Project 1

Finding Lanes on the Road:

It is very important for Autonomous Car to detect the lanes on the road , so that car can stay within lane boundary, in-order avoid collisions. To develop a robust lane finding algorithm that adapt to all whether condition is quite complex. There are some steps involved for simple lane finding algorithm is briefly below.

Steps involved in Finding Lanes:

- Load a test image for manipulation.
- Get the image information like (shape, channel).
- Defining region of interest.
- Convert original image into grayscale.
- Apply gaussian blur to smoothen edges.
- Apply canny edge detection for smoothed gray image.
- Hough Transform to find lanes within our region of interest and trace them in red.
- Separate left group lane and right group lane.
- Pipeline to detect the line segment in the image.
- Annotate video (overlay pipeline processing).

Load a test image for manipulation:

```
import matplotlib.pyplot as plt

import matplotlib.image as mpimg

# reading in an image

image = mpimg.imread('solidWhiteCurve.jpg')

# plotting the image using matplotlib

print(image.shape)

plt.imshow(image)

plt.show()
```

Output:

```
This image is: <class 'numpy.ndarray'> with dimensions: (540, 960, 3)
```

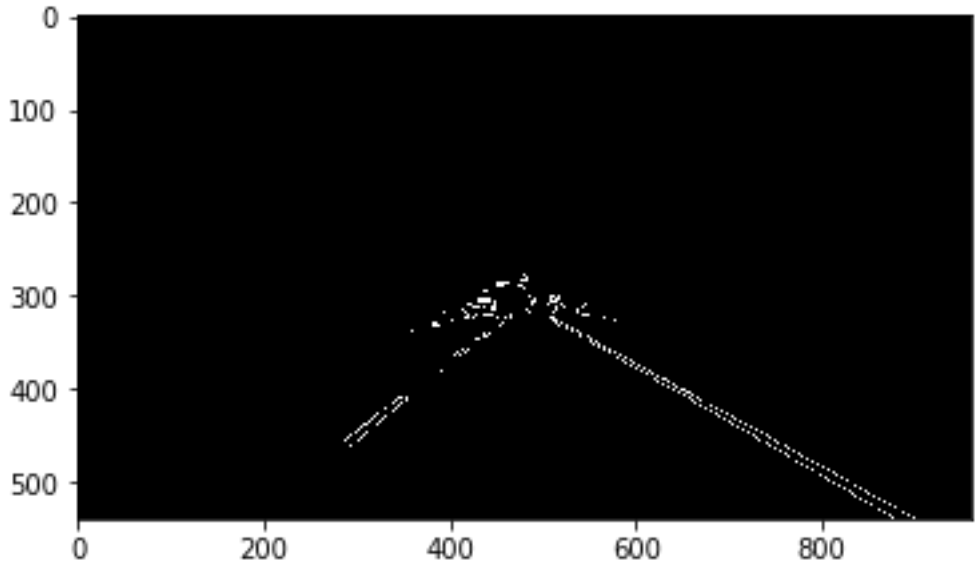
Defining region of interest:

Define a region of interest that fully contains the lane lines. Shape that will achieve this goal is a triangle that begins at the bottom left corner of the image, proceeds to the center of the image and then follows another edge to the bottom right corner of the image.

```
region_of_interest_vertices = [(0, height),(width / 2, height / 2),(width, height)]
```

Applying Grayscale/Gaussian Blur/Canny Edge Detection:

Convert the image into grayscale. This will remove color information and replace it with a single intensity value for each pixel of the image. Gaussian blur is a pre-processing technique used to smoothen the edges of an image to reduce noise. Canny Edge Detection to find areas of the image that rapidly change over the intensity value. Canny function helps to identify edges of an objects present in the image and discard remaining portion.



Hough Transform:

Hough Transform is feature extracting technique in an image and find the class of shapes present with region of interest. Opencv has predefined function for Hough Transform and it helps to find the lane lines.

Building Lane Finding Pipeline:

To build a pipeline, need to determine which lines are in the left group and right group. In our images, the left lane markings all have a negative slope, meaning the lines travel upwards towards the horizon as we move from left to right along the lines. On the other hand, all of our right lane markings have a positive slope, traveling downwards towards the bottom of the image as we move along them from left to right.

```
def pipeline(image):

    height = image.shape[0]
    width = image.shape[1]

    region_of_interest_vertices = [
        (0, height),
        (width / 2, height / 2),
        (width, height),
    ]
    grayScale = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
                                # convert RGB into grayscale
```

Step1: Defining region of interest

Step2: convert RGB into grayScale

Step3: Canny edge detection function

```
cannyEdge = cv2.Canny(grayScale, 100, 200)#Canny edge detection function

roiImage = regionOfInterest(cannyEdge,
    np.array([regionOfInterestVertices],np.int3))

laneLines = cv2.HoughLinesP(cropped_image,rho=6,
    theta=np.pi / 60,threshold=160,lines=np.array([]),minLineLength=40,
    maxLineGap=25) #Hough Transform Lines function to extract the lanes line


left_x = []
left_y = []
right_x = []
right_y = []

for line in lines:
    for x1, y1, x2, y2 in line:
        slope = (y2 - y1) / (x2 - x1)
if slope < 0.5:
    continue
if slope <= 0:
    left_x.extend([x1, x2])
    left_y.extend([y1, y2])
else:
    right_x.extend([x1, x2])
    right_y.extend([y1, y2])

min_y = int(image.shape[0] * (3 / 5))
max_y = int(image.shape[0])

left = np.poly1d(np.polyfit(left_y, left_x, deg=1))

leftStart = int(left(max_y))
leftEnd = int(left(min_y))

right = np.poly1d(np.polyfit(right_y,right_x,deg=1))

rightStart = int(right(max_y))
rightEnd = int(right(min_y))

line_image = drawLines(image,
    [[
        [leftStart, max_y, leftEnd, min_y],
        [rightStart, max_y, rightEnd, min_y],
    ]],thickness=5)
```

Step4:

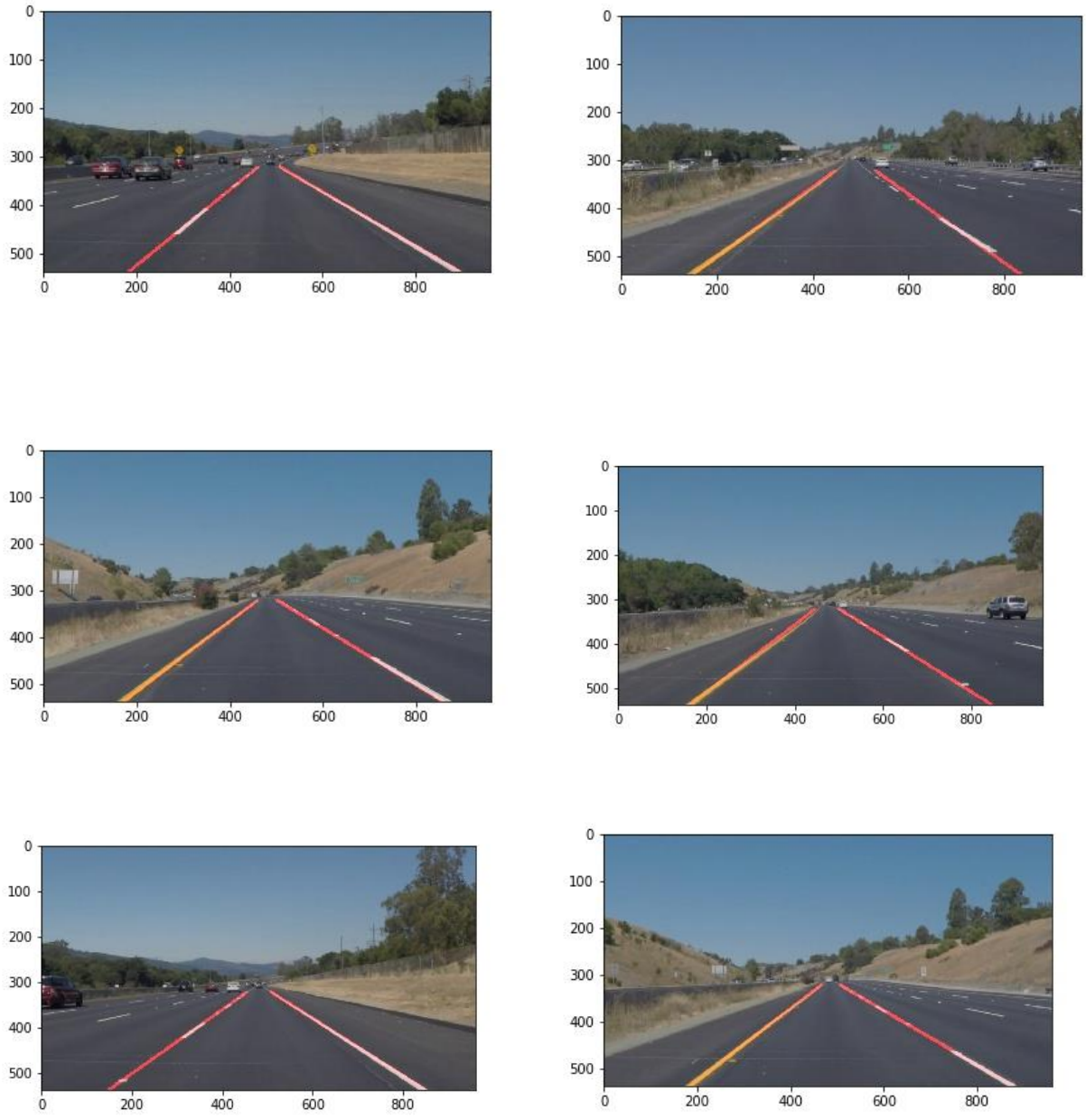
finding slopes in-order group separately for left lane and right lane

If negative slope is considered to be left lane slope.

If positive slope is considered to be right lane slope.

Step5: drawLines function will draw a line segment for left lane and right lane.

result images refer below



[Refer:] Result of solidWhiteRight.mp4 and SolidYellowLeft.mp4 present in (/test_video_output)

Shortcomings

- I have observed some problems with the current pipeline, challenge video the lane is covered by some shadow and my code originally failed to detect it.
- Straight lines do not work when there are curves on the road, even if reduce a parameters

Future Improvements

- Give some intro to hsv in order work with challenge video and explore some opencv library function.
- Make a video tutorial for working with Anaconda and Jupyter notebook.