P1 Reflection

Jiyang_wang@yahoo.com

Finding Lane Lines on the Road

1. Goals

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

2. Pipeline

My pipeline consists of 8 steps as follows:

Step 1: Color filtering to get prepared for image gray scaling

We aim to single out white and yellow pixels on the image, which can be done by color filtering over the original RGB images.

Shortcoming: prone to noises introduced by tree shades.

Improvement: As suggested and tested in https://github.com/naokishibuya/car-finding-lane-lines, color filtering over HSL has better result than over RGB. Figure 1 and Figure 2 below provide a comparison. HSL is used in this project.

Figure 1 – coloring filtering over RGB image

Figure 2 - Coloring filtering over HSL image

Step 2: Gray-scale the image

This is a simple step. Below is a picture after gray scaling.

Figure 3 – Gray Scaling

Step 3: Apply Gaussian Smoother

This step is primarily for smoothing out noises.

Below is the result of Gaussian smoothing (kernel size = 15).

Shortcoming: not customized to lane lines

Possible improvement: using a filter oriented to “tilted line” shape.

Figure 4 – Gaussian smoothing

Step 4: Canny Edge Detection

It is important to select the right parameters of Canny edge detection, notably low_threshold and high_threshold, by adjusting their values in order to suppress fake edges while detecting most of the lane line edges. Below shows the result of Canny edge detection with low_threshold = 20 and high_threshold = 60.

Figure 5 – result of Canny edge detection

Shortcoming: a lengthy process of adjusting parameters, but still the result may not be suitable to other more complicated situations in the field.

Possible improvement: We may try Difference of Gaussian (DoG) edge detection, but this is more computation costly.

Step 5: ROI Selection using a polygon

Roughly speaking, we are interested in the area surrounded by the red lines below:

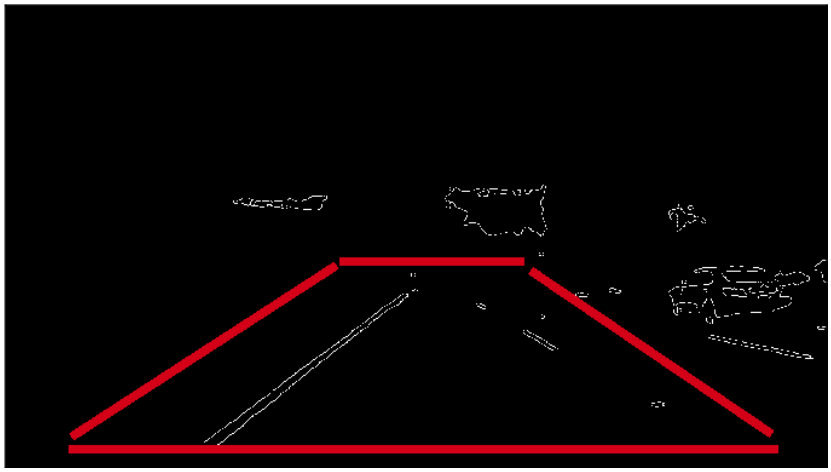

Figure 6 – Region of Interest (ROI)

And the result of ROI selection:

Figure 6 – Result of ROI selection

Step 6 - Apply Hough transform

As of now, those line segments in the image are just pixels to computer. We need to recognize them as two lines: the left lane line and the right lane line.

The first step is to form line segments over those white pixels, each represented by two points (x1, y1) and (x2, y2) This is done by Hough transform whose parameters need to be tuned very carefully. The figure below shows the lines (in red) that Hough transform establishes, with the following parameters:

- rho = 1 # distance resolution in pixels of the Hough grid
- theta = np.pi/180 # angular resolution in radians of the Hough grid
- threshold = 20 # minimum number of votes (intersections in Hough grid cell)
- min_line_length = 20 # minimum number of pixels making up a line
- max_line_gap = 200 # maximum gap in pixels between connectable line segments

Figure 7 – result of Hough transform

Shortcoming: parameters are very sensitive to performance, adjusting which is a daunting task, and still may not be appropriate to many other un-tested cases. Plus, it results in a number of line segments instead of one individual line for each lane, so one more step is needed (see following step).

Possible improvement: The white pixels on the image after Canny edge detection can be "clustered" first, and the centroids of these clusters are used as points for line fitting. This method may be more efficient and more immune to

noises in the case below (the white horizontal line marked in red rectangular) cannot be masked by previous steps because its intensity is high enough to stay.

Figure 8 – A horizontal line segment across the right lane

Step 7 - Averaging line segments

This is the step following Hough transform, which averages over weighted sum of the line segment's slope and intercept with the length of line segment as its weight, so as to form a single line for each lane, represented by (slope, intercept) tuple.

As Hough transform results in a number of line segments of which not all are useful while some may be even misleading, we need to make validity check. Especially, it turns out that we cannot rely only on the positive/negative sign of the calculated slope of each Hough line segment to decide to which side (left lane or right lane) the line segment belongs.

Take figure 8 above as an example. The horizontal line in red area is actually thick enough when the car gets close to it, and thus it gives a number of small line segments by Hough transform with various slopes. If we use slope as the single criteria to decide the side, we'll make wrong decision and the right lane line will be dragged by the line segments in the red area towards the left lane. This would

be catastrophic in the real world if the car follows the lane lines we provide in this way.

Figure 9 shows the result of my code.

Figure 9 – Result of line averaging and extrapolating

Step 8 – Extrapolating lines and draw the full segment of lane lines

Using slope and intercept, we can turn a line to two end points of the line falling in ROI on the image. This process is called line extrapolating. This is the major change to draw_lines() for being able to draw full segment of lane line.

Shortcoming: it uses a straight line even on the curve.

Possible improvement: the curve line may be fitted by a quadratic function in the phase of line averaging.

Final Results:

Three mp3 videos are produced by my code: **sideWhiteRight.mp4**, **sideYellowLeft.mp4** and **challenge.mp4** under directory "./output_videos". Intermediate test results over 8 test images are also shown.