Finding Lane Lines on the Road

Write up Template

Finding Lane Lines on the Road

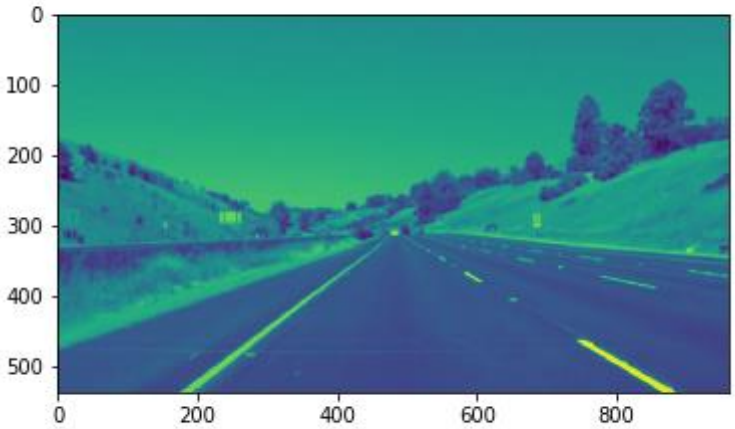
The goals / steps of this project are the following:

- Make a pipeline that finds lane lines on the road
- Reflect on your work in a written report

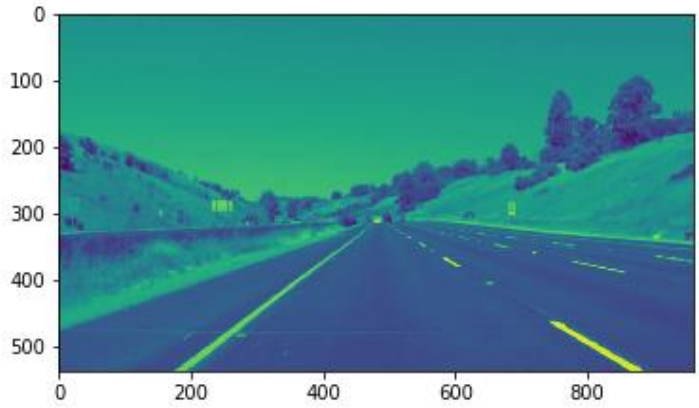
Reflection

My pipeline consisted of 5 steps.

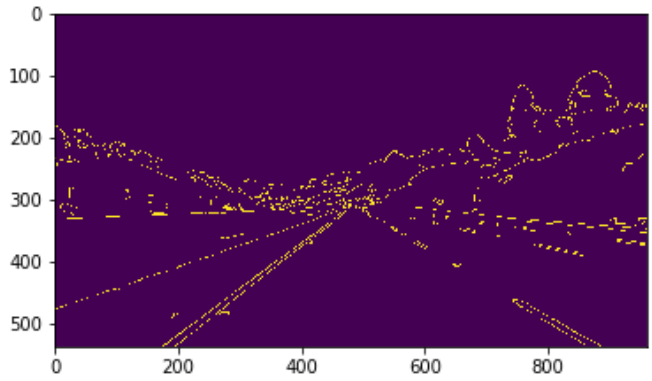
1. I convert the image to **grayscale**. This ensure pixel values stay within 0-255 and provide reasonable parameters to work with for future stages.



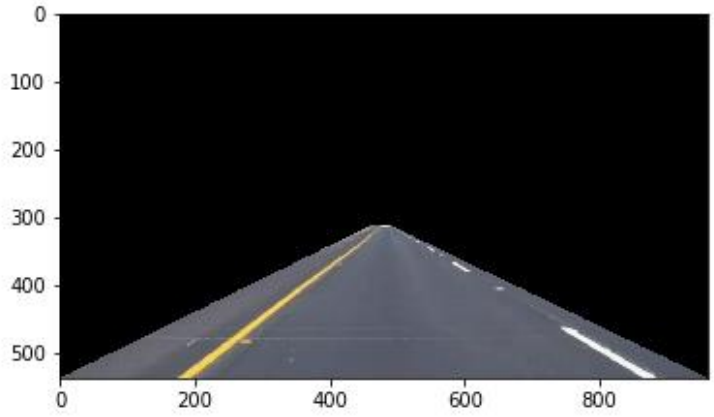
2. Next we shall perform **Gaussian smoothing** for suppressing noise and gradients by averaging. We choose a kernel size of 5



3. **Apply canny edges** – a popular edge detection algorithm. We choose low and high thresholds of 50 and 150 to get to an image similar to the one below.

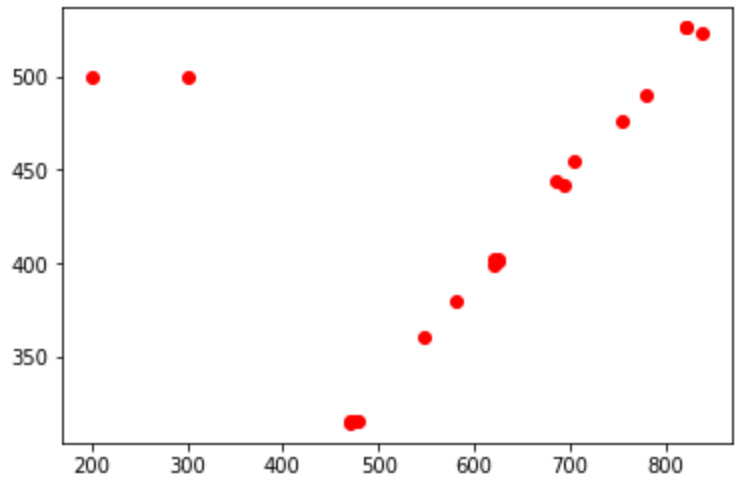


4. Next we define the **region of interest** that we have to carve out so that we focus on the lines we need and eliminate the rest. We start by choosing the edges of the images and adjusting the vertices of the polygon (trapezium) until desired. See below.

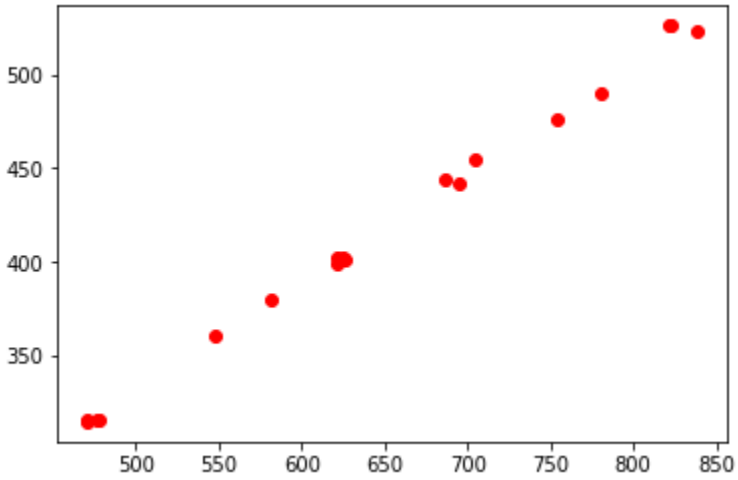


5. We then **derive Hough lines** by adjusting the following parameters.
    a. Parameter List
        i. rho = 2
        ii. theta = (PI/180)
        iii. threshold = 30
        iv. Minimum line length = 60
        v. Maximum line gap = 50

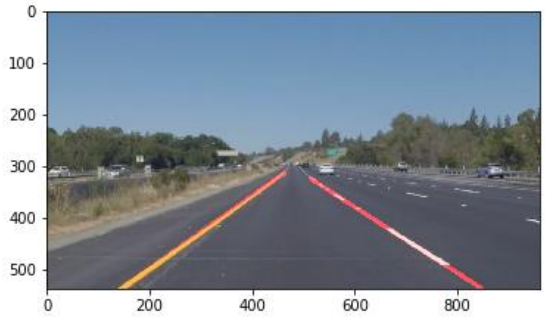
6. After applying the Hough Transform we needed to **extrapolate the line segments** you detect to map out the full extent of the lane. In order to do that I modified the draw_lines() function by following the steps outlined:-
        i. Get the slope of a Hough line
        ii. Bucket LEFT and RIGHT line segments (Negative slopes indicates LEFT segments and positive the RIGHT)
        iii. For both LEFT and RIGHT segment lines
            1. Use all the X,Y points and fit a polynomial (line) using the numpy “polyfit” function
            2. Then use the numpy “poly1d” function to get the actual y=mx+b function.
            3. This function will be used to fit all points that are outside the dataset i.e. that are not returned by Hough function.
            4. Generate points outside the dataset to extrapolate.
        iv. The algorithm above works for most cases but it does not the “solidYellowLeft.mp4” video. This needed careful analysis why the pipeline was failing. By analyzing the X Y points at problematic frames in the video we found below:-

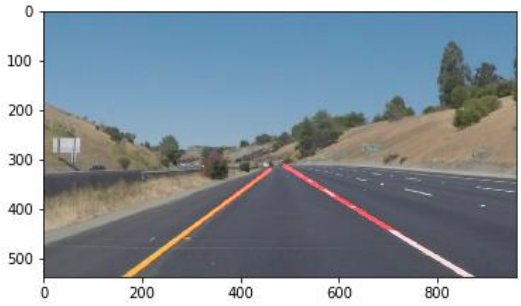


1. There are X and Y points that are outliers that needs to be limited. One more interesting observation is that slopes calculated for these points are WAY less than the average slopes of all other points. Henceforth we can conclude we need to eliminate outliers by ignoring all points that have slopes lesser than average by a certain threshold. By trial and error, the threshold was set to 0.2. Now by print the points we get the below chart which validates our logic and the pipeline "works" for the solidYellowLeft.mp4" video.



The final output on all the images are as follows:-





2. Identify potential shortcomings with your current pipeline

The following shortcomings in the pipeline:-

1. The TOP co-ordinates of LEFT and RIGHT segments does not match perfectly.
2. The pipelines only works for lanes with straight lines. Curves are not captured correctly.
3. The pipelines only works for image 960x540 (since we have hardcoded our region of interest)

3. Suggest possible improvements to your pipeline

The following are the possible improvements:-

- Smoothen the top edges so that marked lines on video looks pleasant.
- Try to edit the Hough transform function to have it working for curved lines (such as challenge.mp4)
- Try to generalize the vertices so that we can define the region of interest better and hence works for all image resolutions