LAB 1. Straight Lane Detection and Departure Warning

Date: 2021.04.30
Student ID:
21500482, 21600144
Name: 이두용, 김주호

1. Introduction

This project mainly deals with straight lane detection and departure warning. The ultimate goal is to effectively detect any valid lanes, and any other line components outside the road must be eliminated or neglected. Additionally, if the vehicle is departing from the course, the program must identify whether or not the vehicle is undergoing lane departure, by showing a warning message. Basic preprocessing steps are used, and image processing methods including Canny line detection and Hough transform are newly introduced in this lab. The lab progress algorithm flow chart is in **Figure 1.**

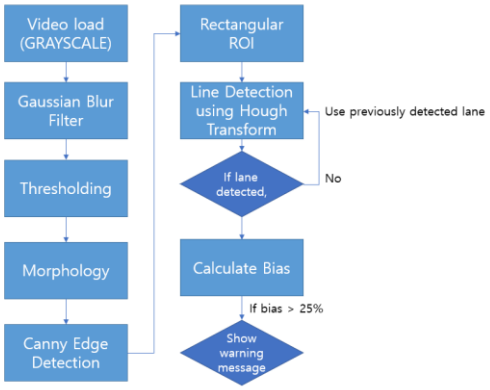


Figure 1. Flow chart of the lab

2. Procedure

2.1 *Preprocessing*

After loading the video in grayscale, several preprocesses must be done. The first thing is to apply Gaussian blur filter to minimize any unnecessary edges to be detected. No big difference between filtered and original video is conspicuous, so figurative demonstration is unnecessary. After filter, thresholding and morphology are applied; thresholding is done using the binary type. Since the outlines were distorted after thresholding, a closed morphology method is used in order to fill in any broken outlines. **Figure 2** demonstrates the images after preprocessing methods. It is obvious that broken lane pieces are solidified for better edge detection.

a) After thresholding b) After closed morphology

Figure 2. Video preprocessing

2.2 *Canny edge detection*

The edges of the preprocessed video are found using Canny edge detection method with threshold range parameters of 130 to 255. The range is found by using the trackbar which sufficiently preserves the unbroken edge outlines. This leads to facilitate more efficient line detection in Hough transformation. The region of interest(ROI) is set to detect any unnecessary line components existing off from the road. The result of Canny edge detection is shown in **Figure 3.**

Figure 3. Edge detected image.

2.3 *Lane detection using Hough transformation.*

In order to set the desirable parameter values in the HoughLineP() function, we first observed the image using the “imagewatch” add-on. As soon as the nearest lane block fades away, the following lane block must be detected as the line in order to maintain stable line detection. Therefore, the distance between the nearest and the following lane blocks is calculated by measuring the pixel coordinate of the two. As a result, the distance range between the lane blocks were 35 to 38, as well as the minimum interception of 25, the minimum line length of 25, and the maximum line gap of 40 where the minimum interception indicates the number of accumulations in the Hough domain. After inputting the parameter values to the HoughLineP() function, the program was able to detect constant lines in each frame. The result of

Hough line detection is demonstrated in **Figure 4.**

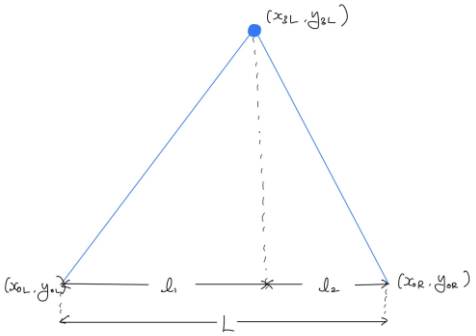
Figure 4. Left and right lane detection colored in blue.

2.4 Line drawing algorithm

Drawing the lines using the lane edges is done by computing the slope of the lines that are defined by the start points (x_1, y_1) and the end points (x_2, y_2) after using the HoughLineP() function. Negative slope indicates that right lane and positive slope indicates the left lane. Valid lines were identified by setting the ranges for the magnitude of both slope values; only the lines having 0.55 to 1 slope magnitudes are used for line drawing, because the actual lanes in the video had slope values of approximately 0.7 to 0.8. Interpreting the simple slope-intercept equation $y = mx + b$, the vanishing point is identified by calculating the interception point of two crossing lines. Then, the conditional statement makes the program to draw only one line per frame only if a valid line is found, by utilizing a switch-like function. In the subsequent frame, the switch variable is set 'true' to process the same algorithm as before. However, if no valid line is identified, the previously detected line is drawn in the original video.

2.5 Departure calculation

How much the vehicle is departed from the lane is computed by calculating the x-axis distance of the vanishing point to the left and right lane, divided by the distance between two lanes. The figurative explanation of lane departure calculation as well as the OpenCV script is shown in **Figure 5.**



a) Vanishing point diagram

```
double bias_int = x3_L;
double bias_val_R = x0_R - bias_int;
double bias_val_L = bias_int - x0_L;
bias_val = (bias_val_R - bias_val_L) / (x0_R - x0_L) * 100;
```

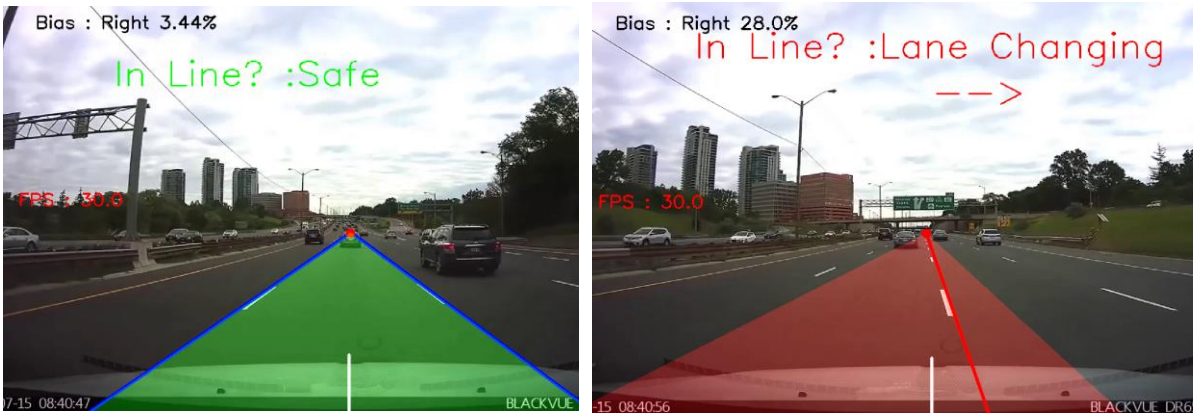
b) Departure rate calculation script in C++

Figure 5. Lane departure calculation.

l_1, l_2, L indicate `bias_int - x0_L`, `x0_R - bias_int`, `x0_R - x0_L` in **Figure 5-b**, respectively, where the variable `bias_int` is the line interception x-coordinate and `x0_L` and `x0_R` are the x starting points of left and right lane. As a result, the departure rate is calculated as below.

$$Rate\ of\ departure\ \% = \frac{l_2 - l_1}{L} \times 100$$

The departure rate is shown in the original video, and the lane changing condition is determined if the departure rate exceeds 25%. In this condition, the message and the road color change from green to red, and the line corresponding to the next detected line is colored in red as well. As the departure rate decreases to the value less than 25%, lane detection returns to its normal state. The final result of this lab is shown in **Figure 6.**



a) In-lane condition b) Lane changing condition

Figure 6. Lane detection result image

3. Conclusion

In this lab, various image processing methods were applied to detect lane in a vehicle-driving video. What made this lab quite intriguing is that such procedures are actively used in autonomous driving system, since accurate lane detection is inevitable for safe driving condition. Same preprocesses were taken place similar to the previous tutorial and lab. In addition to the image processing methods, FPS, frames per second was displayed in order to check whether or not the lane detecting program is adequate enough for other type of videos. This is because 30 FPS is mostly used in lane detection. Although the program we made might not work in other road conditions, learning the basic algorithm of lane detection using Canny edge detection and Hough transform were the main purposes of this lab.

4. Appendix

```
#include "opencv2/imgcodecs.hpp"
#include "opencv2/highgui.hpp"
#include "opencv2/imgproc.hpp"
#include <stdio.h>
#include <iostream>

using namespace cv;
using namespace std;
```

```
std::vector<std::string> block_string(std::string input, int want_block_length)
{
    std::vector<std::string> ret;
    int length;


    std::string str = input;

    while (str.length() > want_block_length)
    {
        std::string msg = str.substr(0, want_block_length);
        std::string rest = str.substr(want_block_length);

        // std::cout << msg << " " << rest << std::endl;
        ret.push_back(msg);

        str = rest;
    }

    // std::cout << str << std::endl;
    ret.push_back(str);

    return ret;
}

int main(int argc, char** argv)
{
    //변수 선언
    Mat image, src, src_gray, dst, cdst, dst_tresh, dst_morph, cdstP, aw_mask;

    // 비디오 load
    VideoCapture cap("Part1_lanechange1.mp4");
    if (!cap.isOpened())   // if not success, exit the programm
    {
        cout << "Cannot open the video cam₩n";
        return -1;
    }
    int key = 0;
    double b, x0_L, y0_L, x3_L, y3_L, x0_R, y0_R, x3_R, y3_R, y3, x_b, y_b;
    x0_R = 0, y0_R = 0, x3_R = 0, y3_R = 0;
    double m_R = 0, m_L = 0, b_R = 0, b_L = 0;
    float bias_val = 0;
    while (1)
    {
        // Check if image is loaded fine
        bool bSuccess = cap.read(image);
        Mat dst_track = Mat::zeros(image.size(), CV_8UC3);
        image.copyTo(src);
        if (!bSuccess)   // if not success, break loop
        {
            cout << "Cannot find a frame from  video stream₩n";
            break;
        }
        key = waitKey(5);
        if (key == 27) // wait for 'ESC' press for 30ms. If 'ESC' is pressed, break loop
```

```
        {
            cout << "ESC key is pressed by user₩n";
            break;
        }

        //이미지 GRAYSCALE 변환
        cvtColor(src, src_gray, COLOR_BGR2GRAY);



        // + Filter
        GaussianBlur(src_gray, src_gray, Size(3, 3), 2);

        // + threshold
        threshold(src_gray, dst_tresh, 130, 255, 0); //binary type, 130
        //imshow("dst_tresh", dst_tresh);

        // + Morphology
        Mat element = getStructuringElement(MORPH_RECT, Size(3, 3));
        morphologyEx(dst_tresh, dst_morph, MORPH_CLOSE, element);

        // Edge detection
        Canny(dst_morph, dst, 130, 255);

        //ROI 설정
        Mat poly_roi;
        Mat poly_mask = Mat::zeros(dst.size(), dst.type()); // ROI 영역만 표시하기위한 검정색
마스크 생성
        Point poly[1][4];

        int H = dst.rows;
        int W = dst.cols;

        poly[0][0] = Point(0, H - 50); //아래 왼쪽
        poly[0][1] = Point(W, H - 50); //아래 오른쪽
        poly[0][2] = Point(W, H / 2); // 위 오른쪽
        poly[0][3] = Point(0, H / 2); // 위 왼쪽
        const Point* ppt[1] = { poly[0] }; // 포인트 값들의 열 (모두 0열)
        int npt[] = { 4 }; // 가장자리의 수
        fillPoly(poly_mask, ppt, npt, 1, Scalar(255, 255, 255), 8);
        dst.copyTo(poly_roi, poly_mask);

        //Hough Transform
        vector<Vec4i> linesP;
        HoughLinesP(poly_roi, linesP, 1, CV_PI / 180, 20, 20, 40); // 최소교차수: 50, 최소길이:
60, 점사이 최대간격: 30
        cout << "line size: " << linesP.size() << endl;

        //Draw the lines
        y3 = H / 2;
        double count_r = 0;
        double count_l = 0;
        double count_b = 0;
        double FPS = 0;
```

```
        for (size_t i = 0; i < linesP.size(); i++)
        {
            Vec4i l = linesP[i];
            double y = l[3] - l[1];
            double x = l[2] - l[0];
            double m = y / x;
            b = l[3] - m * l[2];
            x_b = (H - b) / m;
            if (l[3] > (H / 2) && abs(m) > 0.55 && abs(m) < 1)
            {
                // Line Drawing
                if (m > 0 && count_r == 0) { //right lane
                    m_R = m;
                    b_R = b;
                    y0_R = H;
                    x0_R = (y0_R - b_R) / m_R;
                    x3_R = (b_L - b_R) / (m_R - m_L);
                    y3_R = m_R * x3_R + b_R;
                }
                if (m < 0 && count_r == 0) { //left lane
                    m_L = m;
                    b_L = b;
                    y0_L = H;
                    x0_L = (y0_L - b_L) / m_L;
                    x3_L = (b_R - b_L) / (m_L - m_R);
                    y3_L = m_L * x3_L + b_L;
                }
                if (abs(bias_val) < 25)
                {
                    if (m > 0 && y3_R != 0) {
                        line(src, Point(x0_R, y0_R), Point(x3_R, y3_R), Scalar(255, 0, 0), 3,
LINE_AA);
                        circle(src, Point(x3_L, y3_L), 8, Scalar(0, 0, 255), -1);
                        count_r = 1;
                        //break;
                    }
                    if (m > 0 && y3_R != 0 && count_r == 0) {
                        line(src, Point(x0_R, y0_R), Point(x3_R, y3_R), Scalar(0, 255, 255), 3,
LINE_AA);
                        circle(src, Point(x3_L, y3_L), 8, Scalar(0, 0, 255), -1);
                        count_r = 1;
                    }
                    if (m < 0 && y3_L != 0) {
                        line(src, Point(x0_L, y0_L), Point(x3_L, y3_L), Scalar(255, 0, 0), 3,
LINE_AA);
                        circle(src, Point(x3_L, y3_L), 8, Scalar(0, 0, 255), -1);
                        count_l = 1;
                        //break;
                    }
                    if (m < 0 && y3_L != 0 && count_l == 0) {
                        line(src, Point(x0_L, y0_L), Point(x3_L, y3_L), Scalar(0, 255, 255), 3,
LINE_AA);
                        circle(src, Point(x3_L, y3_L), 8, Scalar(0, 0, 255), -1);
                        count_l = 1;
                    }
```

```
                }
            }
            if (abs(bias_val) > 25 && (x_b < (W / 2) + 200) && (x_b > (W / 2) - 200))
            {
                if (bias_val < 0 && y3_R != 0 && count_b == 0) {
                    line(src, Point(x_b, H), Point(x3_R, y3_R), Scalar(0, 0, 255), 3, LINE_AA);
                    circle(src, Point(x3_R, y3_R), 8, Scalar(0, 0, 255), -1);
                    count_b = 1;
                    //break;
                }
            }

        }

        //FPS display
        string fps = to_string(cap.get(CAP_PROP_FPS));
        string fps_str = fps.substr(0, 4);
        cv::putText(src, "FPS : " + fps_str, Point(100, H/2), CV_FONT_HERSHEY_SIMPLEX,1,
Scalar(0,0,255),2);


        //Bias Calculation
        double center = W / 2;
        double bias_int = x3_L;
        double bias_val_R = x0_R - bias_int;
        double bias_val_L = bias_int - x0_L;
        bias_val = floor(100.00*((bias_val_R - bias_val_L) / (x0_R - x0_L) * 100))/100.00;

        string bias;
        string bias_per;
        string bias_rate;

        if (bias_val > 0)
        {
            bias = "Bias : Left ";
            bias_per = to_string(abs(bias_val));
            bias_rate = bias_per.substr(0, 4);
            cv::putText(src, bias + bias_rate + "%", Point(W * 0.1, 50),
                CV_FONT_HERSHEY_SIMPLEX, 1, Scalar::all(0), 2);
        }
        else
        {
            bias = "Bias : Right ";
            bias_per = to_string(abs(bias_val));
            bias_rate = bias_per.substr(0,4);
            cv::putText(src, bias + bias_rate + "%", Point(W * 0.1, 50),
                CV_FONT_HERSHEY_SIMPLEX, 1, Scalar::all(0), 2);
            cout << bias + bias_per << endl;
        }

        if (abs(bias_val) > 25)
        {
            if (bias_val > 0)
            {
                cv::putText(src, "In Line? :Lane Changing", Point(W * 0.2, 100),
```

```
                    CV_FONT_HERSHEY_SIMPLEX, 2, Scalar(0, 0, 255), 2);
                cv::putText(src, "<--", Point(W * 0.5, H * 0.25),
                    CV_FONT_HERSHEY_SIMPLEX, 2, Scalar(0, 0, 255), 2);
            }
            else
            {
                cv::putText(src, "In Line? :Lane Changing", Point(W * 0.2, 100),
                    CV_FONT_HERSHEY_SIMPLEX, 2, Scalar(0, 0, 255), 2);
                cv::putText(src, "-->", Point(W * 0.5, H * 0.25),
                    CV_FONT_HERSHEY_SIMPLEX, 2, Scalar(0, 0, 255), 2);
            }
        }
        else
        {
            cv::putText(src, "In Line? :Safe", Point(W * 0.2, 150),
                CV_FONT_HERSHEY_SIMPLEX, 2, Scalar(0, 255, 0), 2);
        }

        line(src, Point(W / 2, H), Point(W / 2, H - 100), Scalar(255, 255, 255), 3, LINE_AA);


        // Addweight ROI
        Mat aw_roi;
        aw_mask = Mat::zeros(src.size(), src.type());
        Point aw[1][3];
        aw[0][0] = Point(x0_L, src.rows); //아래 왼쪽
        aw[0][1] = Point(x0_R, src.rows); //아래 오른쪽
        aw[0][2] = Point(x3_R, y3_R); // vanishing point
        const Point* awppt[1] = { aw[0] }; // 포인트 값들의 열 (모두 0열)
        int aw_npt[] = { 3 }; // 가장자리의 수

        if (abs(bias_val) < 25)
            fillPoly(aw_mask, awppt, aw_npt, 1, Scalar(0, 255, 0), 8);
        else
            fillPoly(aw_mask, awppt, aw_npt, 1, Scalar(0, 0, 255), 8);

        Mat src_aw;
        addWeighted(src, 1, aw_mask, 0.3, 0.0, src_aw);

        // Show results
        namedWindow("red_line", WINDOW_FREERATIO);
        imshow("red_line", src_aw);

        // Wait and Exit
        //waitKey();
    }
    return 0;
}
```