Finding Lane Lines on the Road

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.

The pipeline I used is based on the steps described in the course and it consists of five steps:

1. The image is turned into gray scale to combine the 3 color channels into a single channel to work with.
2. The image is blurred using gaussian blur with a kernel size of 5 pixels to reduce noise in the image.
3. Then I use the Canny edge detection algorithm with the low threshold set to 50 and the high threshold set to 120.
4. The image is then masked using a trapezoid polygon to focus detection only to the region where we anticipate the road lines.
5. I then detect straight lines in the image using the Hough line transform with the parameters set to the following values:

- Distance resolution `rho` set to 1 pixel.
- Angular resolution `theta` set to `pi/90` radians.
- The threshold parameter set to `10`.
- The minimum line length of 100 px.
- The maximum gap in a line of 80 px.

In order to draw a single line on the left and right lanes, I modified the draw_lines() to group the detected lines into those corresponding to the left or right lane by analyzing their slope and then finding the parametric expression of the line using the least-squares method.

Identifying parameters of the left and right lane

To detect, whether a line in the image belongs to the right or left lane, I used the value of the slope of the line:

- for positive slopes, the line belonges to the right lane,
- for negative slopes, the line belongs to the left lane.

I implemented this divison incorrectly at first, because I swapped the negative and positive criterion of the lanes. I expected a line going from the bottom left corner of the picture to the middle of a picture to have a positive slope as it looks like an increasing linear function, which has a positive slope. I then had to remind myself, that the coordinate system is different from the typical cartesian system used in mathematics. The y axis of the picture is pointing downwards and therefore the signs of the slopes are flipped.

To filter out any horizontal and vertical lines, I kept only lines with the slope between 0.4 and 0.8 and -0.4 and -0.8 respectively. I also calculate the average slope of all of the lines belonging to a single lane and remove lines which are not aligned with the rest of the lines. The implementation of these filters is implemented in the `belongs_to_left_lane`, `belongs_to_right_lane`, and `remove_outliers` functions.

For each lane, I then find the parameters `a` and `b` of a line (`y = ax + b`) which best fits all the starting and end points of the lines corresponding to each lane. Using these parameters, I can calculate the end points of the line at the bottom edge of the picture and in the middle of the picture and plot a red line for each of the lanes into the image.

The `draw_lines` function

All of the filtering and parameter detection described in the previous section is used from the `draw_lines` function. I kept the API of all the functions which was given in the Jupyter notebook template but I do not think it is a good way of implementing this algorithm.

Ideally, the parameter detection would be a sixth step of the pipeline and the `draw_lines` function would not call any code related to lane detection and its only concern would be to plot the lines. This separation of concerns would be simpler, easier to read, and easier to cover with unit tests.

2. Identify potential shortcomings with your current pipeline

There are many shortcomings of this algorithm which can all be seen in the output of the `challenge.mp4` test video. The algorithm works well for straight lines (both yellow and white, solid or broken) when the lighting conditions are ideal. As soon as the road is not well lit and the contrast between the lanes and the road surface decreases, the edge detection fails to find an edge and therefore the line detection consequently fails as well.

Another problem arises from noise from shadows and road imperfections which are also detected as straight lines and they interfere with the lane fitting algorithm which then produces incorrect line position and slope.

The third problem of this lane detection algorithm is that it assumes that the lines are straight and therefore it cannot detect any upcoming turns.

3. Suggest possible improvements to your pipeline

Possible improvements of the pipeline would be fine tuning of the parameters of the bluring and edge detection steps. The slope-based filter might also benefit from better tuning in order to reduce false positives.

To fix the third problem, we would have to fit a higher-order polynomial instead of a straight line. To make that possible, the minimum line length parameter of the Hough line transformation would have to be decreased which might lead to more false positive detections of lines outside of the lane markings and it could lead to overall worse performance of the algorithm, so all these changes would need to be made very cautiously.