Finding Lane Lines on the Road

The aim of the project is to develop a pipeline to find the lane markings on the road. The pipeline steps will be explained using the image **solidWhiteRight.jpg** given in the folder test images

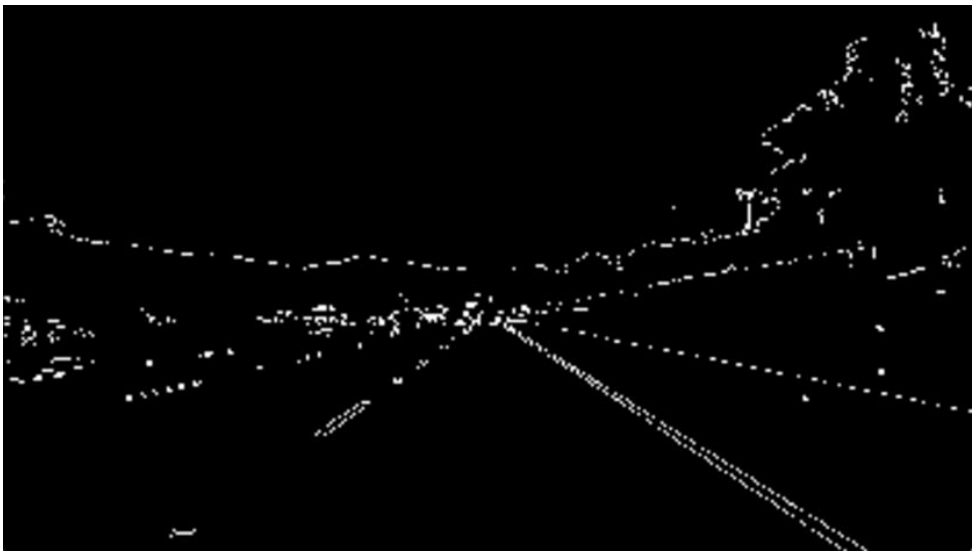
The original image

Step1: Apply Gray Scale Transform to the original image.
Code: gray = grayscale(image)

The resulting image is as shown below

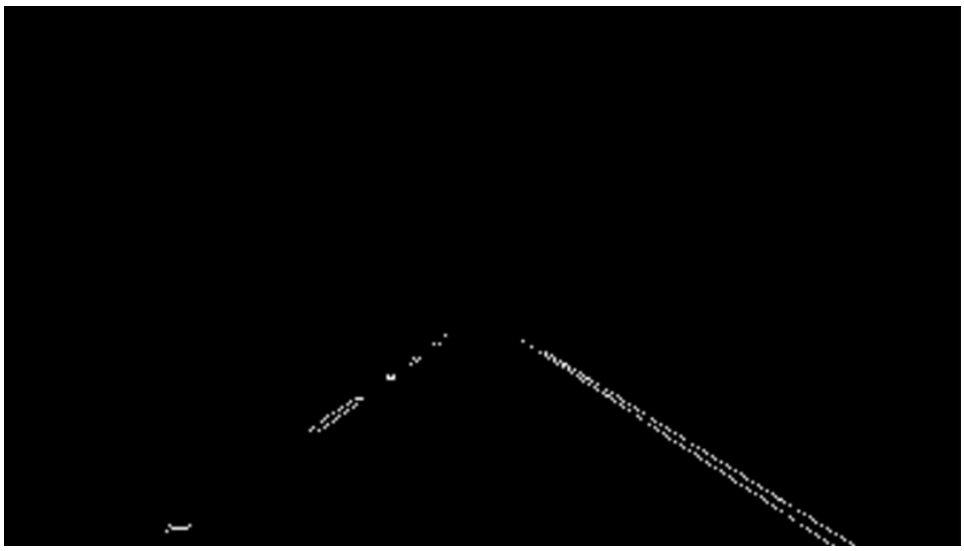
Step2: Detect Edges in the image
Apply Gaussian Blur with kernel size 3 before applying Canny Edge Detector to avoid noise.
Code: *gray_blur = gaussian_blur(gray, kernel_size)*

A Canny Edge detector with a low threshold of 50 and high threshold of 200 is applied. These values are determined by trial and error method. The best value is chosen.
Code: *gray_edges = canny(gray_blur, low_threshold, high_threshold)*



Step3: Masking the region of interest
A region of interest determined by the vertices of polygon and is applied to the previous output. This shall mask out the irrelevant edges by retaining only the lane markings within the masked region. The values of vertices of polygon is determined by trial and error method.

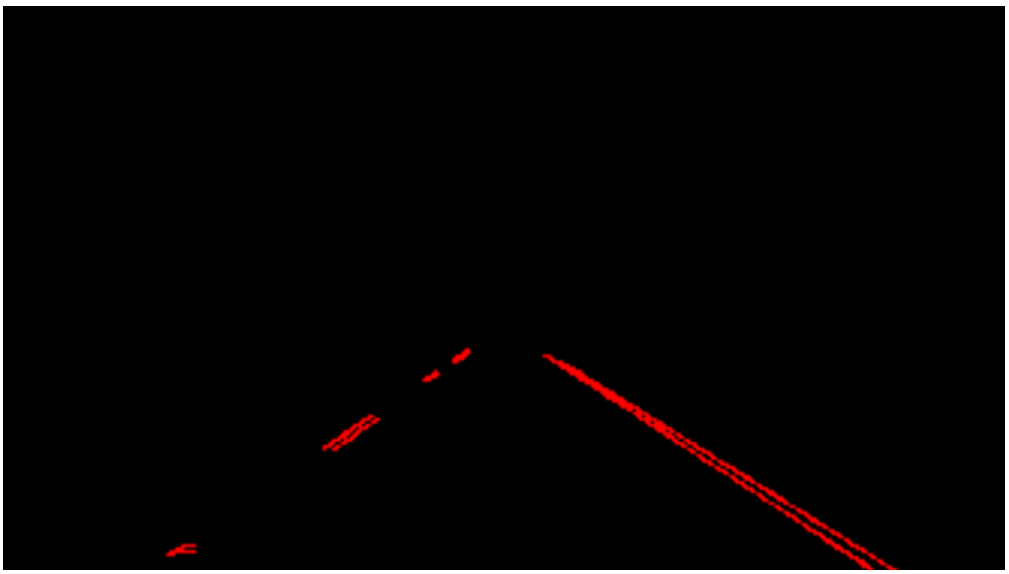
Code:
vertices = np.array([[(0,imshape[0]),(450, 328), (490, 320), (imshape[1],imshape[0])]], dtype=np.int32)
masked_image = region_of_interest(gray_edges, vertices)



Step 4: Marking the Lanes using Hough Transform

The detected edges are then marked with lines using the Hough Transform. The draw line function called within the Hough transform function shall mark the lanes with red color.

Code:
rho = 1
*theta = 1 * np.pi/180*
threshold = 15
min_line_len = 10
max_line_gap = 12
line_image = hough_lines(masked_image, rho, theta, threshold, min_line_len, max_line_gap, False)



Step5: Adding the original image with the previous output

The original image shall be added with the Hough transform output to obtain a color image with lanes marked in red as shown below

Code: color_line_image = weighted_img(line_image, image, α=0.8, β=1., λ=0.)

Step 6: Next step shall be to extrapolate the segmented lines to form a continuous line on the lanes

The **draw_line** function shall be modified to extrapolate the line segments. (Renamed **draw_lines_continuous**)

The principle behind extrapolation is to determine the average slope and intercept of each lane and determine the new line based on these values.

The code steps will be explained below

Initializing variables for left lane

left_slope_sum = 0	Sum of left slope
left_intercept_sum = 0	Sum of Intercept
left_average_slope = 0	Average Slope = Sum of left slope/ Number of lines
left_average_intercept = 0	Average Intercept = Sum of left intercept/Number of lines
left_count = 0	Number of lines on left lane

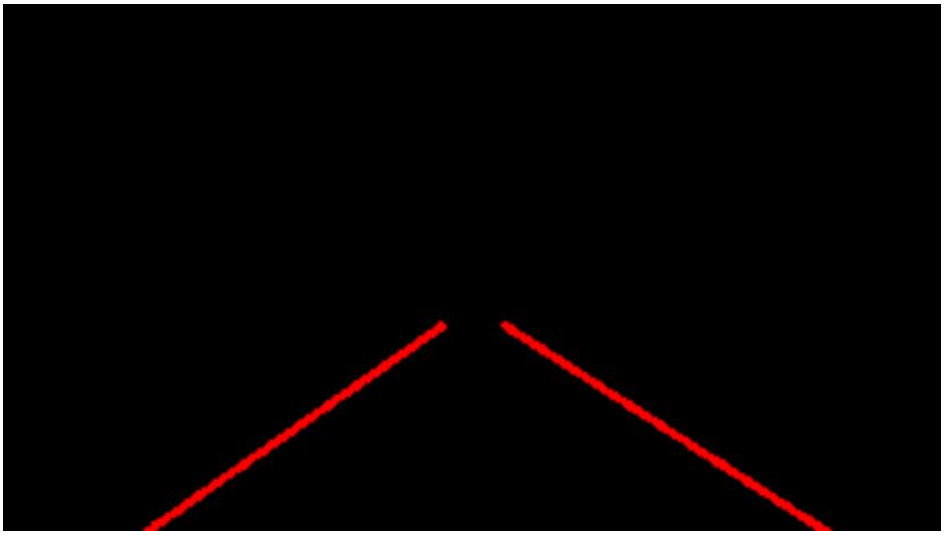
The similar variables are defined for right lane

right_slope_sum = 0
right_intercept_sum = 0
right_average_slope = 0
right_average_intercept = 0
right_count = 0

Next, we determine the slope and intercept of every line

slope = (y2-y1)/(x2-x1)
intercept = y1 - slope*x1

Based on slope value we determine if the line belongs to left lane or right lane. If the slope is negative then it is considered as left lane if it is positive then it is right lane. A threshold value greater than 0.5 for slope is considered, which shall reject irrelevant lines. These lines deviate the slope by a large margin resulting in undesired output.



The sum of slopes, sum of intercepts and number of lines are calculated after the above condition check
left_slope_sum = left_slope_sum + slope
left_intercept_sum = left_intercept_sum + intercept
left_count = left_count + 1

Next the average slope and average intercepts are calculated
left_average_slope = left_slope_sum/left_count
left_average_intercept = left_intercept_sum/left_count

Consider the new coordinates of y1 and y2.
Y1 shall be the end of the image frame
Y2 shall be the lowest value among the y2 co-ordinates of various lines provided as input. This value is chosen so that the extrapolated lines shall have variable length instead of fixed value. The other ways to consider y2 values are y2 = 0.5 * y1 or simply a hardcoded value y2 = 331.

Calculate the new values of x1 and x2 using formula X = (Y –Intercept) / Slope
x1 = int((y1 - left_average_intercept)/left_average_slope)
x2 = int((y2 - left_average_intercept)/left_average_slope)

The resulting output is as shown in the image below

Potential Short Coming with the pipeline

The current pipeline works well for the **solidWhiteRight.mp4** and **solidYellowLeft.mp4** videos but not for the **challenge.mp4.** Hence this pipeline is not a generic pipeline which works on all three test videos. A separate functions **process_image_challenge** and **draw_lines_challenge** has updates to the current pipeline which renders it to work on **challenge.mp4** video.

Improvements to Pipeline

Further improvement shall be to make the pipeline very generic to work at least on these three road conditions. Experimenting with HSV and HSL color model rather than RGB shall be an addon to make it work on all three test images.
Scenarios like a car crossing the lane within the region of interest is not considered. This might give a wrong lane marking and can be another improvement area.

The output of various test images are shown below

Update pipeline for Challenge Video

Here the image is converted from RGB to HSL as it provides better result in detecting lane markings under the tree and also on bright areas.

Convert RGB to HSL
hls_image = cv2.cvtColor(image, cv2.COLOR_RGB2HLS)

We determine the white mask and yellow mask from the image using **cv2.inRange** function and selecting appropriate threshold values as input

#Threshold values to detect white lane
lower_value = np.uint8([0, 205, 0])
upper_value = np.uint8([255, 255, 255])
white_mask = cv2.inRange(hls_image, lower_value, upper_value)

#Threshold values to detect yellow lane
lower_value = np.uint8([11, 0, 95])
upper_value = np.uint8([45, 255, 255])
yellow_mask = cv2.inRange(hls_image, lower_value, upper_value)

The two masks available are ORed using **cv2.bitwise_or** function and the output of this is ANDed with the original image using **cv2.bitwise_and** function to provide an image with yellow and white lanes detected. The accuracy of detecting lanes are higher by using HSL image instead of RGB.

image_mask = cv2.bitwise_or(white_mask, yellow_mask)
yellow_white_image = cv2.bitwise_and(image, image, mask = image_mask)

Once the masked image is available the usual pipeline procedures discussed above are followed. The vertices of the polygon which detects the region of interest are updated according to the camera angle and position.

Potential Short Coming with the pipeline

However, the challenge pipeline has to be improved further at 0:06 of the clip under the tree shade. The lines deviate from the left lane since the pipeline failed to detect the shaded yellow lines.

Improvements to Pipeline

To detect the shaded lanes under the trees the pipeline HSL color value range has to be updated. With the optimal value it will be possible to detect the light lanes under bright sun as well as lanes under shade.