Finding Lane Lines on the Road

Reflection

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

The first part of the project is to detect line segments of the *test images* in **test_images** directory and draw the lines on to the images and saving the output in the **test_images_output** directory. To do that following helper functions are used:

- `grayscale()` : To discard color information
- `gaussian_blur()` : To remove the noise from the image
- `canny()` : To use canny algorithm to detect the edges
- `region_of_interest()` : Since we know the stable position of the camera, we can assume the lane lines to be observed should be in a specific portion of the image. This function is used to create a polygonial shape and filter the required region of lane lines. This helper function gets the polygon corners from the function `assign_polygon_corners()`.
- `hough_lines()` : This helper function is using **HoughLinesP** function in **OpenCV**. It helps us to decide which points (coming from canny edge detection) should create lines. After deciding the line segments, `draw_lines()` helper function is used to

draw the lines on the image.

- `weighted_img()` : To combine the lines with the original image.

After drawing the lines on the test images, the same functions are used as a pipeline to draw the lines on the solid white lane video (**solidWhiteRight.mp4**). The output can be found **test_videos_output** directory with the same name.

The second part of the project is to connect/average/extrapolate line segments and draw them on the frames of **solidYellowLeft.mp4**. To do this, the previous pipeline is used as a starter to find/detect line segments. After finding the line segments; left and right lanes are separated with `calculate_slope()` and `determine_lane_lines()` functions by their slopes as the following;

- Positive slope indicates right lane line
- Negative slope indicates left lane line

However, there is still ***noisy*** slope data in the seperated lanes. Based on the observations, these unaccepted ***noisy*** slopes are filtered with some slope thresholds. After getting optimized lane lines, `numpy.polyfit` function is used in the function `polynomial_fit()` to obtain polynomial fits (m = slope & b = y-intercept).By using slopes and y-intercepts; ***x1, y1, x2, y2*** main line points are calculated in `find_line_endpoints()` function. `draw_lines()` function is used as base and a new `draw_extrapolated_lines()` function is created. This function is using line stacks of right and left lanes with the new **thickness** value of **10**. The obtained extrapolated lines are

superimposed on the video frame by using `weighted_img()` helper function.

2. Identify potential shortcomings with your current pipeline

The current pipeline is very basic and primitive. It is very vulnerable to weather and lighting conditions, even to the lane line color. Another shortcoming is the lines created by the algorithm are no more effective when it comes to edges of the road curves. Another shortcoming could be the position of the camera. With each different camera orientation, the region of interest should be modified.

3. Suggest possible improvements to your pipeline

Since we are required to use **Canny Edge** detection algorithm, the **RGB** images are converted to **grayscale**. By doing that, valuable color information is lost. A possible improvement would be to use **S** (**saturation**) channel by using the HLS color space. Therefore, more robust detection could be achieved.