Finding Lane Lines on the Road

Writeup – Juan Cheng 2017.07.19

Finding Lane Lines on the Road

The goals / steps of this project are the following:

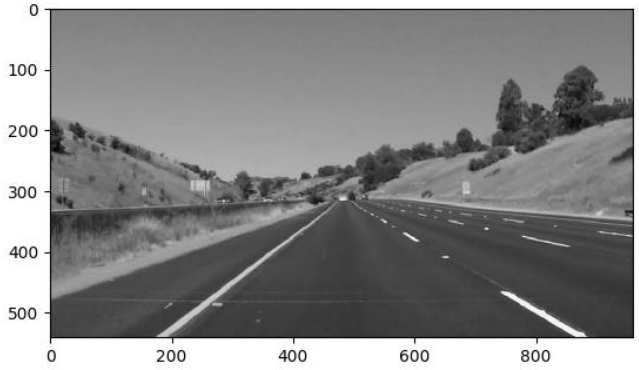
- Make a pipeline that finds lane lines on the road
- Reflect on your work in a written report

Reflection

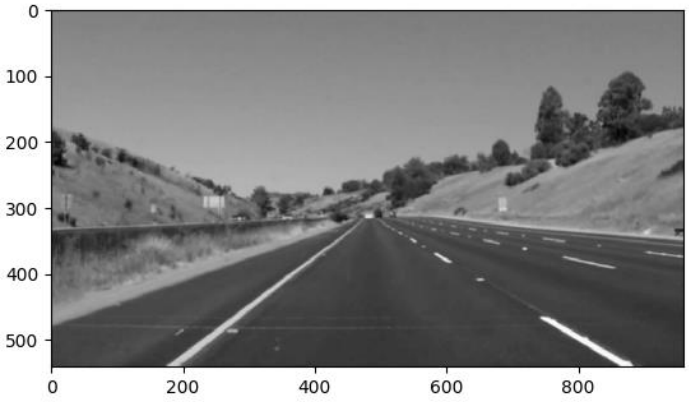
1. Describe your pipeline. As part of the description, explain how you modified the draw_lines() function.

My pipeline consisted of 6 steps.

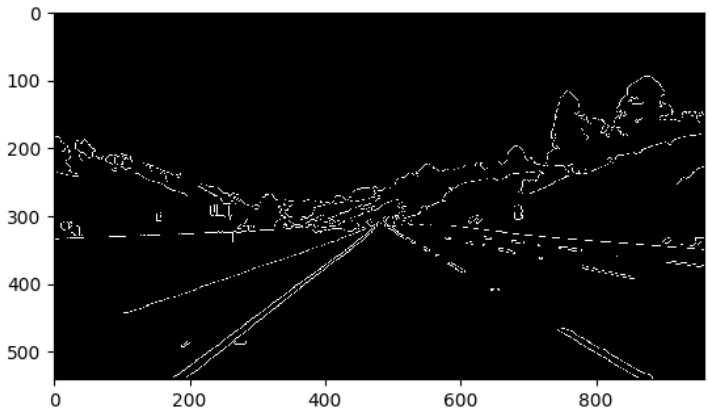
1. Convert the color image to grayscale



2. Apply Gaussian smoothing



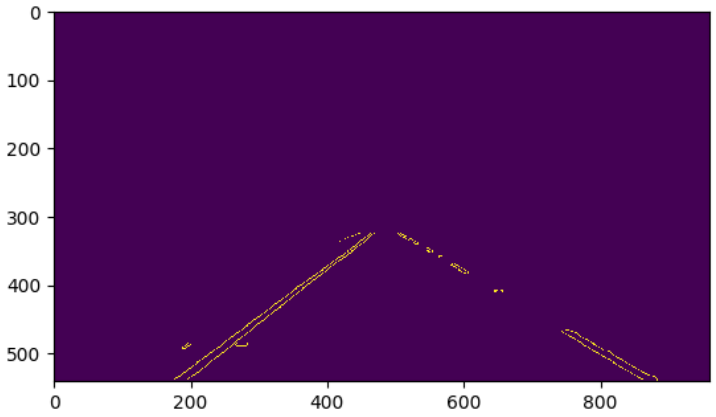
3. Apply Canny Edge Detector



4. Create a masked edges using trapezoid-shaped Region of Interest (ROI)

The way to calculate vertices is key to find the best fit of ROI. I started with hard-coding x, y coordinate values for the top left, top right, bottom left, bottom right points. It didn't work well. I then used width of top/bottom edge of trapezoid, expressed as ratio (0-1) of image width, and height of the trapezoid, expressed as

ratio (0-1) of image height to calculate those 4 points. It worked much better and it's more adaptive to image size.



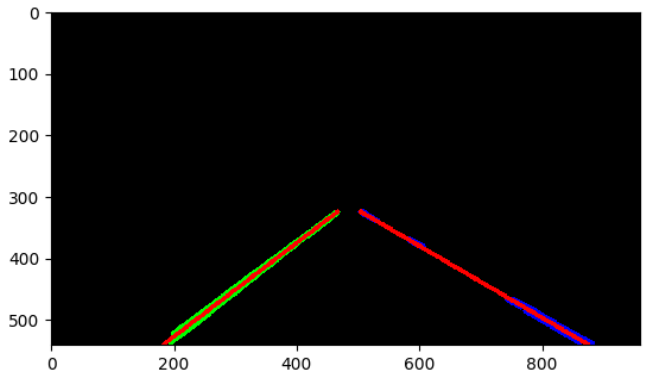
5. Run Hough on edge detected image, return an image with Hough lines drawn

In order to draw a single line on the left and right lanes, I modified the draw_lines() function as follows:

1) Begin with error case checkup, don't draw lines if the length of lines is 0.

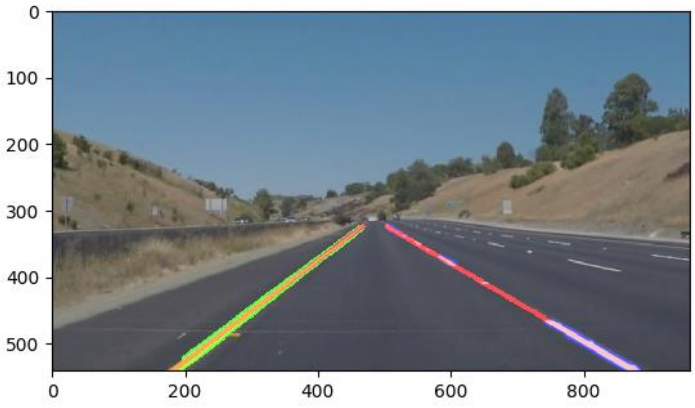
2) Split lines into left and right lines. Do a sanity check for infinite slope and filter out line segments based on bounding the slope and the line location relative to the image center (i.e. only expect the left line slope on left side of the image and the right slope on the right).

3) Run linear regression to find the best fit line for the left and right lane lines. Use 1 level least squares polynomial fit to extrapolate lines.



In the plot above, the green lines are the selected Hough segments for the left line fit and the purple lines are the same for the right line fit. The red lines are the final extrapolated lane lines. These extra visualization aids were included for debugging purposes.

6. Draw lane lines on the original image (including the same debugging lines from step 5).



2. Identify potential shortcomings with your current pipeline

The region of interest works for the first two videos, but doesn't fit well for the challenge one. I plot the region of interest (ROI) image on the videos and find out that the ROI is misaligned with area that turns sharply. It would be better if ROI can be more adaptive in size and shape for changing regions.

The current Hough detector looks for straight lines. The lane lines for the challenge video are notably curved resulting in changing slopes entering the line fitting algorithm.

Also, the slope filtering is not perfect. I see some noise line segments get picked up and produce some glitches for the line fitting.

3. Suggest possible improvements to your pipeline

A possible improvement would be to use a dynamic region of interest that can be adaptive to the changing regions, such as shifting left or right to accommodate curves and automatically adjust the size and edge. On a fully autonomous vehicle, the steering angle could possibly be used to adjust the region.

Investigate use of the cv2.HoughCircles() method for detection curved lane lines.

Another potential improvement could be to make slope filtering smarter, especially for the noise segments of lines. Use of polar coordinates (line angles and offsets) may help here especially when the vehicle changes lane and the lane lines are vertical. Use of filtering of the lane lines could also be applied as the lines would be expected to change relatively slowly compared to the update rate.

Also, it would be good to dig deeper to the Canny Edge detection and tune the low and high thresholds based on the statistical process of the data. One example where this could be beneficial is the short section in the challenge video where the road surface gets lighter and the yellow lane line become more difficult to detect.