2021-1 DLIP LAB #2

Simple Lane Detection

21500264 Seongryeong Park
21600372 Yoonkyoung Song

1. Introduction

The experiment is to create a program that detects lanes at driving video and notifies the degree of lane departure. The purpose of this experiment is to inform the driver by detecting lane departure situations, which are the main causes of vehicle crashes. Nowadays, the need for this program is increasing as the supply of self-driving cars expands. Therefore, learning related algorithms is important. It detects only straight lanes in addition to other environmental variables with images recorded through the camera. Based on the algorithms learned in class, lane detection is detected, and its principles must be deeply understood.

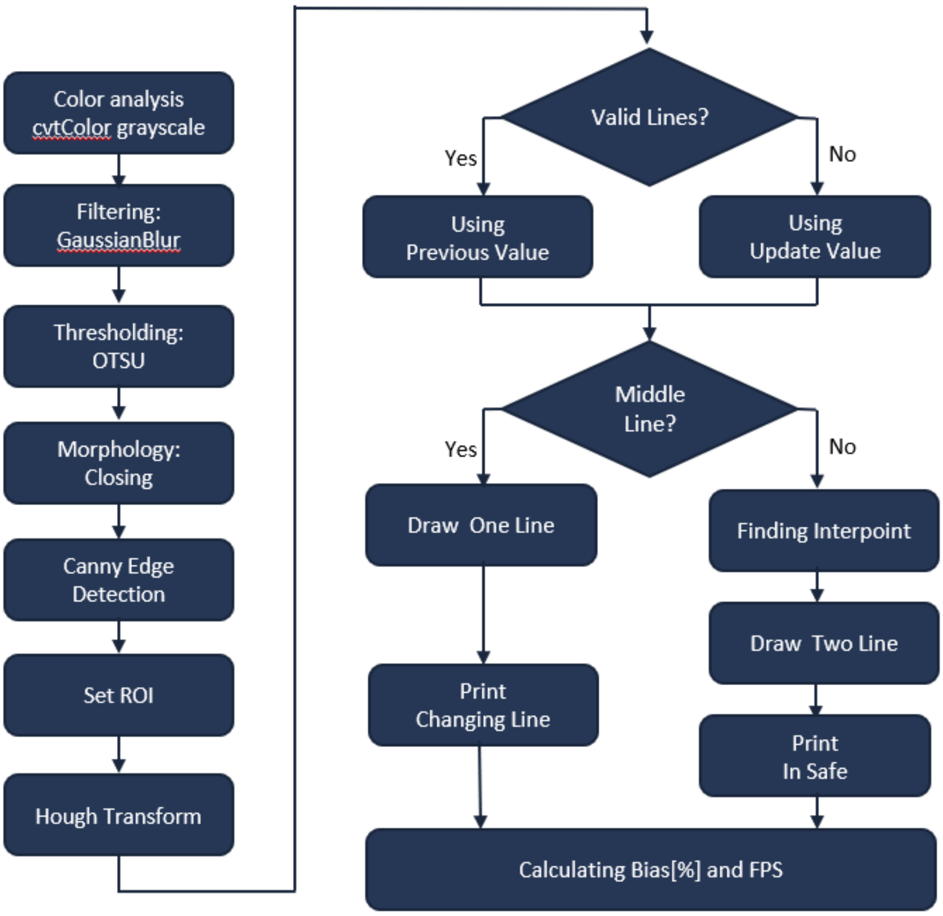


Figure 1. Flow Diagram

2. Procedure

First, we went through a basic preprocessing process to detect lanes. It changed the image of RGB scale to grayscale and used Gaussianblur, the most common method of denoising. Furthermore, as an image segmentation method, we use the Otsu method, which can segment intensities into two classes using histograms of images. Finally, we use morphology, which can improve the shape of objects represented in white in binary images. Among them, closing, which applies erode after dilating, was used to eliminate black noise in the white lane.

Figure 2. Preprocessing Image

Canny edge detection is one of the most widely used edge finding algorithms. We previously used Gaussian blur to remove edges because noisy images make it difficult to find it properly. The gradient is calculated, and non-maxima suppression is applied to detect the edge through thresholding. We set the minimum/maximum thresholding value via C++ and run the algorithm.

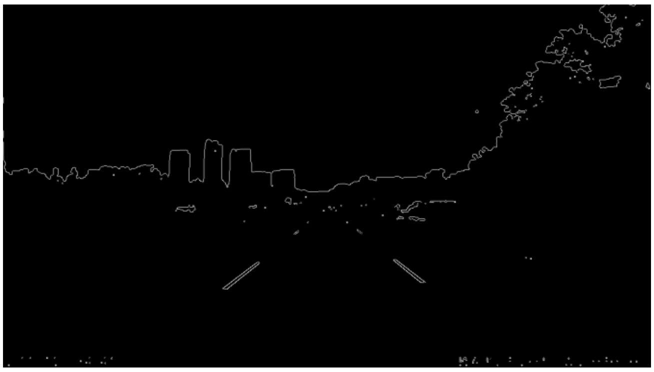


Figure 3. Canny Detection

Applying Hough transform after Canny edge detection has the potential to detect unnecessary straight lines in other sections of the video in addition to lanes. Thus, to detect the lane only in the roadway segment, we made a mask that corresponds to only about half the area of the image and merge the canny edge detection with source video.

(a) mask (b) canny and mask

Figure 4. Detected Lane

Using the ROI image, lines are found when threshold is 65 in the HoughLines function and angles are between 0 and 180 degrees. Then if draw the lines, result is in multiple overlapping lines. Since the required line is a line close to the center of the car, theta returned to find the middle line during the right, left, and lane change was given a condition to find each line, and theta's range to find each line is as follows.

$$35° \leq \theta_{left} \leq 60°$$

$$120° \leq \theta_{right} \leq 145°$$

$$\theta_{mid} < 35°, \qquad \theta_{mid} > 145°$$

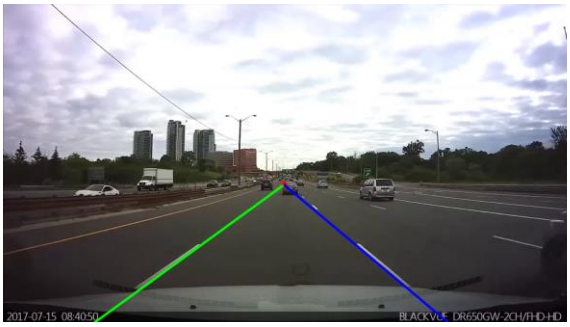
Since only one of the lines in each range has to be found, we receive theta corresponding to each condition and return the nearest line that most strictly regulates for each. Therefore, the right line has the smallest angle, and the left line has the largest angle to find one innermost line in each situation.

Table 1. Flag Setting

<table>
<tr><td>Mode</td><td colspan="2">1</td><td colspan="2">2</td><td colspan="2">3</td><td colspan="2">4</td><td>5</td><td>6</td></tr>
<tr><td>flag</td><td colspan="8">0</td><td colspan="2">1</td></tr>
<tr><td>Left flag</td><td colspan="2">1</td><td colspan="2">1</td><td colspan="2">0</td><td colspan="2">0</td><td>-</td><td>-</td></tr>
<tr><td>Right flag</td><td colspan="2">1</td><td colspan="2">0</td><td colspan="2">1</td><td colspan="2">0</td><td>-</td><td>-</td></tr>
<tr><td>middle flag</td><td colspan="2">0</td><td colspan="2">0</td><td colspan="2">0</td><td colspan="2">0</td><td>0</td><td>1</td></tr>
<tr><td rowspan="2">Draw color</td><td>Left</td><td>Right</td><td>Left</td><td>Right</td><td>Left</td><td>Right</td><td>Left</td><td>Right</td><td rowspan="2">R</td><td rowspan="2">R</td></tr>
<tr><td>G</td><td>B</td><td>G</td><td>Y</td><td>Y</td><td>B</td><td>Y</td><td>Y</td></tr>
</table>

* R, G, B and Y mean color each red, green, bule and yellow.

Table 1 sets the modes using multiple flags, and the modes represent each situation. Flag returns whether a line is found on either side or in the middle, and each left, right, or middle flag returns whether it can find each line in the current frame. If the flag is 0, two lines are drawn, so find the intersection of the two straight lines, find the intersection, and draw the line.



(a) mode 1

(b) mode 2

(c) mode 3

(d) mode 4

(e) mode 5

(f) mode 6

Figure 6. Draw Line

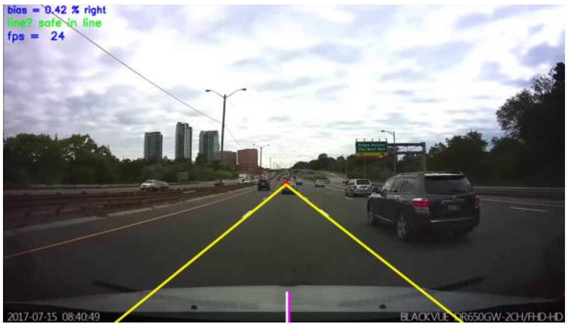
We also calculate the degree of bias in direction through two straight lines. To represent the degree of bias numerically, we compute the width using the two x-pixel value of the points that meet the rows of the images. The bias can be calculated through the difference between the focal point of the two x-pixels and the focal point of the image.

$$bias = \frac{\left(\frac{(x_{right} + x_{left})}{2} - \frac{img.cols}{2}\right)}{x_{right} - x_{left}} \times 100\ [\%]$$

The sign of the bias indicates that the direction of the difference is biased, with bias greater than zero deflecting to the right and bias less than zero deflecting to the left. Subsequently, when the mode is changed by lane change, the last calculated deflection value is output.

Fps calculates the number of frames by accumulating frame counts in less than one second from the start time and resetting the start time and count after one second.

(a) left bias and safety



(b) right bias and safety

(c) line changing

Figure 6. Text Writing of Bias, Safety Check and FPS

3. Conclusion

In this experiment, we considered which image processing techniques should be used to detect lanes in vehicle driving images. In the process, we could recognize that Canny Edge detection and Hough line transform are widely used for lane detection.

To prevent lane departure, the biggest cause of vehicle collision, lanes were detected and how unbiased they were driving within the designated lanes. First, to detect lanes, we first need to eliminate noise through pre-processing tasks such as filtering and morphology. It then learned how to select only the suboptimal information that it wanted to plot in the Region of Interest. In addition, it was designed that how to detect and output information about the desired straight line in the image beyond lane detection. Through this, It was found out the current situation of the vehicle by looking at the line from the image. This exploitation of images to find and specify desired lines will be used in various fields of utilization.

4. Appendix (Code)

```cpp
#include "myOpenCV.h"

using namespace std;
using namespace cv;

float rho_left=0, theta_left= CV_PI;
float rho_right = 0, theta_rigth = 0;
float rho_mid = 0, theta_mid = 0;
bool flag = true;
int fps = 0;
int ffps = 0;
double bias = 0;
double bcent = 0;

cv::Point interP;

int main()
{
    Mat image, image_gray, image_disp, mask, dst;
    vector<vector<Point> > contours;
    vector<Vec4i> hierarchy;


    VideoCapture cap("Part1_lanechange1.mp4");

    bool bSuccess = cap.read(image);

    if (!bSuccess)
    {
        cout << "Cannot find a frame from video stream/n";
    }

```

```cpp
    clock_t start = clock();

    for (;;)
    {
        clock_t fstart = clock();
        cap.read(image);
        image.copyTo(image_disp);

        if (image.empty()) {
            break;
        }

        cvtColor(image, image_gray, COLOR_RGB2GRAY);
        Mat cdstP = image.clone();


        imFilter(&image, &image, 3, GAUSSIAN);
        threshold(image_gray, image_gray, 0, 255, CV_THRESH_OTSU);
        imMorphology(&image_gray, &image_gray, MORPH_RECT, 5, CLOSING);
        Canny(image_gray, image_gray, 100, 150, 3); //canny의 값 조절

        // set ROI
        Point pts[1][4];
        pts[0][0] = { image.cols * 1 / 5 + 50, image.rows * 5 / 8 };
        pts[0][1] = { 0, image.rows - 70 };
        pts[0][2] = { image.cols, image.rows - 70 };
        pts[0][3] = { image.cols * 4 / 5 - 50, image.rows * 5 / 8 };

        const Point* ppt[1] = { pts[0] };
        int npt[] = { 4 };

        mask = Mat::zeros(image.size(), CV_8UC1);
        fillPoly(mask, ppt, npt, 1, Scalar(255, 255, 255), 8);
        //imshow("mask", mask);
```

```cpp
        Mat roiImg;
        bitwise_and(image_gray, mask, roiImg);

        vector<Vec2f> lines;
        HoughLines(roiImg, lines, 1, CV_PI / 180, 65, 0, 0, 0, CV_PI);

        bool leftflag = false;
        bool rightflag = false;
        bool midflag = false;

        //detecting lines
        if (lines.size() > 0) {

            for (size_t i = 0; i < lines.size(); i++) {

                float rho = lines[i][0], theta = lines[i][1];

                if (theta <= CV_PI * 145 / 180 && theta >= CV_PI * 120 / 180) {
                    rho_right = 0, theta_rigth = CV_PI * 2 / 3;
                    if (theta > theta_rigth) {
                        theta_rigth = theta;
                        rho_right = rho;
                        rightflag = true;
                        flag = true;
                    }
                }
```

```
                if (theta >= CV_PI * 35 / 180 && theta <= CV_PI * 60 / 180) {
                    rho_left = 0, theta_left = CV_PI / 3;
                    if (theta < theta_left) {
                        theta_left = theta;
                        rho_left = rho;
                        leftflag = true;
                        flag = true;

                    }
```

```
                if (theta < CV_PI * 35 / 180 || theta > CV_PI * 145 / 180) {
                    rho_right = 0, theta_rigth = CV_PI * 120 / 180;
                    rho_left = 0, theta_left = CV_PI * 60 / 180;

                    theta_mid = theta;
                    rho_mid = rho;
                    midflag = true;
                    leftflag = false;
                    rightflag = false;
                    flag = false;
                }
            }
        }


        /* draw line according to each flag */
        if (flag && (!midflag)) {

            drawinterpoint(&cdstP, &interP, rho_left, theta_left, rho_right, theta_rigth, RED);

            if (leftflag && rightflag) {
                drawline(&cdstP, interP, rho_left, theta_left, GREEN);
                drawline(&cdstP, interP, rho_right, theta_rigth, BLUE);
            }
            else if (leftflag && (!rightflag)) {
                drawline(&cdstP, interP, rho_left, theta_left, GREEN);
                drawline(&cdstP, interP, rho_right, theta_rigth, YELLOW);
            }
            else if ((!leftflag) && rightflag) {
                drawline(&cdstP, interP, rho_left, theta_left, YELLOW);
                drawline(&cdstP, interP, rho_right, theta_rigth, BLUE);

            }
            else if ((!leftflag) && (!rightflag)) {
                drawline(&cdstP, interP, rho_left, theta_left, YELLOW);
                drawline(&cdstP, interP, rho_right, theta_rigth, YELLOW);
            }
```

```
            calculateBias(&cdstP, &bcent, &bias, rho_left, theta_left, rho_right, theta_rigth);

            /*draw bias line*/
            Point pt1(cdstP.cols / 2, cdstP.rows - 70);
            Point pt2(cdstP.cols / 2, cdstP.rows);

            line(cdstP, pt1, pt2, WHITE, 3);

            Point bi(bcent, cdstP.rows - 70);
            Point as(bcent, cdstP.rows);
            line(cdstP, bi, as, PINK, 3);

        }
        else {
            drawinterpointCenterline(&cdstP, interP, rho_mid, theta_mid, RED);
        }



        /*write information in frame*/
        ///bais
        char Massage_bias[50] = "bias = ";
        char sbias[20];
        sprintf(sbias, "%.2lf", bias);
        strcat(Massage_bias, sbias);


        if (bias > 0) {
            char per[20] = " % right";
            strcat(Massage_bias, per);
            putText(cdstP, Massage_bias, cv::Point(10, 20), cv::FONT_HERSHEY_SIMPLEX, 0.7, BLUE, 2);
        }
        else {
            char per[20] = " % left";
            strcat(Massage_bias, per);
            putText(cdstP, Massage_bias, cv::Point(10, 20), cv::FONT_HERSHEY_SIMPLEX, 0.7, BLUE, 2);
        }

```

```
        ///line change
        if(!flag){
            char Massage_safe[50] = "line? change line";
            if (bias > 0) {
                putText(cdstP, Massage_safe, Point(10, 50), FONT_HERSHEY_SIMPLEX, 0.8, RED, 2);
            }
            else {
                putText(cdstP, Massage_safe, Point(10, 50), FONT_HERSHEY_SIMPLEX, 0.8, RED, 2);
            }
        }else {
            char Massage_safe[50] = "line? safe in line";
            putText(cdstP, Massage_safe, Point(10, 50), FONT_HERSHEY_SIMPLEX, 0.8, GREEN, 2);
        }
```

```
		/// fps
		char sfps[20];
		char Massage_fps[50] = "fps =  ";
		sprintf(sfps, "%d", ffps);
		strcat(Massage_fps, sfps);

		putText(cdstP, Massage_fps, Point(10, 80), FONT_HERSHEY_SIMPLEX, 0.8, BLUE, 2);


		/* frame check */
		clock_t finsh = clock();
		clock_t del = finsh-start;
		if (del > 1000) {
			ffps = fps;
			fps = 0;
			start = clock();
		}
		else {
			fps++;
		}

		imshow("roiImg", roiImg);
		imshow("image", cdstP);
		/// 영상 저장
		//outputVideo << cdstP;

		char c = (char)waitKey(10);
		if (c == 27)
			break;

	}

	return 0;
}
```