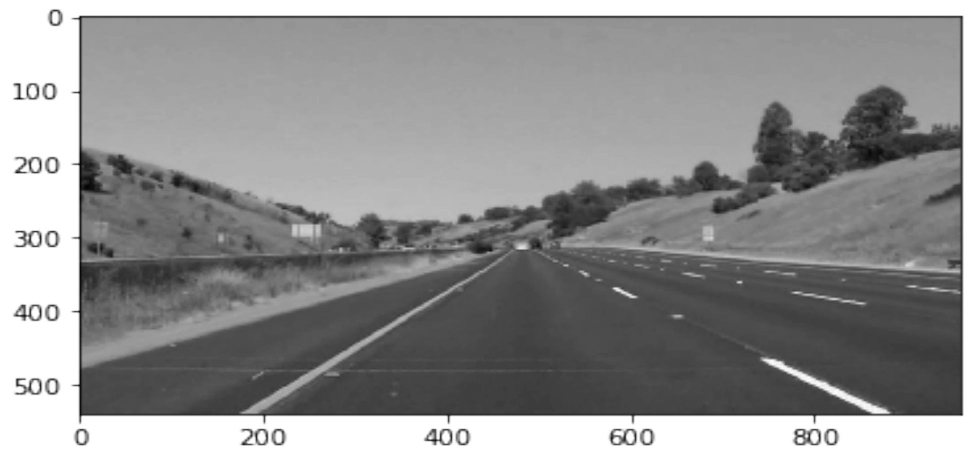
Reflection

1. Describe your pipeline. As part of the description, explain how you modified the draw_lines() function.

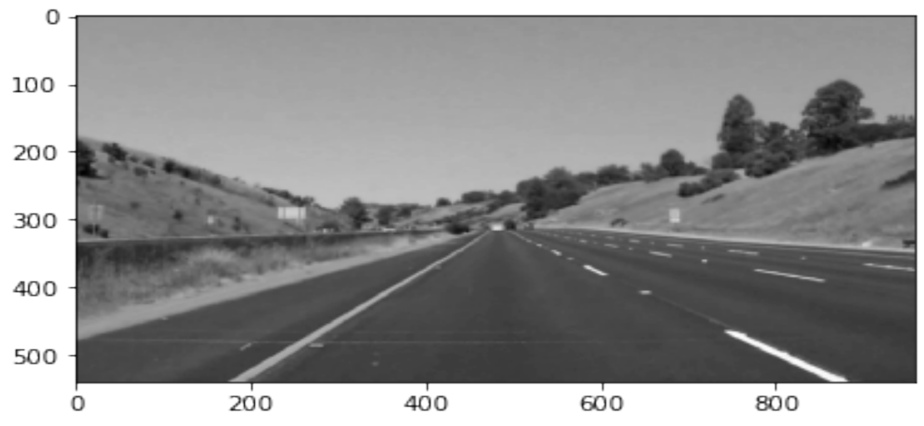
Answer: My pipeline consisted of 4 steps.

1) Gray scaling and Gaussian blurring: First I converted the images to gray scale and Gaussian smoothing:
   We need to convert the image to grayscale for further edge detection, we then apply Gaussian smoothing function to the image. The Gaussian smoothing will average out the anomalous gradient in the image before applying canny edge detection:

   Gray sample plot below:

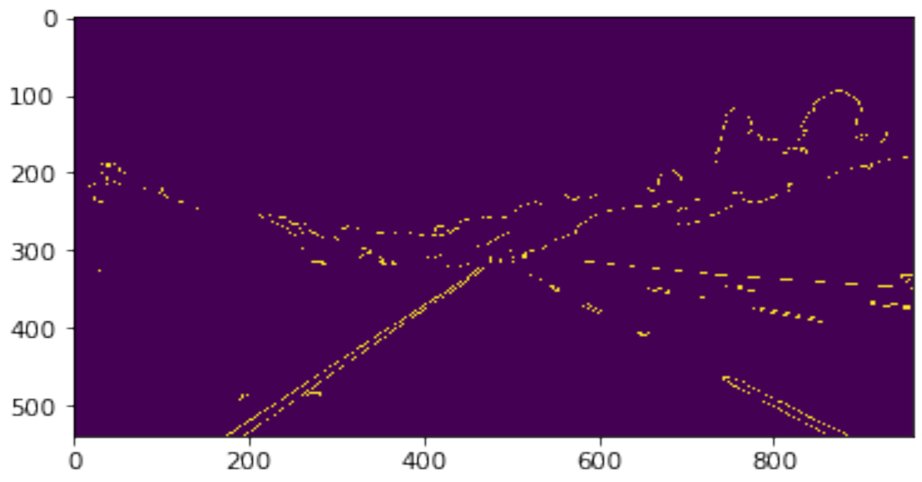


After performing Gaussian smoothing:



2) Canny edges: Now, I am performing canny edge detection with a lower threshold of 100 and an upper threshold of 200. The output edges are a binary image with white pixels tracing out the detected edges and black everywhere else.

Canny edge detection below:



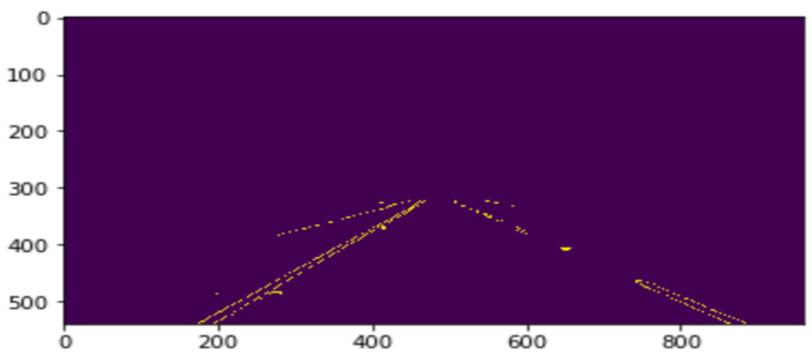
3) Canny Edges: At the 3rd step I am keeping the polygon on interest for the edges detected. Please refer below the code snippet for finding the vertices provided and to call the function provided to build the region of interest:

```
#calclutaing the vertices for each image

h = imshape[0]
w = imshape[1]
vertices = np.array([[(w/6,h),(w/3, 3*h/5), (3*w/5, 3*h/5), (w,h)]], dtype=np.int32)

# mask the canny output with region of interest
masked_image = region_of_interest(edgesImage, vertices)
```

Please not the height, width of a picture and how I am guessing the four co-ordinates of the edges of the polygon. Please refer below the masked image with region of interest:



4) Hough Transform:

In Hough space, we can represent "x vs. y" line as a point in "m vs. b" instead. The Hough Transform is just the conversion from image space to Hough space. So, the characterization of a line in image space will be a single point at the position (m, b) in Hough space. After detecting the edges we want to identify the lines which form the two lane lines and as explained in the lesson we are using Hough transform for this. We are operating on the image edges (the output from Canny) and the output from HoughLinesP will be lines, which will simply be an array containing the endpoints (x1, y1, x2, y2) of all line segments detected by the transform operation. The other parameters define just what kind of line segments we're looking for.

Here are the parameter values I chose:

```
rho = 2 # distance resolution in pixels of the Hough grid
theta = np.pi/180 # angular resolution in radians of the Hough grid
threshold = 45    # 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
```

***Please note I chose the values of these parameters randomly every time looking at the results and changing the values in turn to get the best the lanes detected in the image

Extrapolating the detected lines: What we need to do here is, once the lines are detected we need to extrapolate both the lines such that it cover the image mostly and we detect the lanes completely. Please refer the steps below:

a) First, I grouped the lines in two groups left lane and right lane, the slope of all the left lines (part of the left lane) would be positive and the slope of all the right lines (part of the right lane) would be negative, so pretty straightforward to group them
b) The maximum Y co-ordinate for all points would the height of the image but need to calculate the minimum y co-ordinate both for left and right lane but iterating through line and keeping the minimum y.
c) Second, to draw extrapolated lines for left and right lane, I need the co-ordinates of the top and bottom points (X and Y both, I know Y, as explained in b)) for both left and right lanes. For that, I need calculate the average slope, average y coordinate values, average x co-ordinate values for all left and right lines. Using these average slopes, y and x values, I calculated the average intercept for both left and right lines. Now, I have average Slope, average intercept and Y values for both top and bottom tips of the left and right lanes, now it’s pretty straightforward to calculate the X-ordinates for both top and both tips for both left and right lanes.
d) Now, I have the X and Y co-ordinates for the top and bottom tips for left and right. I can now draw the extrapolated line.

Please refer the code for extrapolating:

```
def draw_lines_extrapolate(img, lines, color=[255, 0, 0], thickness=10):
    imshape = img.shape

    #Defining the max and min Y co-ordinates to which we need to extrapolate left and righ
    ymin = img.shape[0] #initilizing with a very large garbage value, inthis case kept the
    ymax = img.shape[0]

    #defining lists for all left lane values for slope, y coordinate & x coordinate, this
    #the mean values for gradient, y coordinate & x coordinate for all left lane lines
    left_slop_list = []
    left_y_list = []
    left_x_list = []

    #defining lists for all right lane values for slope, y coordinate & x coordinate, this
    #the mean values for gradient, y coordinate & x coordinate for all right lane lines
    right_slop_list = []
    right_y_list = []
    right_x_list = []

    for line in lines:
        for x1,y1,x2,y2 in line:
            grad, intercept = np.polyfit((x1,x2), (y1,y2), 1)
            ymin = min(min(y1, y2), ymin)

            if (grad > 0):
            #if grad is greater than zero then this line must belong to left lane
                left_slop_list += [grad]
                left_y_list += [y1, y2]
                left_x_list += [x1, x2]
            else:
            #if grad is less than zero then this line must belong to right lane
                right_slop_list += [grad]
                right_y_list += [y1, y2]
                right_x_list += [x1, x2]

    left_slop_mean = np.mean(left_slop_list)
    left_y_mean = np.mean(left_y_list)
    left_x_mean = np.mean(left_x_list)
    left_intercept_mean = left_y_mean - (left_slop_mean * left_x_mean)

    right_slop_mean = np.mean(right_slop_list)
    right_y_mean = np.mean(right_y_list)
    right_x_mean = np.mean(right_x_list)
    right_intercept_mean = right_y_mean - (right_slop_mean * right_x_mean)


    left_x_top = int((ymin - left_intercept_mean) / left_slop_mean)
    left_x_bottom = int((ymax - left_intercept_mean) / left_slop_mean)
    right_x_top = int((ymin - right_intercept_mean) / right_slop_mean)
    right_x_bottom = int((ymax - right_intercept_mean) / right_slop_mean)


    cv2.line(img, (left_x_top, ymin), (left_x_bottom, ymax), color, thickness)
    cv2.line(img, (right_x_top, ymin), (right_x_bottom, ymax), color, thickness)
```

Source:
https://classroom.udacity.com/nanodegrees/nd013/parts/fbf77062-5703-404e-b60c-95b78b2f3f9e/modules/83ec35ee-1e02-48a5-bdb7-d244bd47c2dc/lessons/8c82408b-a217-4d09-b81d-1bda4c6380ef/concepts/a21e50a1-bf49-4ec4-ab1d-5b25d0aa4428

https://classroom.udacity.com/nanodegrees/nd013/parts/fbf77062-5703-404e-b60c-95b78b2f3f9e/modules/83ec35ee-1e02-48a5-bdb7-d244bd47c2dc/lessons/8c82408b-a217-4d09-b81d-1bda4c6380ef/concepts/90e3a984-eb68-48d3-8724-e58303a8d2cc

https://classroom.udacity.com/nanodegrees/nd013/parts/fbf77062-5703-404e-b60c-95b78b2f3f9e/modules/83ec35ee-1e02-48a5-bdb7-d244bd47c2dc/lessons/8c82408b-a217-4d09-b81d-1bda4c6380ef/concepts/22086a6f-15ce-4121-bacd-1969f2b61fb9

2. Shortcoming and Improvements:

I am able to extrapolate in both the videos: solidWhiteRight.mp4 & solidYellowLeft.mp4

** But I am not convinced with my result with solidYellowLeft.mp4. If you could observe my result video “yellow.mp4”, although I am able to extrapolate completely, but at certain sub second instances I am losing the track of the lanes. I feel there is something to do in detecting the Yellow solid lines as my “white.mp4” is behaving perfectly as expected.

I would feel there is a way to improve my implementation for the second video-solidYellowLeft.mp4. I am not so sure how to do this but precisely in detecting solid “Yellow” lines my implementation could be improved. I could not build a pretty deep of gray scaling as I would feel it’s something to do with gray scaling. Overall, I am able to extrapolate the lines to detect left and right lanes but I was hoping I could do a better job in the yellow video as well the bonus challenges. I am looking forward to advices and help to improve my implementation for these two case.