PROJECT 1: FINDING LANE LINES ON THE ROAD

Junho Kim*
College of Computer Science, Kookmin University
Seoul, Republic of Korea. 02707
junho@kookmin.ac.kr

ABSTRACT

This project aims to develop an algorithm to finding lane lines on the road from a video stream. Our pipeline handles per image in a video stream in a way that we detect the full extent of the left and right lane lines. Finally, we draw one line for the left side of the lane and other for the right.

1 Introduction

This project aims to develop an algorithm to finding lane lines on the road from a video stream.

We first establish a pipeline about the lane line detection, assembled with Helper Functions that are related to the previous lessons. Next, we will explain our pipeline which detect the full extent of the left and right lane lines. Finally, we will draw one line for the left side of the lane and other for the right.

Basic Pipeline of Lane Finding We built a basic pipeline of lane finding, which is assembled with *Helper Functions*, as follows.

1. Load each image I from `test_images/`.
2. Obtain a grayscale image I_g from I.
3. Obtain a blurred image I_b, by apply Gaussian blur to I_g with the kernel size to 3
4. Obtain an edge image I_e, by apply Canny edge detection to I_b.
5. Compute a masked image I_m from I_e, with a Region Of Interest (ROI) enclosed by three vertices.
6. Obtain a set of line segments L, by computing Hough transform to I_m.
7. Obtain the output image O, by combining L and I

Fig. 1 shows the output images from the basic pipeline. As shown in Fig. 1 the pipeline works well for detecting lane line pixels. However, the goal of this project has not been achieved since the full extent of the lane lines has not yet been detected.

2 Our Pipeline of Lane Finding

Our pipeline is based on the basic pipeline, and we added the two steps for detecting the full extent of the left and right lane lines.

1. Load an image I.
2. Obtain a grayscale image I_g from I.
3. Obtain a blurred image I_b, by apply Gaussian blur to I_g with the kernel size to 3

*Use footnote for providing further information about author (webpage, alternative address)—*not* for acknowledging funding agencies.

Figure 1: Outputs of our basic pipeline: Line segments are handled merely using Helper functions and not classified into left or right lanes.

4. Obtain an edge image I_e, by apply Canny edge detection to I_b.
5. Compute a masked image I_m from I_e, with a Region Of Interest (ROI) enclosed by three vertices.
6. Obtain a set of line segments L, by computing Hough transform to I_m.
7. **(added)** Classify line segments in L (see Sec. 2.1)
8. **(added)** Compute the extent of the left/right lane lines (see Sec. 2.2)
9. Obtain the output image O, by combining L and I

2.1 Classifying line segments

We classify each line segment in L into three categories; 1) left lane, 2) right lane, and 3) noises. We simply use the slope of each line segment to classify them. The slope of a line segment is computed by $(y_2 - y_1)/(x_2 - x_1)$, where (x_1, y_1) and (x_2, y_2) indicates the image-space coordinates of two endpoints.

1. L_L: A line segment seems to belong to the left lane L_L, when its slope is less than -0.5.
2. L_R: A line segment seems to belong to the right lane L_R, when its slope is greater than 0.5.
3. L_n: A line segment seems to belong to noises L_n, when its slope is in $[-0.5, 0.5]$

2.2 Extending lane lines

We find a single line segment l_L, which represent the left lane line, by connecting/averaging/extrapolating the line segments in L_L.

1. For each line segment l in L_L, we perform the following processes.
   (a) Extrapolate each line segment l, bounded by the y-values about the apex and the image height.
   (b) Compute x-values about the apex and the image height from the extrapolated line segment.
   (c) Compute the length of l as its weight factor.
2. Consider all line segments in L_L, we compute the weighted average of x-values about the apex and the image height.
3. Set the representative lane line segment, whose endpoints are $(x^{\text{avg}}_{\text{apex}}, y^{\text{avg}}_{\text{apex}})$ and $(x^{\text{avg}}_{\text{height}}, y^{\text{avg}}_{\text{height}})$.

Similarly, we find a single line segment l_R, representing the right lane line from L_R.

Fig. 2 shows the output images from the advanced pipeline of lane finding.

Figure 2: Outputs of our advanced pipeline: Left and right lanes are represented with a single representative line segment, respectively.

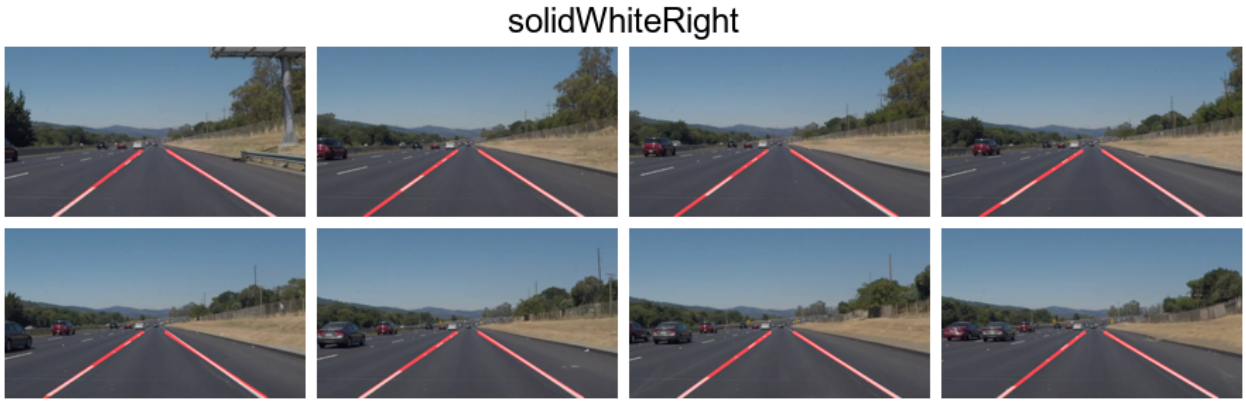

Figure 3: Montage of our outputs for `solidWhiteRight.mp4`

3 Experiments with Video Streaming

We can think of a video stream as a sequence of independent images. In experiments, we apply our advanced pipeline to per frame in the input video and generate the output video, where each frame is processed with our advanced pipeline of lane finding. Fig. 3, Fig. 4, and Fig. 5 show some frames sampled from their output videos.

As shown in Fig. 3 and Fig. 4, our pipeline properly detects the left/right lane lines in `solidWhiteRight.mp4` and `solidYellowLeft.mp4`. However, as shown in Fig. 5, our pipeline didn't work properly when the driving road is curved in `challenge.mp4`.

4 Limitations

The followings are the limitations of our pipeline.

First, our pipeline cannot handle the curved driving road since our algorithm in Sec. 2.2 simply averaging the extent of each line segments in a linear manner. Fig. 5 shows the failure cases in processing curved lane lines. This limitation would be overcome when we use the 2nd or 3rd order polynomial fitting in Sec. 2.2.

Second, there are jittering of the detected lane lines in the output videos, since each frame is independently processed. We believe that jittering would be reduced by utilizing the previous detected lane lines for detecting lane lines for the current frame.

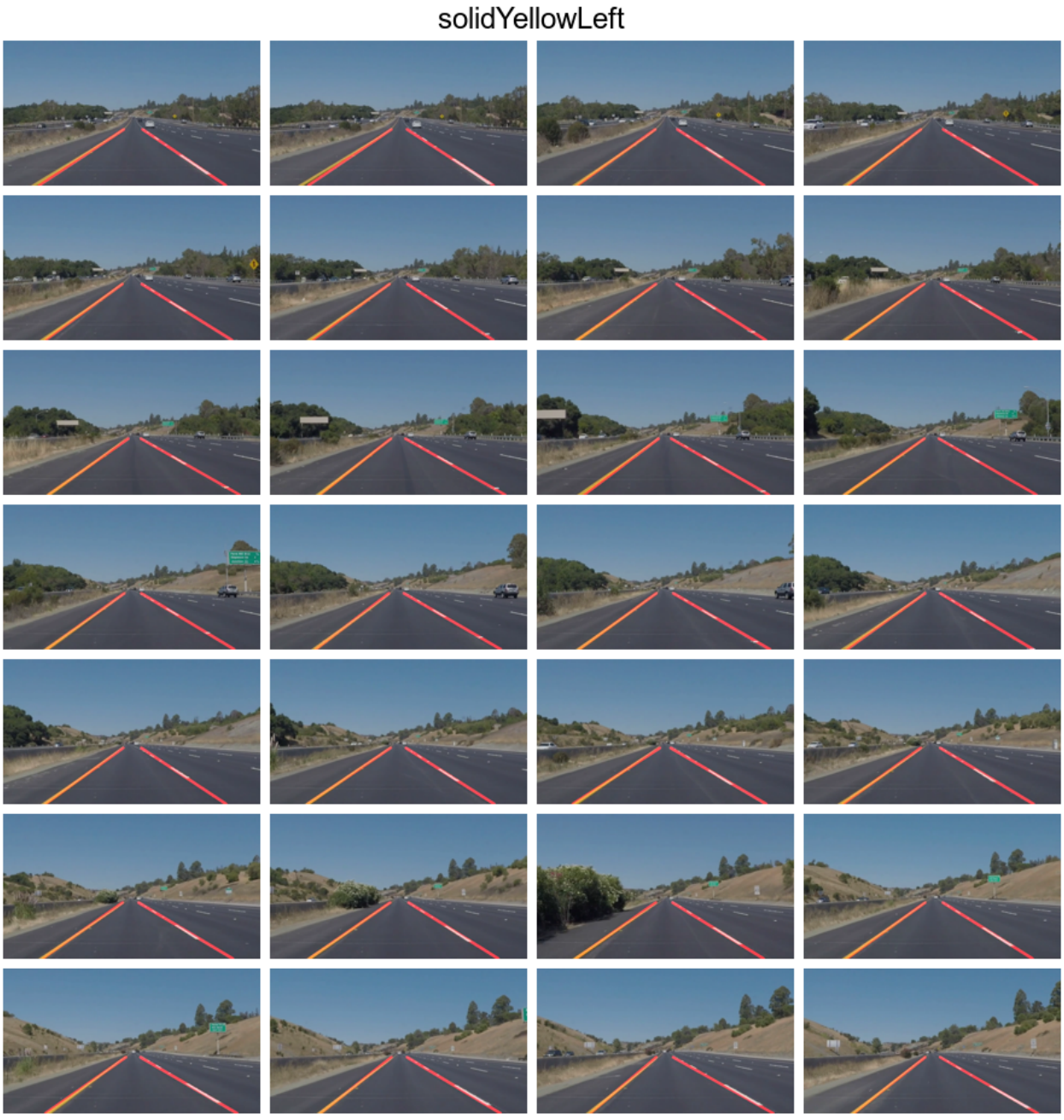

Figure 4: Montage of our outputs for `solidYellowLeft.mp4`

Figure 5: Montage of our outputs for `challenge.mp4`

5 Future Work

The followings include the future directions to improve our algorithms.

- Detecting curved lane lines: When a car runs on a curved road, it is possible to detect the left/right lane lines as two curved lines. To do that, our algorithm in Sec. 2.2 should be replaced with a high order polynomial fitting.
- Minimizing jittering in detected lane lines: Contiguous frames in a video should be processed relevantly. It would be a future direction to adopt Bayesian approach so that the detected lane lines in the previous frame are utilized as the prior knowledge of the current frame. Also, it is possible future direction to utilize filtering techniques, such as Kalman filtering.