DLIP LAB3 : Straight Lane Detection and Departure Warning

Name : HanMinung

ID : 21800773

Class : Deep learning Image processing

LAB : Straight Lane Detection and Departure Warning

School of Mechanical and Control Engineering

2023 Spring Semester

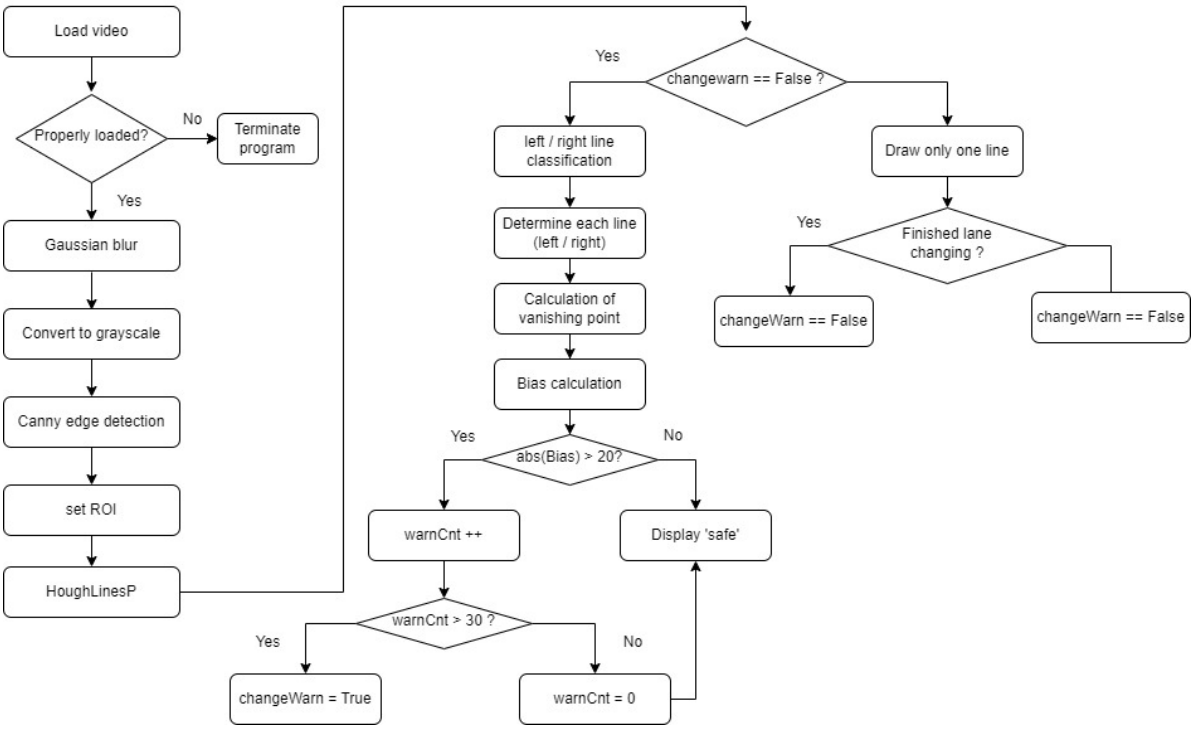
Github : https://github.com/HanMinung/DLIP

---

1. Introduction

Lane detection is a critical process for vehicles, whether they are operated by humans or autonomous driving systems, to properly recognize road lines and maintain their lanes. Recently, there has been active research on achieving optimal lane detection. Lane detection techniques based on CNN, a deep learning method, have been widely studied. In this lab, we utilized a classic image processing technique using Canny edge detection as a preprocessing step and Hough transform based on its result, instead of CNN-based techniques, to perform lane detection. In the following section, the detailed process of the preprocessing and postprocessing applied to the video result will be introduced.

2. Flow chart



1. If the video is loaded correctly, the program proceeds to the next step. Otherwise, the program is terminated.
2. Following that, the program applies frame blur processing and grayscale conversion to prepare the image for Canny edge detection.
3. To avoid detecting unnecessary lines, the program sets a region of interest for the image on which Canny edge detection is performed. The selected region of interest corresponds to one lane within the frontal field of view.
4. The program applies `HoughLinesP` function to the previously set region of interest. Later on, it distinguishes between the left and right lines using the gradient, among other tasks. Further details regarding these processes and their results will be presented in subsequent sections.

3. Preprocessing

3.1. Blur

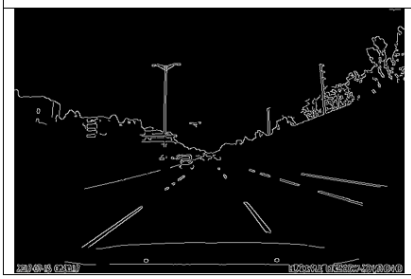
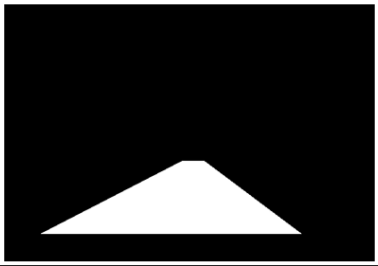
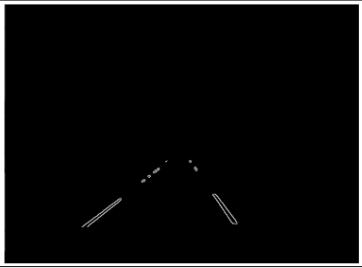
Blur processing is an essential step when processing images or video. To prevent unnecessary detection during post-processing, it is crucial to apply an appropriate blur processing technique to each frame. In this lab, a Gaussian blur with a kernel size of 5 was applied to remove noise present in each frame and enable easy post-processing. The comparison between the original image and the image after blur processing is shown below.

Source frame	Blurred frame
	

3.2. Canny edge detection, masking, ROI

After removing noise through blur processing, Canny edge detection was performed with minthreshold and maxthreshold values selected through the trackbar. The maxthreshold value was set to three times the size of the minthreshold value, with values of 100 and 300 selected, respectively.

Furthermore, To avoid detecting lines in unnecessary areas and significantly increasing the computational load, setting the ROI area was crucial process. The second 'mask result' image in the figure shows the ROI area set, while the first image shows the result of Canny edge detection. The final step involved combining the ROI result with the Canny edge detection result to obtain the lines for the ROI, as shown in the last figure.

Canny edege	Mask result	ROI setting
		

4. Postprocessing

4.1. Line extracting

The line was detected by applying the `HoughLinesP` function after all the preprocessing steps. The following parameter values were set for the function used in this lab. In Hough transform-based line detection, the minimum accumulator value determines the probability of finding candidate groups that can be regarded as straight lines. If the minimum value is set too high, it may result in missing important straight lines. Therefore, the following input parameter values were set to ensure that the function can detect all necessary straight lines.

- minimum accumulator : 10
- minimum line length : 5
- minimum line gap : 200

4.2. Line classification, Vanishing point, Line extending

4.2.1. Line classification

During the lane detection process, multiple lines were detected for a single lane, posing a problem. To solve this issue, the slope of each line was calculated, and the left and right lanes were differentiated based on their pixel coordinates. The left lane has a negative slope, while the right lane has a positive slope. To ensure accurate detection, the left and right lanes were classified separately based on their slope ranges (-85 [deg] to -30 [deg] for the left lane, 30 [deg] to 85 [deg] for the right lane). Next, to detect only one straight line for each lane, the left and right lane lines were separately averaged. This involved averaging all left lane lines and all right lane lines, respectively, resulting in the detection of one straight line for each lane. The selected straight line for each lane can be viewed in the image below (**below 4.2.3. Line extending**).

4.2.2. Calculating vanishing point

In this lab, vanishing point was found with user-defined function `lineIntersection`, `det`. The first step in obtaining the vanishing point is to use the cross product since two points are known for each straight line with the return value of function `HoughLinesP`.

1. A vector that represents each straight line can be expressed with point information on each straight line.
2. By calculating the cross product of two vectors, a normal vector that is perpendicular to both straight lines can be obtained.
3. Later, the vanishing point can be calculated by solving the system of equations obtained from the two equations of the lines that pass through the intersection point and are perpendicular to each line.

$$\vec{v_{left}} = ((x_2 - x_1), (y_2 - y_1))$$
$$\vec{v_{right}} = ((x_4 - x_3), (y_4 - y_3))$$

$$\vec{n} = \vec{v_{left}} \times \vec{v_{right}} \quad (\times means\ \ outer\ \ product\ \ of\ \ two\ \ vectors)$$

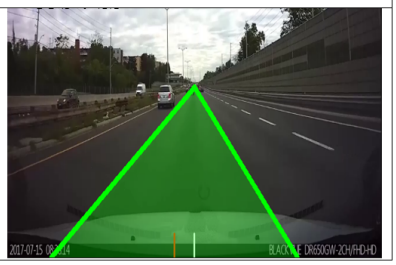
$$line\ 1 : (x, y) = s * \vec{v_{left}} + (x1, y1)$$
$$line\ 2 : (x, y) = t * \vec{v_{right}} + (x2, y2)$$

4.2.3. Line extending

After obtaining information about the vanishing point and the selected lane's points on the line, the `extendLine` function (user-defined) was used to connect the line on the selected lane from the bottom of the image to the vanishing point. This function takes the information of the two lines and the vanishing point as inputs and performs post-processing (line extending) of the information. In case a straight line is not detected while the vehicle is changing lanes and a strange value can be referenced, the process of exception handling is important. The processing steps of this function are as follows :

1. Exception handling: If the vanishing point information is empty, it is intended to use the previous vanishing point value information stored in previous frame.
2. Draw a straight line using the coordinates of the vanishing point and the input straight line point information.
3. Extend the line so that it extends to the bottom of the image.
4. Handle exceptions to prevent NAN values from being returned.
5. Return the two sets of coordinates for the resulting straight line, which are the point at the bottom of the image and the vanishing point.

Each results of the parts like line classification, getting vanishing point, line extending in post processing are as follows :

Detected line	Vanishing point	Extended line
		

4.3. Calculation of bias

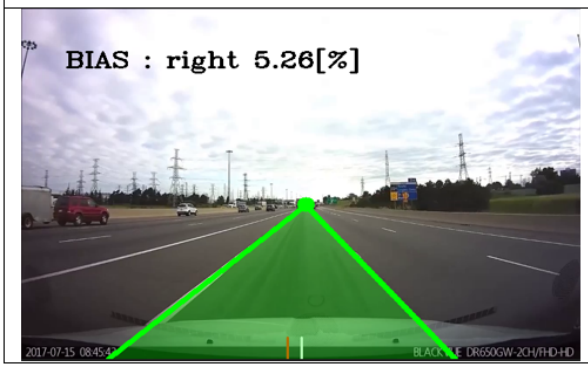
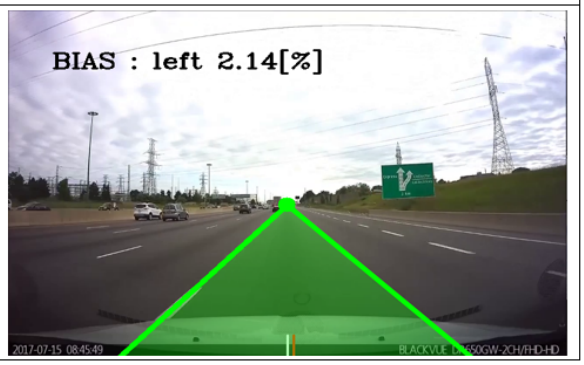
Initially, to calculate the bias and define the center of the road, the center of the road was defined as the midpoint between the final point which locates in the end point with y direction in the image of two detected lanes.

```
roadCen = (line1[2] + line2[2]//2)          # Center X point coordinate
```

However, due to the camera image having a certain amount of offset that determines the field of view, it resulted in a bias that triggered a warning even when the vehicle did not deviate significantly. Therefore, a new method was devised for accurate calculation of bias. As a result of determining the center point of the road using the following method, it was possible to prevent the problem of a warning being displayed even when the bias value is not significant, such as when the vehicle changes lanes.

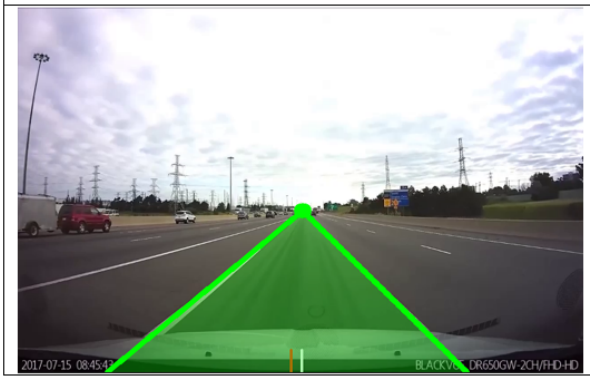
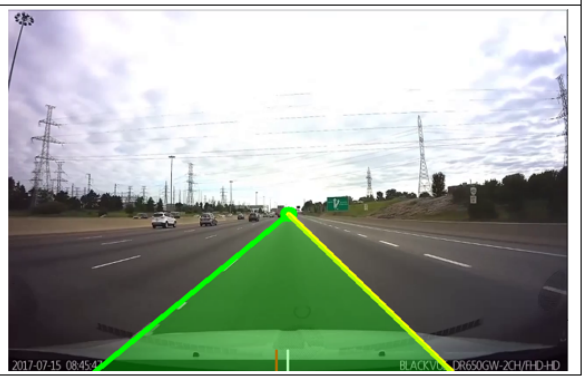
```
line1X = (line1[2] + (line1[0] - line1[2])//4)
line2X = (line2[2] - (line2[2] - line2[0])//4)

roadCen = (line1X + line2X) // 2
vehicleCen = imgWidth // 2
bias = round((vehicleCen - roadCen) / roadCen * 100.0, 2)
```

After calculating the bias and defining it as left bias when negative and right bias when positive, the direction was displayed along with the absolute value in the frame with function `putText`.

Bias calculation (left)	Bias calculation (right)
	

4.4. Further process

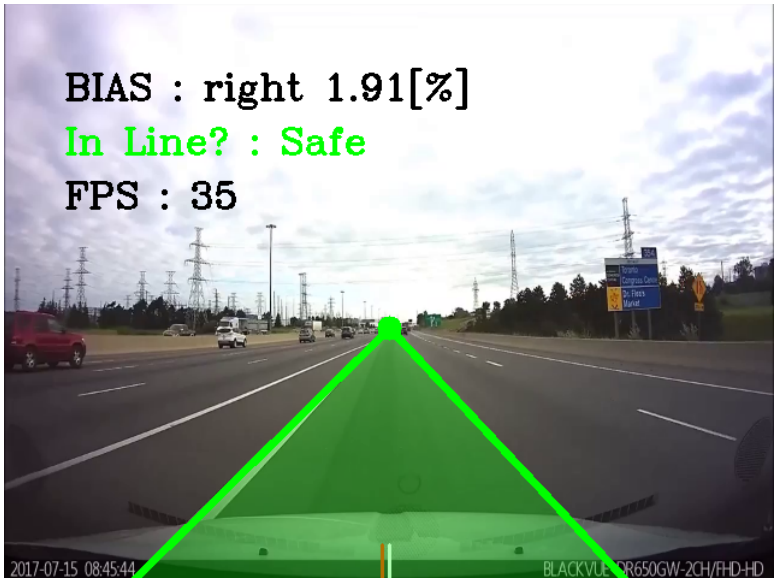
In detecting lanes, there are cases where detection is not done well in some parts, so an appropriate exception handling process is necessary. To address this issue, each time a lane is detected, it is saved as the previous value, and when a lane is not detected, the value from the previous frame is used. For cases where the detection is done well, the lane is drawn in green, and for cases where the previous value is referenced because the detection is not done well, the lane is drawn in yellow. can be observed in the video, it is possible to confirm that the color of a specific line keeps changing because the line is not detected in some parts.

Valid line detection (Green)	Referred line (Yellow)
	

The code includes a check for when the bias exceeds the threshold value for an extended period of time using the variable 'warnCnt'. When this situation persists and the "warnCnt" value exceeds the threshold of 30 (count), it is considered as a lane change and the following state of the vehicle is outputted using the `putText` function. 'Safe' is outputted in normal situations, but when the aforementioned situation occurs, "Lane change" is outputted. The following figure shows the situation for both states.

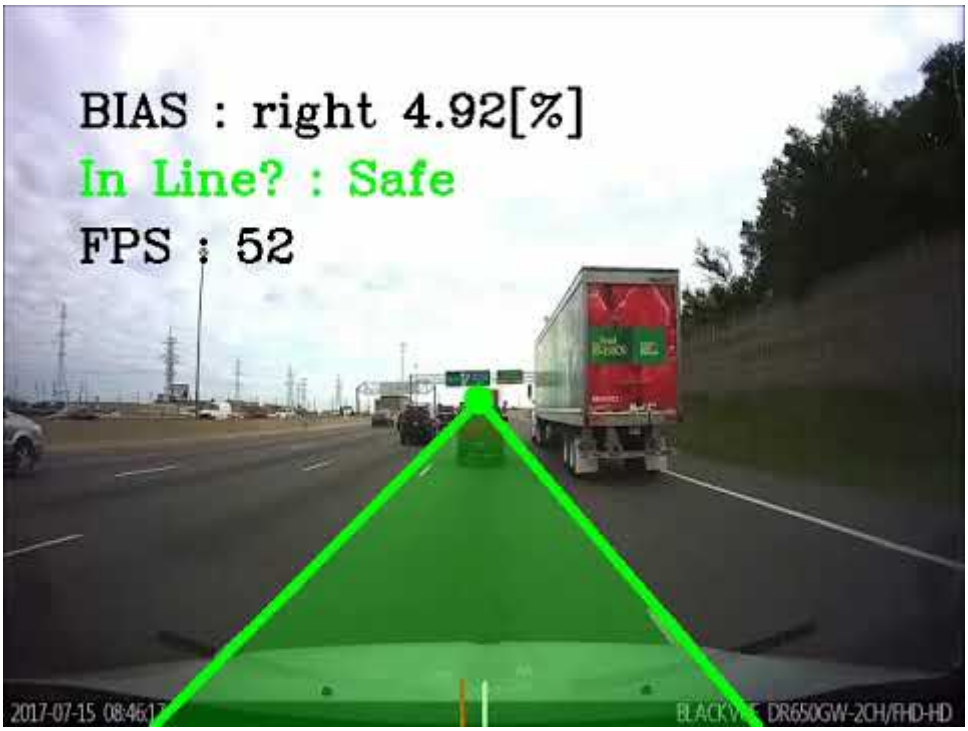
Safe state	Lane changing state
	

In addition, the FPS was calculated in the code. FPS stands for frames per second, which refers to the number of frames captured per second. The code measures FPS using the following method: It measures the start time at the beginning of the loop before image processing, and measures the end time after all processing is complete. Then, the reciprocal of the difference between the two times is calculated to obtain the FPS.



5. Result

Link : https://youtu.be/JpZxD2YS4PE



6. Discussion analysis

The purpose of this lab is to apply the Hough line or stochastic Hough line by performing appropriate pre-processing and post-processing. To remove noise, blur processing was used with a Gaussian filter of kernel size 5. Setting the region of interest (ROI) was also important as it prevented unnecessary line detection in areas outside of the ROI, reducing computation. After appropriate pre-processing, straight lines were detected using stochastic Hough transform, which detects straight lines by using an accumulator. It was important to select appropriate values for the input factor minimum accumulator, minimum length of straight lines to be detected, and distance between straight lines. After detecting straight lines, classification was performed to select one straight line for each lane, and the average value for each class (left, right) produced a single straight line. Vector information for each line was obtained using the pixel coordinate of the end point & starting point of the straight line, and the vanishing point was calculated by performing cross product operation. The process of constructing whole system was built to update the flags and apply the appropriate algorithm for changing lanes with those flags. However, the possibility of overfitting was present, so various lane detection algorithms using deep learning networks and classical image processing should be applied and compared to solve this problem.