Project 1 Report

Xiaoyuan Lin

28/01/2021

Refelction

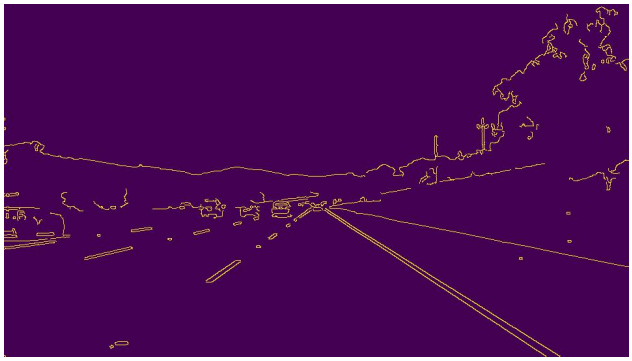
1. Describe your pipeline. As part of the description, explain how you modified the draw_lines() function

The pipeline consisted of the following steps:

1. Transforming the image to grayscale.

2. Gaussian blurring the image, set kernel size to 5.

3. Canny edges detection, calculate the gradient. Modify the upper and lower threshold to get the best possible result.



4. Define ROI and get masked edges.

5. Define Hough space, and Hough transform the masked edges.
6. Generate the merged left and right side lines, adjust the weight -> get the final result.

The idea to generate the result is to calculate an average line using those valid line segments from step 5. There are three ways that I have tried to generate the average line. The first one is calculating slope and intercept for each segment and using the mean function to calculate the average slope and intercept. Then, define y_min and y_max and calculate the x value backward. This method does not perform well for un-solid lines. Therefore, I tried the second method which is using linear regression to calculate the average line. When iterating each line, put the x1, x2, y1, y2 values into two lists, one for x and one for y. Then looping through these two lists and using the linear regression formula to calculate the line which is most suitable for these points. We then use the slope and intercept generated by linear regression and y_min, y_max value we defined before to calculate x values. This method seems to work well for images, but when I tried this on video, it sometimes returned the error that the slope is 0.

To solve this problem, I tried an alternative to calculate the average slope. We get the total length in x-axes and total length in y-axes, and simply calculate the slope by y/x. Then, calculate the average start point. The detailed step for this is: for the two points of the same line, using the average slope and y_max(at the

bottom) to generate two new x values corresponding to y_max, then calculate the average. After having the x_start, simply calculate x_end in the same manner as described above.

For example, if we only have two points(x_1, y_1), (x_2, y_2)

$$x_{start_1} = x_1 + (y_{max} - y_1)$$

$$x_{start_2} = x_2 + (y_{max} - y_2)$$

Then the average x value for the start point is

$$x_{start} = \frac{x_{start_1} + x_{start_1}}{2}$$

. The final start point can be shown as

$$(x_{start}, x_{start} + \frac{y_{min} - y_{max}}{average - slope})$$

. The final line is like $[x_{start}, y_{max}, x_{end}, y_{min}]$

2. Identify potential shortcomings with your current pipeline

1. The selection of the parameters is not really scientific.
2. When there is no x variable in the line, it fails. For example, turning or the road is not straight.
3. The detected line is trembling, need a better way to cancel the noise.
4. There are some better ways to implement the code.

3. Suggest possible improvements to your pipeline

A possible improvement would be to think of a way to tuning the parameters. Another one is to implement a case where the slope is 0, but this would be a little bit complex since we can't simply calculate the average slope this time. Maybe we need to detect corners or something like that. And the code structure can be improved.