Project 1: Finding Lane Lines on the Road

The goal of this project is to write a software pipeline using image analysis techniques, that will identify position of the left and right lanes lines on the road, in the images and video stream.

Reflection

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

My pipeline consisted of 7 steps as listed below

1. Grayscale –
    a. Grayscaling takes the original color image in three color channels (r,b,g) as input and returns an image with only one color channel. This is the first step towards building a code to detect boundaries of an object in an image or series of image. Rapid changes in the brightness (pixels) in a grayscale image is where we find strongest gradients (edges)
    b. Output image from function 'grayscale' is provided as input to function 'gaussian_blur()'
2. Gaussian Blur -
    a. Applying Gaussian blur suppresses noise and spurring gradients by averaging adjacent pixel values. We apply additional layer of adjustable Gaussian blurring before applying Canny edge detection to smooth the image out.
    b. Two inputs to the function 'gaussain_blur()' are grayscale image 'gray' and kernel size =3. Higher kernel size applies smoothing over a larger area.
    c. Output of image from function 'gaussian_blur()' is used to apply Canny edge detection
3. Canny Edge Detection –
    a. Canny function from OpenCV detects strong gradients in the smoothened image 'gaussian_img' by highlighting pixels above specified 'high_threshold' value and rejecting pixels below 'low_threshold' values. Once strong edges are detected, pixels between high and low threshold interval are highlighted if they are connected to pixels above high threshold.
    b. The output image from Canny edge detection – 'canny_img' is a binary image with strong edges highlighted in white and rest of the area blacked out.
    c. I have set the low to high threshold ratio to 1:3 and in the middle region (60 to 160) of the 0 to 255 scale to detect sufficiently dark images.
    d. High threshold – 160
    e. Low threshold – 60

4. Region of interest –
    a. Region of interest is applied to narrow down the region so as to try and eliminate almost everything else on the image except lane lines
    b. The function 'region_of_interest()' takes image processed with Canny edge detection along with [x,y] coordinates of four vertices that would form polygon of region of interest.
    c. I have defined the four vertices as a fraction of height and width of the image focusing on the lower half where lanes generally lie in all the test images and videos.
    d. The output image contains only the image defined by polygon formed from four vertices and rest of the image is set to black
5. Hough lines –
    a. Hough lines function detects lines on the masked image using openCV function 'HoughLinesP', after applying Hough Transform based on following input parameters
        i. Masked image containing only region of interest
        ii. Distance and angular resolution of grid in Hough space – kept minimum at 1 and pi/180 (1 degree) respectively for a finer grid
        iii. Threshold number of intersections in a grid cell needed for it to be defined as line (=35) set high enough such that it only detects true lane lines and filters the noise out
        iv. Minimum length of line allowed (=5)
        v. Maximum gaps in pixels allowed to be part of same line(=2)
    b. Output of this function is array of lines with coordinates of their endpoints
    c. This is used in the function 'draw_lines()' to defined one single line for left and right lane lines
6. Draw lines –
    a. In order to detect the complete left and right lane lines, and draw one single line on the left and right lanes, I modified the draw_lines() function as follows
        i. Iterate through each line endpoints (x1,y1,x2,y2) from 'lines' array and calculate their slope and center
            - Slope = (y1-y1)/(x2-x1)
            - Center = [(x2+x1)/2, (y2+y)/2]
        ii. If the slope is greater than a small positive number (> 0.1), the slope and corresponding center is saved in a separate array of left slope and left center. Same steps are followed for right lane if the slope is smaller than a small negative value (<-0.1)
        iii. Averaged all left/right slope and center values to get single slope and center for left and right lane
        iv. Two calculate endpoints of the left and right lane lines, I assumed two y coordinates equal to image height and 60% of image height. Then, I calculated corresponding x coordinates using slope and centers for left and right lanes
            - Formula used - (y-y')=M(x-x'), where [x',y'] are coordinates of calculated average center

v. Converted all coordinate values into integers to provide as input to line function in openCV
vi. And used cv2.line function two draw lines on the original image with color red and thickness of 5

7. Overlay images with Hough lines –
    a. As final step, I overlaid the left and right lane lines on top of the original image using weighted image function. The function takes processed black image after Hough Transform with only lane lines drawn on it and overlays it on top of the original image
    b. Transparency of the processed image is set to a higher value than original image

Step by step outputs for test image 'solidWhiteCurve.jpg' is as follows:

Figure 1: test image

Figure 2: Grayscale

Figure 3: Gaussian Blur

Figure 4: Canny Edge Detection

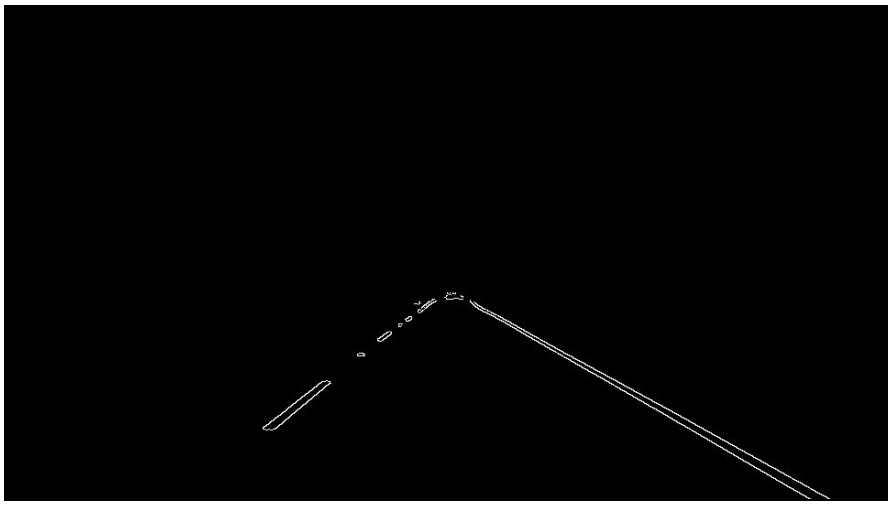

Figure 5: Region of Interest

Figure 6: Final image after Hough Transform

Figure 7: Final image after Hough transform with r,b,g channels reversed

Figure 8: Final image with draw lines function

2. Identify potential shortcomings with your current pipeline

Potential shortcomings with my current as pipeline identified after it was run on challenge video

- Region of interest is not well defined. So, when the challenge video is run with current region of interest, it captures edges from left divider, which skews calculation of average slope and center for left lane
- Unwanted edges are detected during tree shadow portion and change in road shade portion and due to portion of car hood visible at the bottom of the image
- The pipeline is not designed for curved lanes and calculates instantaneous vertical lane lines during the curved portion of the road
- Wobbly lane lines are detected on the challenge video

3. Suggest possible improvements to your pipeline

Possible improvements that can be implemented to my current pipeline to avoid above mentioned shortcomings are as follows –

- Region of interest can be made smaller such that it doesn't capture following disturbances
    a. Left divider
    b. Black car on the right
    c. Visible hood of the car at the bottom of the image
- To remove spurring gradient detection during tree shadow and road shade change portion of the video, Gaussian blur needs to be intensified with higher kernel size
- Low and high threshold of Canny Edge detection can be modified through iterations to avoid unwanted edges. Increasing value of minimum numbers of votes required to detect a point in Hough Transform will filter out noise.
- Possible solutions to remove errors due to curved lane lines
    a. Reduce height of the region of interest so as to not include lane curvature in the average slope and center calculation
    b. Instead of taking average of left and right slopes, we can take a median value or only consider average of slopes of the lane lines detected in the small section of the region of interest closer to the car where lane lines are straight