Udacity Self-Driving Car Nanodegree Project 1

—Finding Lane Lines on the Road

-Guisheng Wang

This was very exciting project for me to see the code I wrote could find the lane line on the pictures and even the video. I believe that I had made the right decision to study on Udacity and looking forward to the following classes and projects. I appreciate the hard work from Udacity and also the support from my mentor!

Describe your pipeline.

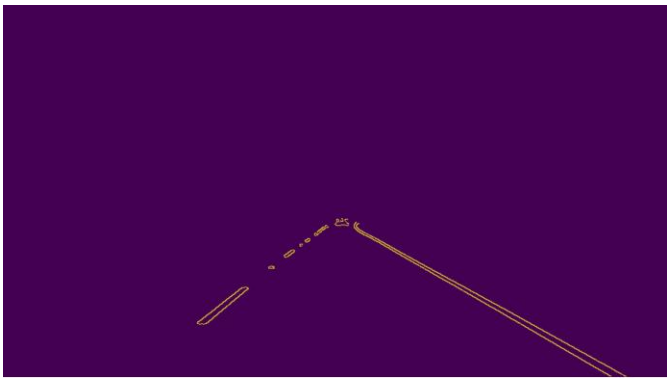
My pipeline consisted of following steps.

- Convert the images to grayscale.

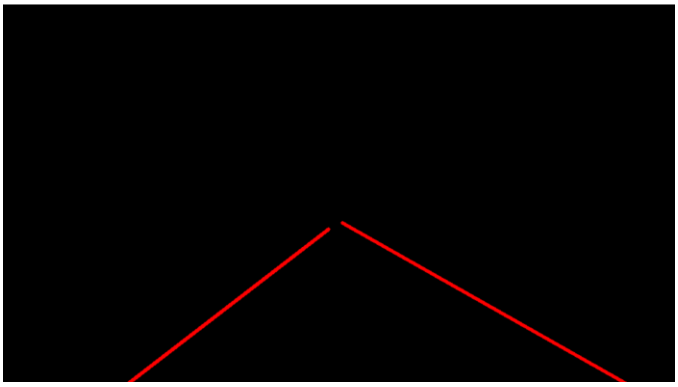
- Apply Gaussian Blur to smoothen edges

- Apply Canny Edge Detection

- Apply Region Of Interest and discard all other lines outside the region



- Perform a Hough Transform to find lanes



- In order to draw a single line on the left and right lanes, I modified the draw_lines() function and named it as draw_whole_lines() which includes following steps
    - ✧ Separate left and right lanes by slope of each
        - left lane: negative slope
        - right lane: positive slope
    - ✧ Calculate the slope and intercept of left and right line by using linear regression on end points of the result in above step.

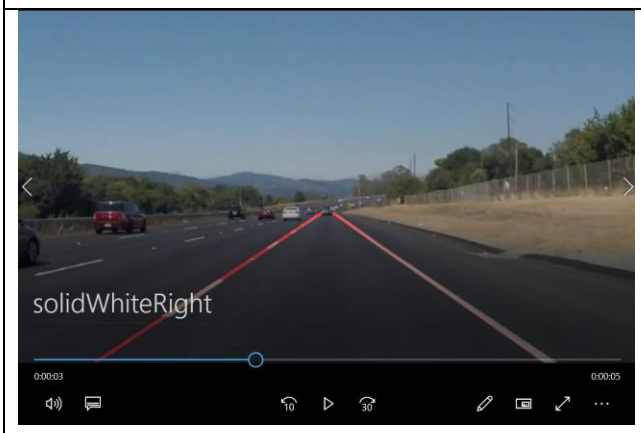
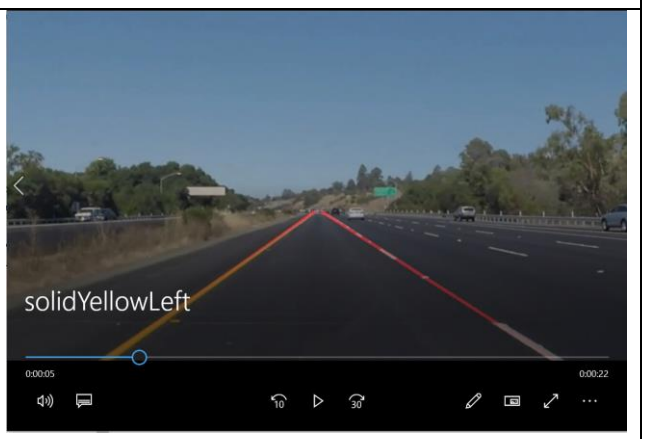
- Drawing the left and right line with the region of interest.

- Combine new image with lanes to the original image.

Videos pipeline

Compared to pipeline for individual images, there is an algorithm for video in draw_ draw_whole_lines() as below,

- Separate left and right lanes by slope of each
  - left lane: negative slope
  - right lane: positive slope
- Calculate the slope and intercept of left and right line by using linear regression on end points of the result in above step.
- Compare the slope with the slope from previous frame,
  - If there is more than 30% of difference, the previous slope and intercept will be used.
  - If there is more than 30% of difference but less than 15% of different, the previous slope and intercept and current slope and intercept will be average and used for this image
  - If there is no lane was found, the slope and intercept of the previous image/frame will be used.

solidWhiteRight	solidYellowLeft
	

Challenge Video

Instead of using grayscale, the HSV was being

- Convert the images to HSV.
- Apply yellow mask and white mask separately and combine the result into one image.
- Apply Gaussian Blur to smoothen edges

  …….The following steps are the same as in the pipeline above.

Shortcoming

- To define the region of interest is a tricky part, at beginning, I thought I need to define different area for each picture but luckily first region also works for other pictures. While in the real time, it might not work due to the movement of the car and also the curve of the road, etc.. This is one shortcoming of this code .
- Another one is that we are using straight line and which might not always work especially at sharp turn.
- The lane line on road might not always visible due to road condition, like snow, rain or degradation of color, roadwork etc. How can the code to calculate the lane line under such conditions? It is an interesting question for me. Hopefully we can learn this in the following classes.
- These are many parameters, like the kernel_size, low_threshold, high_threshold, etc. I am not sure I am using the best combination of these parameters.

Possible improvements

- If the GPS location and road information could be combined with picture captured by the camera, the region of interest could be optimized based on the road curve.
- The polynomials could be a better alternative than straight line.
- An algorithm to optimize each parameter could be a good choice to reach the best settings instead of "guessing" by programmer.