Project 1 Write-up: Finding Lane Lines on the Road

In order to solve this problem, my pipeline so that it contains the following steps:

1) Color Selection
2) Region of interest selection
3) Grayscaling
4) Gaussian Smoothing
5) Canny Edge Detection
6) Hough Transform Line Detection

The goal was to get an accurate lane line tracing on my video on both sides.

Before I could apply the above steps to solve my problem, I imported all the important libraries that are crucial for solving my problem. This included Pyplot, mpimg (to read the images), Numpy and cv2.

Next I setup helper functions, which would utilize the cv2 image-processing library to process the data in different ways.

The first helper function called greyscale converts the image into one color channel.

The second helper function called canny applies the Canny Transform to the image. This function uses the Canny Algorithm to find edges in the input image and marks them in the output map.

The third helper function called gaussian_blur applies a gaussian noise kernel, which blurs an image using a Gaussian Filter.

The fourth helper function called region_of_interest applies an image mask which helps to find out the region of the image containing the line segments as the function only keeps the region of the image defined by the polygon formed from the "vertices". The rest of the image is set to black.

The fifth helper function draw_lines is the function you might want to use as a starting point once you want to average/extrapolate the line segments you detect to map out the full extent of the line.

The sixth helper function hough_lines is a transform used to detect straight lines. It does this by finding the number of intersections between curves. The more curves means that the line represented by that intersection have more points. The Hough Line Transform keeps track of the intersection between curves of every point in the image.

The seventh helper function weighted_img takes the output of the hough_lines transform function, which is just an image with lines drawn on it and combines it with the initial image before preprocessing and balances the weights so that lane markers are visibly seen overlaid on the original image.

Next I tested my pipeline to run on the images under the "test_images" folder. For this I had to make some additional functions.

I defined a new function called draw_straight_lines which takes values for x1_right, x1_left, x2_right, x2_left, y1_right, y1_left, y2_right and y2_left. So what this does is basically, stores in values for different vertices and draws two straight processed lines with color and thickness. Lines are drawn on the image inplace (mutates the image, y_min is the point at which the lines should terminate.

The next new function I defined was called hough_straight_lines which takes the output from the draw_straight_lines function and returns the image with straight hough lines drawn.

Next I define a function called process_img which uses helper functions with their associated parameters to modify the image. First we grayscale it, then blur it and then use canny edge detection to find the edges. After that we use region_of_interest function to map the edges to make a mask and then use hough_straight_lines function to create a black image with red lines outlining the lanes. We use the weighted_img function to overlay the hough transform image over the original image.

The process_dir function takes all the images in the directory and applies the above techniques to all of them.

For videos, we define a set of lists for the points x1, x2, y1, y2. Then we calculate moving averages with qty set to 5. The draw_straight_lines function is modified a bit to adjust for the moving averages and recalculates the moving average if difference is less than 20 % from average. Then it updates the moving averages and at the end, draws lines on the image the same way.

The rest is the same as the rest of the code used to process still images.

At the end the draw_line function is changed a bit to define a line to run the full length of the visible lane based on the line segments that were identified with the transform by using moving averages. After using gradients to move and classify the lines as belonging on the left or right lane marker, I averaged the position of each of the lines and extrapolated to the top and bottom of the lane.