write_up

February 3, 2020

1 Finding Lane Lines on the Road

1.1 Reflection

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

My pipeline is consisted of 5 steps. First, I converted the images to grayscale by calling the helper function ***grayscale()***, which applies grayscale transform and return image with only one color channel. Then I applied Gaussian blur to the grayscale image. The benefit of the applying gaussian blurring is that it reduces image details and noise. For this I called ***gaussian_blur()*** helper function with *kernel_size=5*.

Next important step is applying, Canny tranform to the image. I called helper function ***canny()*** with parameters: *low_threshold=50, high_threshold=150*. This applies Canny edge detection algorithm to the image. After that, The task is to define region of interest in the image because there is only certain parts of the image is only important for this current problem to solve. To achieve this task again I used the already provided helper function ***region_of_interest()***. This function black out other part of the image except the are defined by the vertices of polygon.

The last and important step is to draw Hough lines on the image that we got after Canny tranformation. Here I called helper function ***hough_lines()*** with below parameters: *rho=1, theta=np.pi/180, threshold=10, min_line_len=5, max_line_gap=5.*

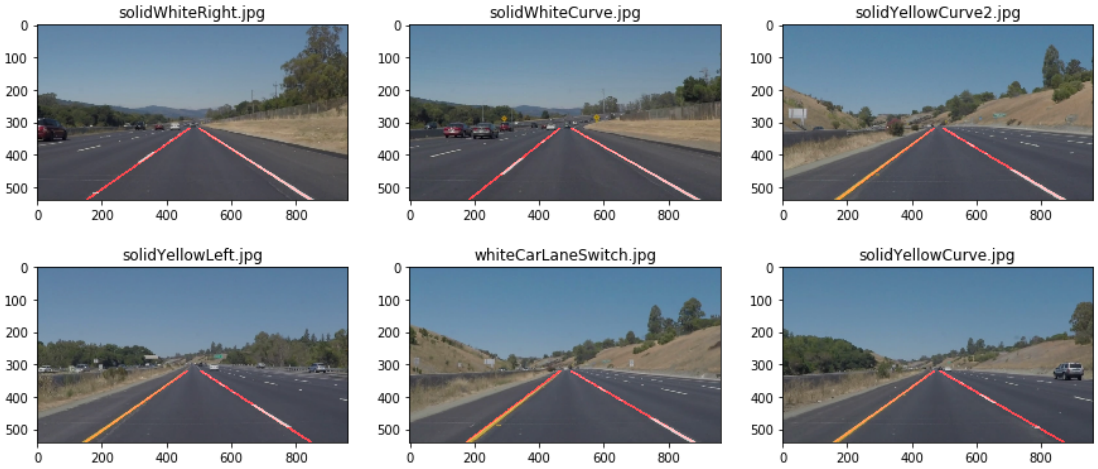
Further, In order to draw a single line on the left and right lanes, I modified the ***draw_lines()*** function by some extent. Firtly, I calculated slope(m) from the given points this is important to segregate the points between left and right lines. Also, there are some outliers points which needs to be removed based on there slope values. After segregating the points I calculated averaged points and used ***polyfit()*** function to get the slope and intercept of the line. I have done the above process for both left and right lines separately. Here It is easy to find out the y-cordinates of the two ends point of the line, As line starts from the bottom of the image and ends in the middle. Hence, *y_max=540 and y_min=320*. Now, as we know the intercept and the slope of the line then I can find out the x-cordinates easily using below formula:

$$x = \frac{(y - intercept)}{slope}$$

.

Once the co-ordinates of two end points are available the the line can be drawn easily using *OpenCV* function ***Line()***.

After the pipeline is ready I ran it on the 6 test images and got the following results:



1.1.2 Identify potential shortcomings with your current pipeline

One potential shortcoming would be what would happen when when we receive images of different size then this pipeline will fail as in the end points I have taken in hard-coded manner.

Another shortcoming could be that this set will not work in case of curve roads as I can see in the *Challenge* section it is getting failed.

1.1.3 Suggest possible improvements to your pipeline

A possible improvement would be to choose the end points automatically so that the pipeline shoould work with all size of the images. Also, eventhough we receive image of different size we can resize and then feed into the pipeline.

Another potential improvement could be to make this pipeline work in all scenario as we can bound ourselves to only some fixed cases. Real world provided more kind of challenges to us, so pipeline needs to be more robust.