Project -1: Finding Lanes Lines on the Road

1. Introduction:

The goal of this project is to create pipeline of algorithm in python that finds the lane lines on the road. The given data is the bunch of test images in the jpg format and two mp4 video streams recordings.

The code would ultimately draw a superimposed line with the lane lines, one on the top of the left side of the land and one in the right side of the lane.

2. Pipeline Procedure

There are many steps involved in the creating the pipeline and are outlines with a brief explanation:

1. Conversion of given images to gray scale images.

The image is first converted into grayscale image, to reject the unnecessary color information from the image. The idea here is to just find the region on the road where the lane lines are and that can be done by observing the abrupt change in the intensity of color that corresponds to the lane line.

There is no need for the image to be colored to observe the change in the intensity.

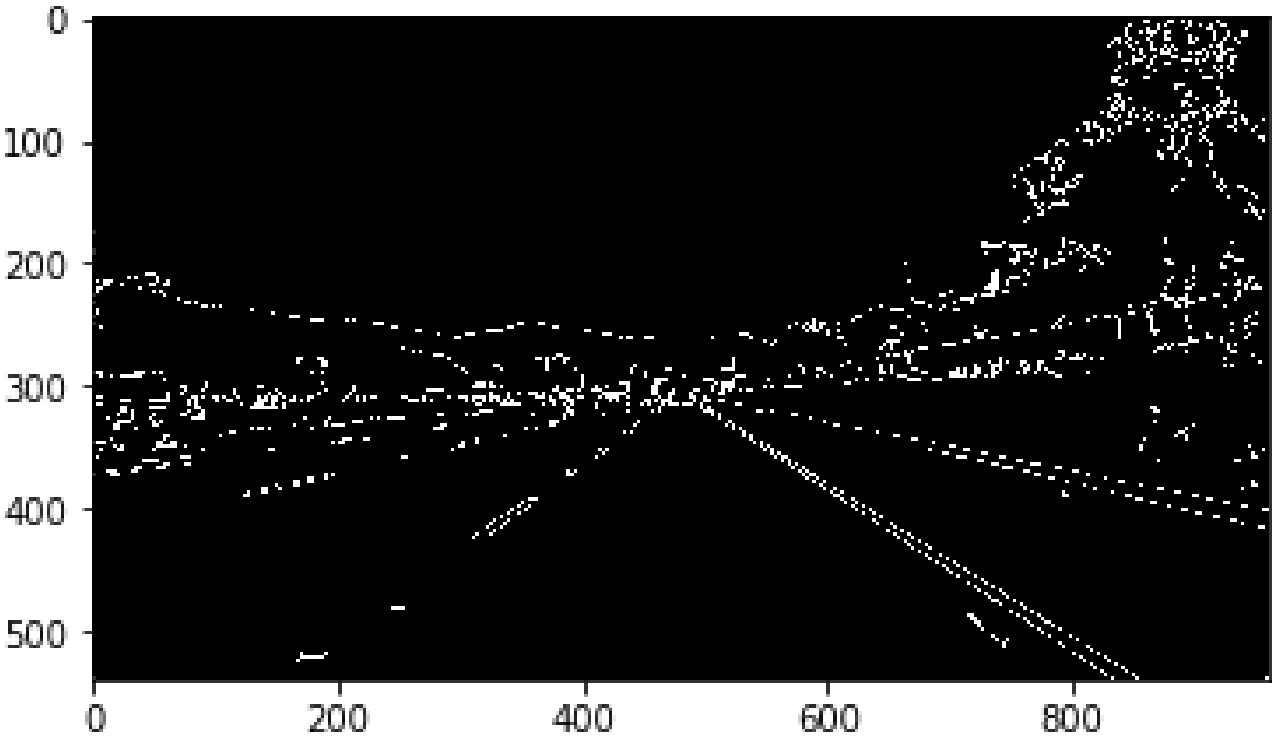
2. Applying Gaussian Blur:

After converting the image into grayscale, a Gaussian blur is applied on the image converted to grayscale with the kernel size of 5.

Gaussian blur removes the unnecessary information from the image. This function smoothes out the smaller edges to make the image much more clearer.

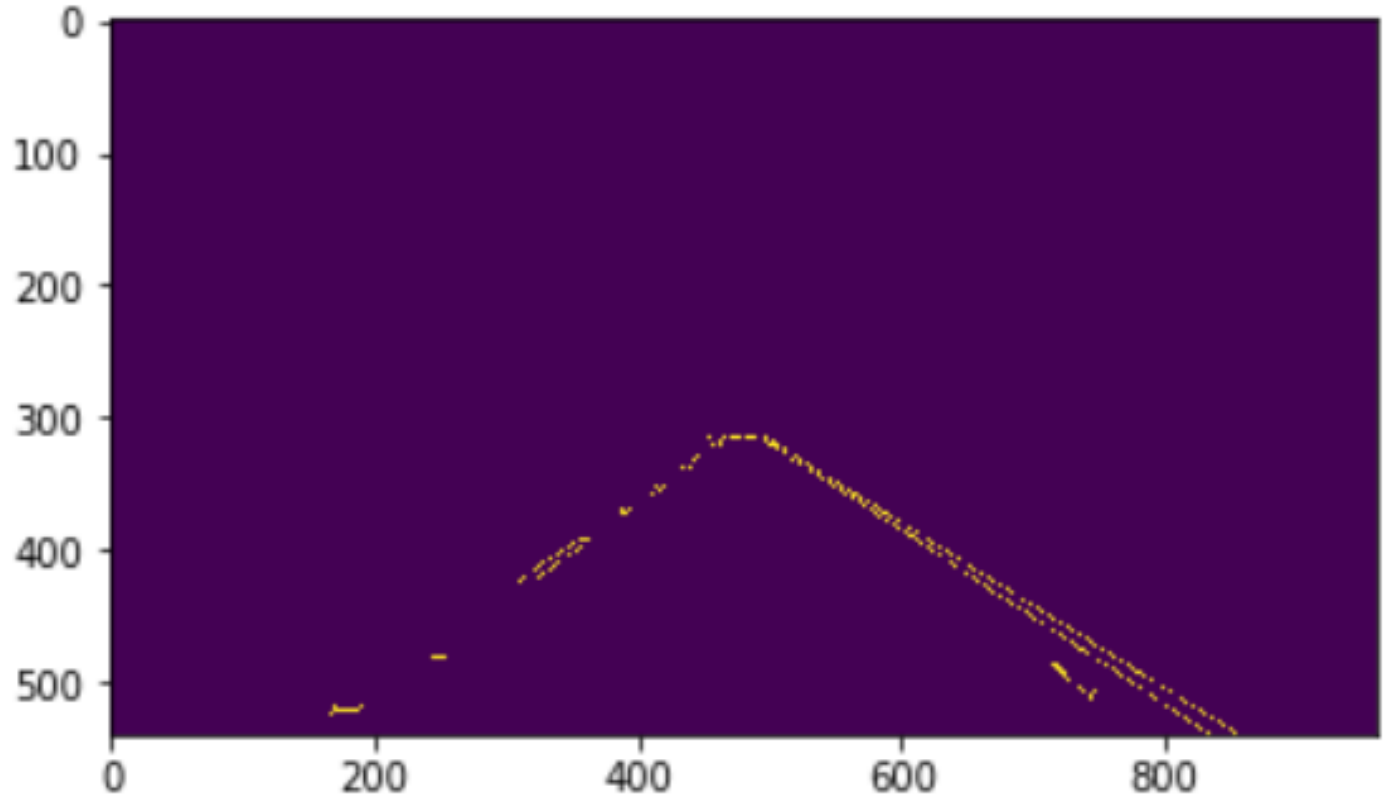
3. Canny Edge Function:

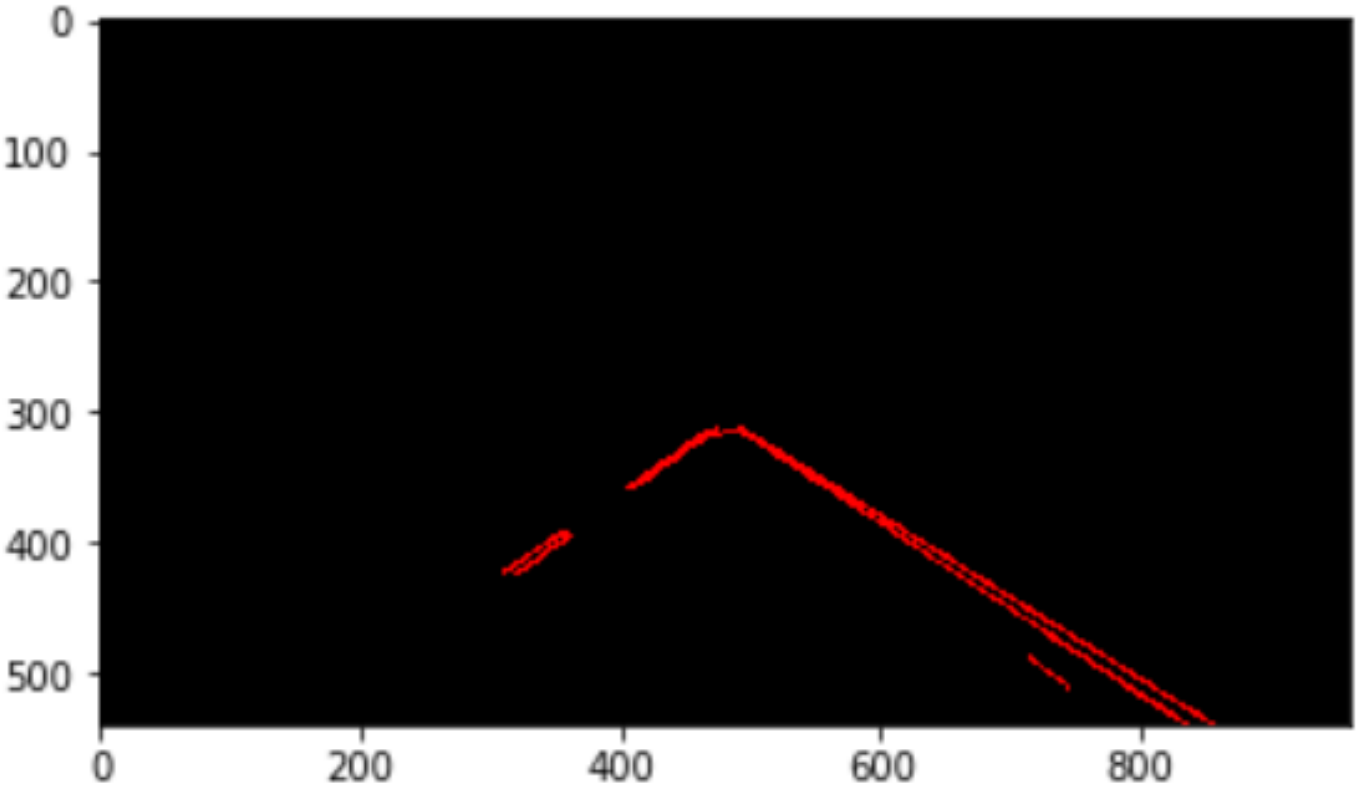
This function from computer vision library takes the gaussian blurred image as an input and converts into and image in binary format. The output from the canny edge detection function looks like as depicted below:



4. Cropping the image:

The next step in the pipeline is cropping the image, obtained from the canny edge detection function. It is done to remove the unnecessary edges detected (which are not lane line)





5. Hough Lines:

Applying Hough line function on the cropped image is the next step in the image processing pipeline. The input arguments and their values used are show below

```
# Hough Lines
rho = 1
theta = np.pi/180
hough_thres = 20 # minimum number of votes
min_line_len = 25 #minimum number of pixels making up a line
max_line_gap = 10 # maximum gap in pixels between connectable line segments
lines = hough_lines(cropped_img,rho,theta,hough_thres,min_line_len,max_line_gap)
```

6. Drawing superimposed Lane Lines:

The next step in the pipeline algorithm procedure is, drawing superimposed Lane Lines using the lines obtained from the Hough lines.

a. The first step is to group the line segments into the left group and right group. This is done by calculating the slope of the all the lines and the grouping them in left group if the slope is positive and in the right-group if the slope is negative.
b. Mean values of the all the left x and y and right x and y values are obtained by np.mean() function.
c. Slope and intercept of the mean left line and mean right line is calculated.
d. Intersection point of the mean Left is obtained with the lowest y in the image(which is ymax) and the selected top y co-ordinate.

e. Intersection point of the mean right line is obtained with the lowest y in the image and the selected top y co-ordinates.
f. The left X(x_left_bottom,x_left_top,x_right_bottom,x_right_top) and Y(y_bottom,y_top) obtained are then used to create left and y superimposed lines using cv2.line() function:
   cv2.line(img, (x_left_bottom, y_bottom), (x_left_top, y_top), color, thickness)
   cv2.line(img, (x_right_bottom, y_bottom), (x_right_top, y_top), color, thickness)
g. The function name in which is code is written is : def superImposedLines(img,lines,ratio_y=0.60,color = [255,0,0],thickness=16):
h. **This function is then called inside the Hough_lines function.**
i. Weighted_img is used to draw these lines over the selected image.

3. Potential shortcomings:

a. There are many assumptions while creating this logic, the algorithm relies on selecting the region of interest by trial and error method and hoping that the lane lines are captured inside the selected region of interest.
b. It highly relies on the selecting the Hough line input arguments- which are selected based on the trial and error again.
c. The algorithm does not consider the variation in lighting condition.
d. It does not consider the case if somehow the camera images obtained gets blurred in rainy condition or even under the condition where there not sufficient light.

4. Improvements:

a. Camera discrepancy consideration.
b. Better region selection where the lane lines are.
c. The algorithm should be tested on more video streams and variety of images.