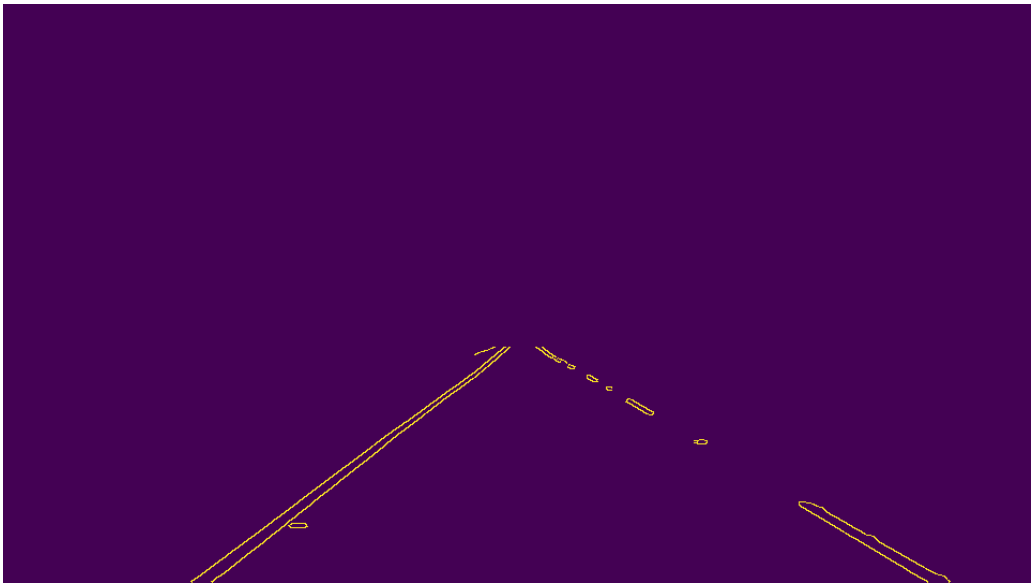
4.) Defining Region of Interest (ROI):

- This function operates on the output of the canny edge detection logic i.e. the "edges".
- There is a blank mask defined with certain color to start of with.
- Now I have defined my region of interest to be a "quadrilateral" with some defined vertices

My vertices are:

```
vertices = np.array([[(140,imshape[0]),(450,320), (520,320), (920,imshape[0])]],
dtype=np.int32)
```

The output of the edge image once this mask is applied is as follows:



5.) Hough Transform:

After the canny transform and choosing our ROI we input that image into our Hough transform algorithm where each point in the image casts their vote – as per our input conditions. The maximum votes casted (greater than the defined threshold) – will determine what in the edge transform image and in my ROI seems to be reasonably qualified as a line.

rho = 1 # distance resolution in pixels of the Hough grid
theta = np.pi/180 # angular resolution in radians of the Hough grid
threshold = 10 # minimum number of votes (intersections in Hough grid cell)
min_line_length = 20 #minimum number of pixels making up a line
max_line_gap = 20 # maximum gap in pixels between connectable line segments

As the hough transform operation is returns all set of qualified lines ([x1, y1, x2, y2]) → these values are used by the draw_lines function to draw onto the image.

Now the other important task finished in this project and is included in the code is –the modifications done to " draw_lines()" function to draw an averaged/extrapolated line on each lane instead of the line segments as shown above.

The following logic was included in the changes done to ***"draw_lines()"*** function:

- The detected lines from the hough transform were used to calculate the slope of each line on each frame of the video test files.
- The lines were classified as right and left side lines based on the slope values. The lines with negative slopes are on the left given the origin of the image is on top left corner and ones with positive slopes are on the right.
- The lines are further filtered based on slope thresholds of [0.3 0.9] for positive & negative slopes. This eliminates any noises in the image which led to the detection of perpendicular and smaller insignificant lines.
- Lines which qualify to be within the slope thresholds are classified and grouped into left and right lanes.
- For each frame I fit a left and right lane using np.polyfit () function.
- The y coordinated for the starting and end of the line on left and right lanes is fixed and the average slopes and intercepts from the fit are used to calculate x coordinated from the y = mx+b realtion.
- Finally the cv2.line() function is used to fit a solid line on both lanes while increasing the thickness of the line.

RESULTS:

Please find the results in the following locations:

- IMAGES: *"test_images_output"*
- VIDEOS: *"test_videos_output"*

POTENTIAL SHORTCOMINGS:

One potential short coming of this pipeline is that it is very much prone to noises and detection of proper lines from each frame to frame.

Another shortcoming could be that the parameters and input values for canny and Hough transforms logic and the filtering on each frame is fixed which might not work well for all frames in an image depending on the light intensity and other conditions.

The ROI defined in my pipeline is also concentrating on the fixed space to search for lane lines which will not work well In a generalized sense on every road and curves.

POSSIBLE IMPROVEMENTS:

One improvement for sure can be done is by using the lane information from previous frames to detect the lane lines in each frame. Now this could be a simple rolling average filter or any adaptive filter which looks at the lines detected in each frame to apply an adaptive gain as a filter constant for the frames.

The ROI and canny thresholds could be improved by looping over the image many times by adjusting the ROI and canny threshold values to come up with best outputs. This could prove computationally a bit more intensive.

I will be working to implement such techniques for challenge video for the future.