Lane Departure Warning System for Road Lane Detection

Date: 2023.04.30

Student ID: 21800805

Name: Hwang SeungEun

I. Introduction

In this assignment, we conducted experiments on lane detection and lane departure errors using Python OpenCV. The goal of this experiment is to help improve road safety and assist drivers in maintaining their position within the lane.

II. Procedure

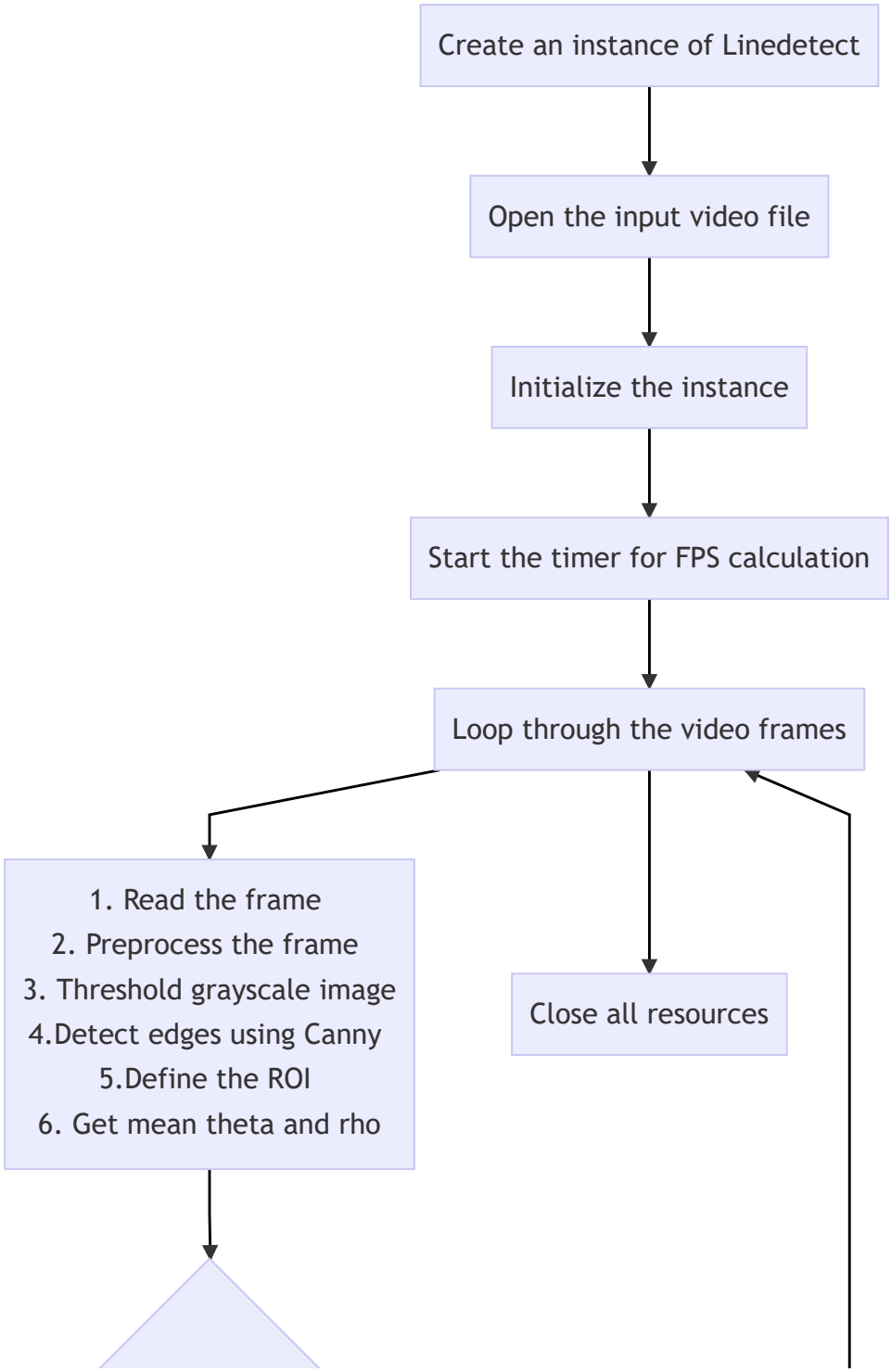
1. Create an instance of `Linedetect`
2. Open the input video file
3. Initialize the instance
4. Start the timer for FPS calculation
5. Loop through the video frames:
    - Read the frame
    - Preprocess the frame
        - Gray scale
        - Thresh hold
        - Canny
    - Define the ROI
    - Find and draw lines with `cv2.HoughLines` on the frame
    - Calculate and display the FPS
    - Show the final output frame
    - Write the result to the output file
    - Handle keyboard input for stopping or pausing the video playback
6. Close all resources

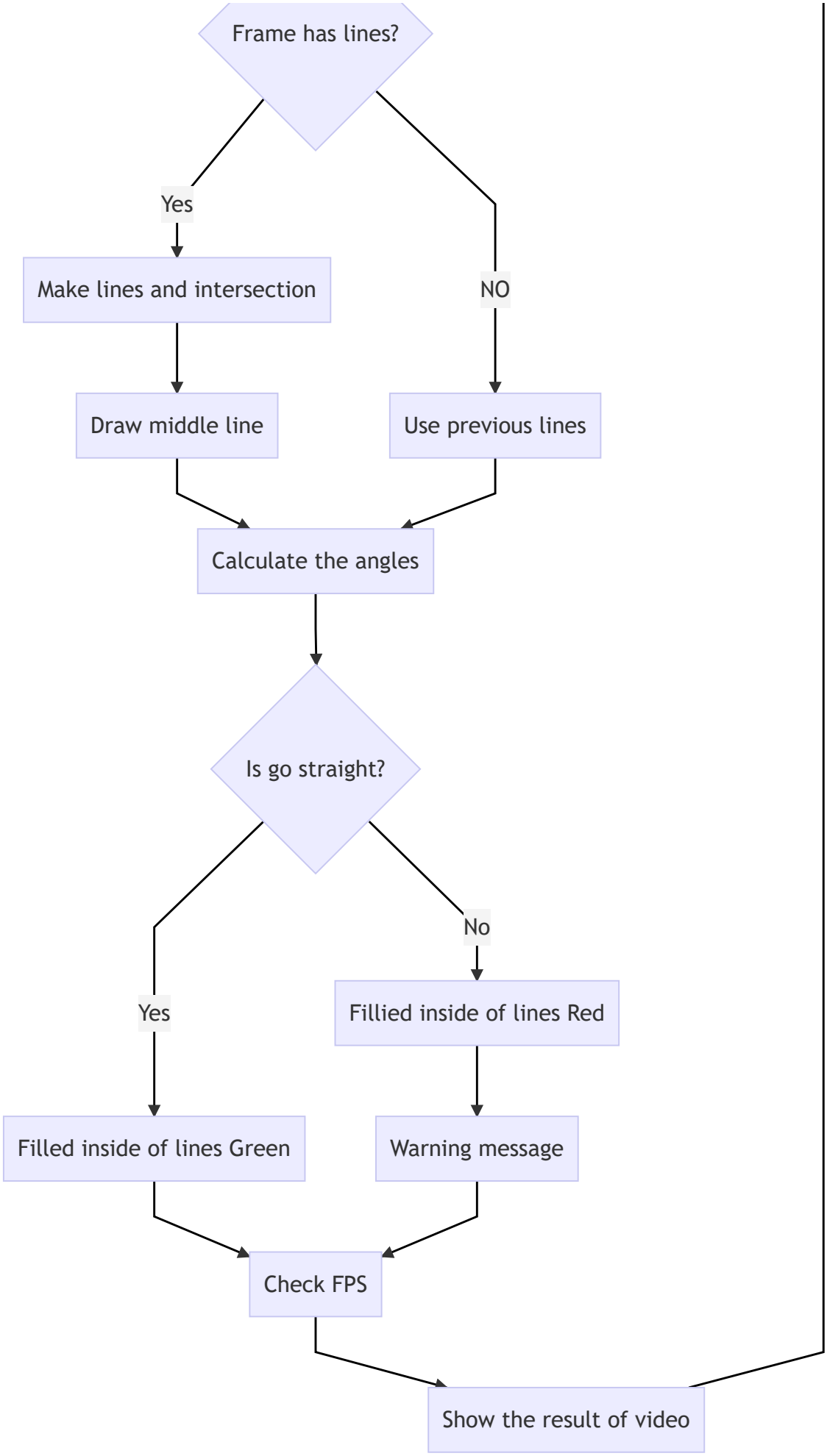
Explain about class of `Linedetect`

- `Initialize()` : Set up capture properties, create trackbars and initialize video writer
- `Trackbar()` : Callback function for trackbars
- `Preprocessing()` : Convert the frame to grayscale, apply binary thresholding and perform edge detection using Canny
- `ROI()` : Define the region of interest (ROI) and apply it to the Canny image
- `getLines()` : Find lines in the ROI using HoughLines, classify them as positive or negative slopes, calculate their intersection point, and draw the lines

- `Intersection()` : Calculate the intersection point of two lines
- `drawLines()` : Draw the positive and negative lines, and the midline
- `getBias()` : Calculate the bias between the lines and determine the warning message
- `showVid()` : Display the video frames
- `outResult()` : Write the result to the output file
- `closeAll()` : Close video writer
- `startTime()` : Record the start time for FPS calculation
- `FPS()` : Calculate and display the frames per second (FPS)

III. Flow Chart




Frame has lines?
Yes
NO
Make lines and intersection
Draw middle line
Use previous lines
Calculate the angles
Is go straight?
No
Yes
Fillied inside of lines Red
Filled inside of lines Green
Warning message
Check FPS
Show the result of video

IV. Experiment

1. Create the Instance

Declare an instance of the class Linedetect.

```
class Linedetect:
    ...

def main():

    LD = Linedetect()
    ...
```

2. Load a Road Video

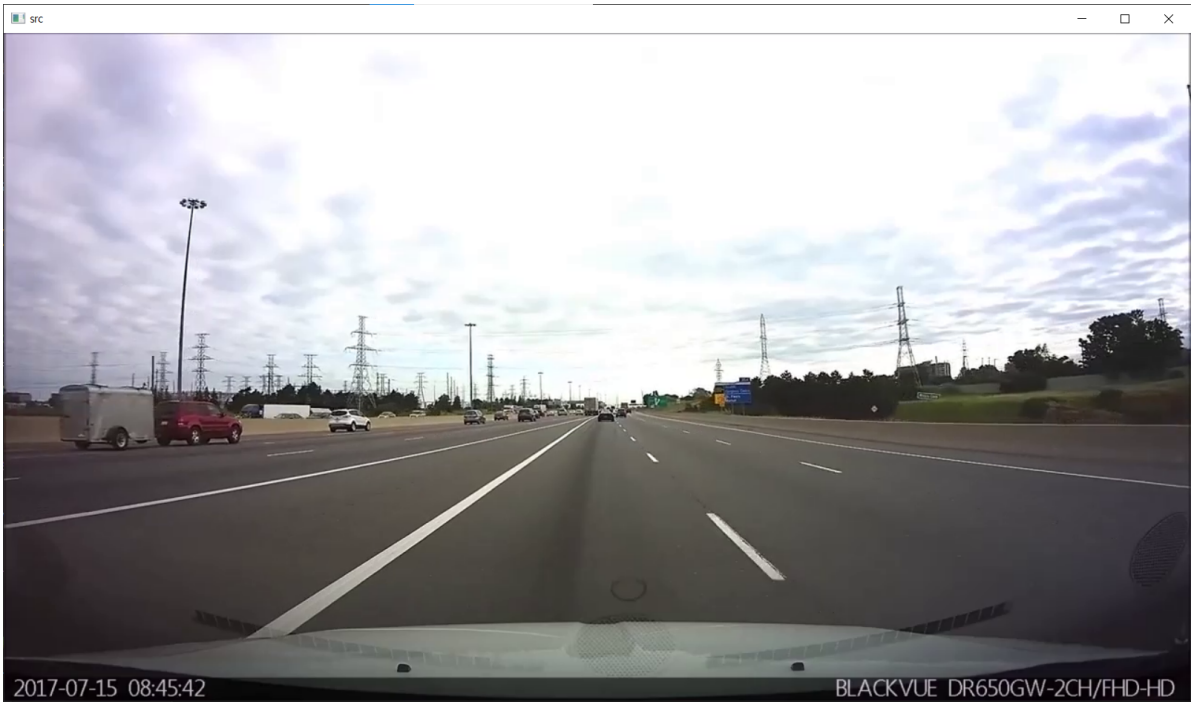


Figure 1. Source Video

```
cap = cv.VideoCapture('road_lanechange_student.mp4')
```

3. Initialization and Start time

Before starting the while loop, initialize the video's width and height, declare the track bar for use in preprocessing, and set up the necessary information for outputting results

```
class Linedetect:
        def Initialize(self, cap):

        if not cap.isOpened():
            raise FileNotFoundError("Video file is not Found!")
```

```
        self.width = int(cap.get(cv.CAP_PROP_FRAME_WIDTH))
        self.height   = int(cap.get(cv.CAP_PROP_FRAME_HEIGHT))

        cv.namedWindow('Track bar', cv.WINDOW_NORMAL)
        cv.createTrackbar('Canny Low', 'Track bar', self.canny_low, 200,
self.Trackbar)
        cv.createTrackbar('Canny High', 'Track bar', self.canny_high, 200,
self.Trackbar)
        cv.createTrackbar('Hough Threshold', 'Track bar', self.hough_thr, 200,
self.Trackbar)
        cv.createTrackbar('Binary Threshold', 'Track bar', self.binary_thr, 200,
self.Trackbar)

        self.fourcc = cv.VideoWriter_fourcc(*'XVID')
        self.output_filename = 'output.mp4'
        self.fps = 30
        self.frame_size = (self.width, self.height)
        self.out = cv.VideoWriter(self.output_filename, self.fourcc, self.fps,
self.frame_size)

    def Trackbar(self, val):
        return val

        def startTime(self):
        self.start_time = time.time()
        self.prev_time = self.start_time

def main():
    ...
    LD.Initialize(cap)

    LD.startTime()
    ...
```

4. Loop through the video frames

4.1 Read the frame and check usable

In the while loop, the progress of the code is determined based on the presence or absence of frames in the cap.

```
def main():

    # procedure 3

    while(cap.isOpened()):

        ret, frame = cap.read()

        if not ret:
            break
```

4.2 Preprocessing

Utilize the track bar to assign appropriate values to each parameter and perform preprocessing.

Figure 2. Preprocessing with Trackbar

```
class Linedetect:
    def Preprocessing(self, _frame):
        self.canny_low = cv.getTrackbarPos('Canny Low', 'Track bar')
        self.canny_high = cv.getTrackbarPos('Canny High', 'Track bar')
        self.hough_thr = cv.getTrackbarPos('Hough Threshold', 'Track bar')
        self.binary_thr = cv.getTrackbarPos('Binary Threshold', 'Track bar')

        self.gray = cv.cvtColor(_frame, cv.COLOR_BGR2GRAY)
        _, self.thresh = cv.threshold(self.gray, self.binary_thr, 255,
cv.THRESH_BINARY)
        self.canny = cv.Canny(self.thresh, self.canny_low, self.canny_high)

def main():

    # procedure 3

    while(cap.isOpened()):

    # procedure 4.1

    LD.Preprocessing(frame)
```

4.3 Reason of Interest (ROI)

Set the Region of Interest (ROI) so that only the current driving lane is visible as much as possible. The area of interest was created in a triangular shape.

Figure 3. Reason of Interest

```
class Linedetect:
    def ROI(self):

        self.mask = np.zeros_like(self.canny)
        # self.roi  = np.array([[(50, self.height - 90),(self.width/2-40,
self.height/2+50),
        #                    (self.width/2+20, self.height/2+50), (self.width-
50,self.height - 90)]], dtype=np.int32)
        self.roi  = np.array([[(50, self.height - 90),(self.width/2,
self.height/2+60),
                           (self.width/2, self.height/2+60), (self.width-
50,self.height - 90)]], dtype=np.int32)
        cv.fillPoly(self.mask , self.roi, 255)
        self.roi = cv.bitwise_and(self.canny, self.mask)

def main():

    # procedure 3

    while(cap.isOpened()):

    # procedure 4.1
    # procedure 4.2

    LD.ROI()
```

4.4 Get line with function HoughLines

Use HoughLines to output the lines in the ROI area. The figure below shows an image that confirms whether the overall lines are displayed correctly. Once all the lines are printed within a single frame, divide the positive and negative slopes and store them separately. Calculate the average values of theta and rho and save them. If no lines are extracted from the current frame, use the average value from the previous frame.

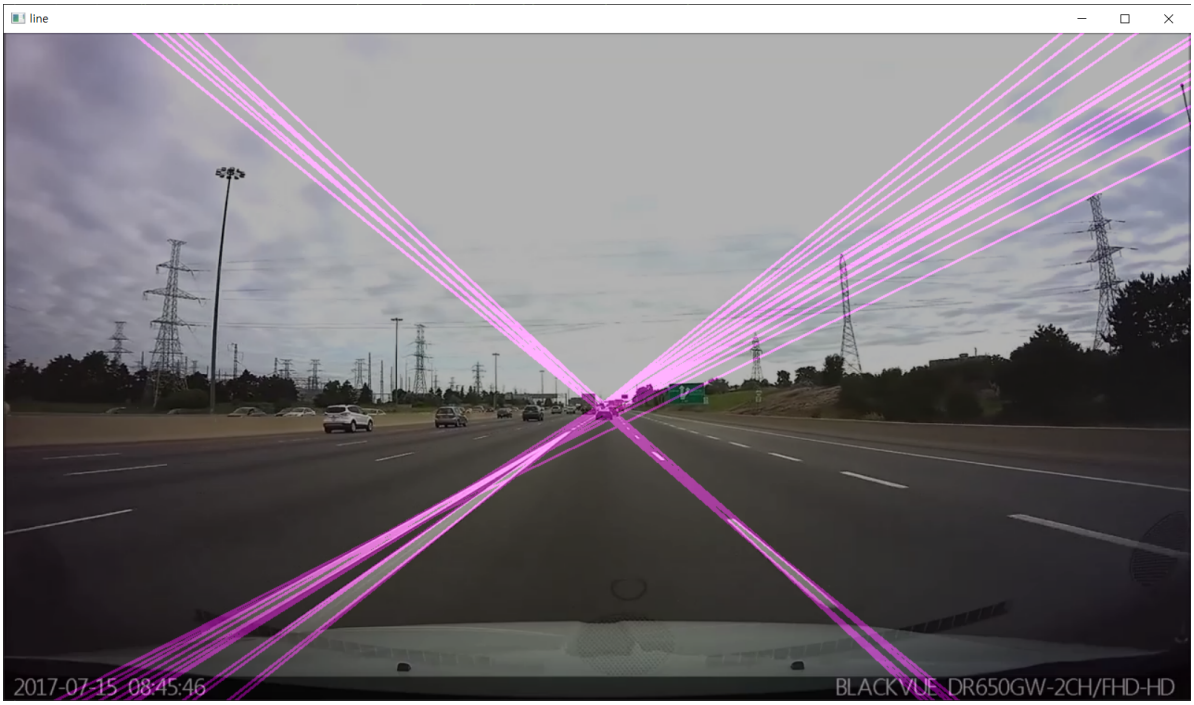


Figure 4. Get All lines with HoughLines

Compute the intersection point using the resulting left and right lane lines.

4.5 Get mean lines and Intersection

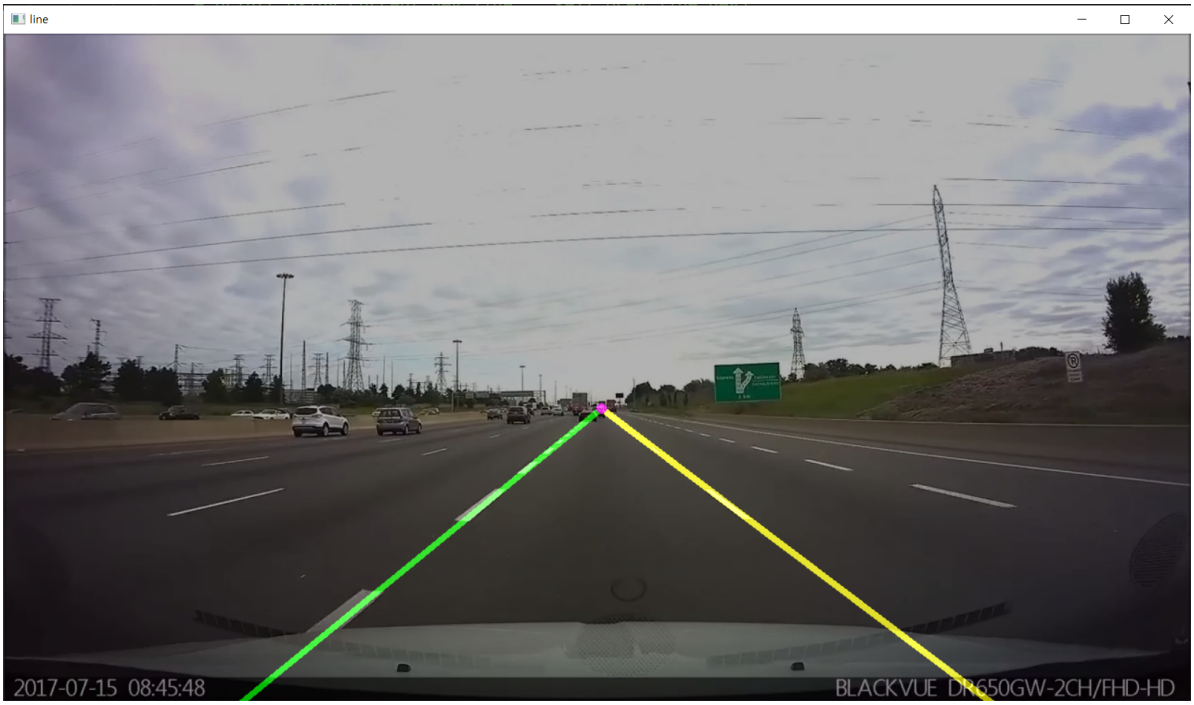


Figure 5. Get Intersection Point

Figure 5 above shows the intersection of the lines obtained through the average slope of the lines in Figure 4 and the average line. When drawing the lines, it is crucial to prevent the slope from diverging to infinity when it reaches 90 degrees. Also, when the line is not detected in the current frame, it is displayed in yellow like the right lane, and when detected, it is displayed in green like the left lane.

4.6 Draw middle line of two lines

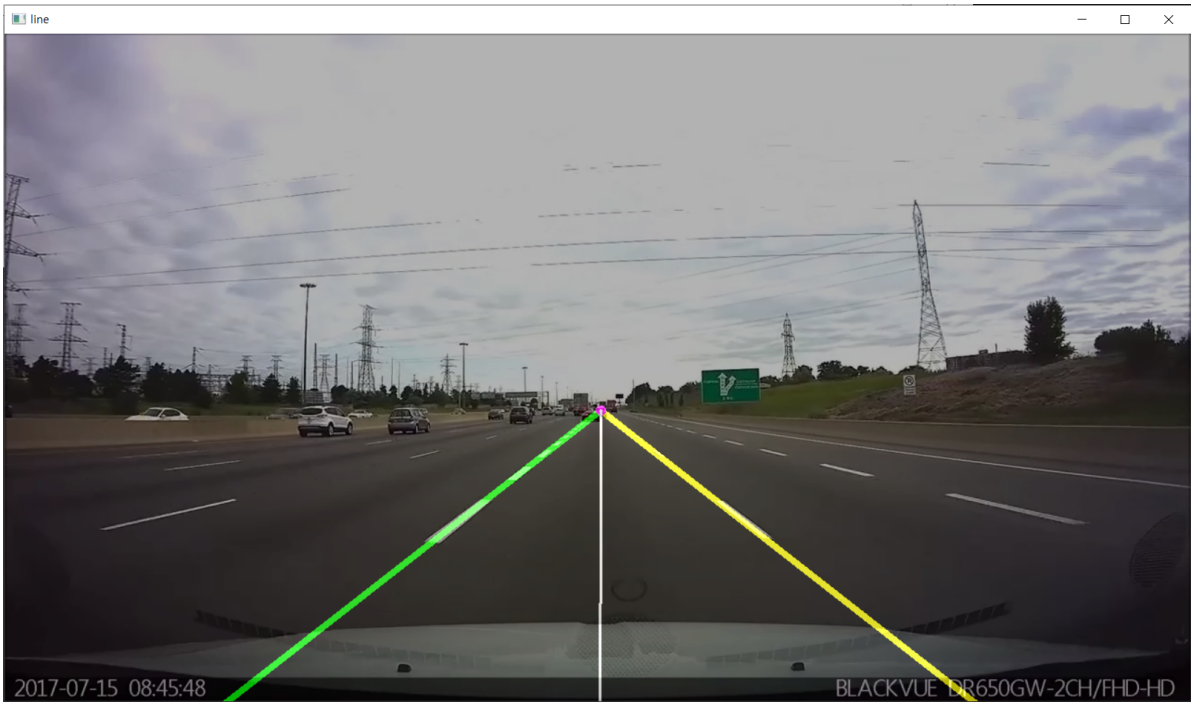


Figure 6. Draw the Middle Line

Measure the angle between the two lines measuring the lane and set it to come to the middle. Treat the middle line so that its slope does not diverge as well.

4.7 Change lane

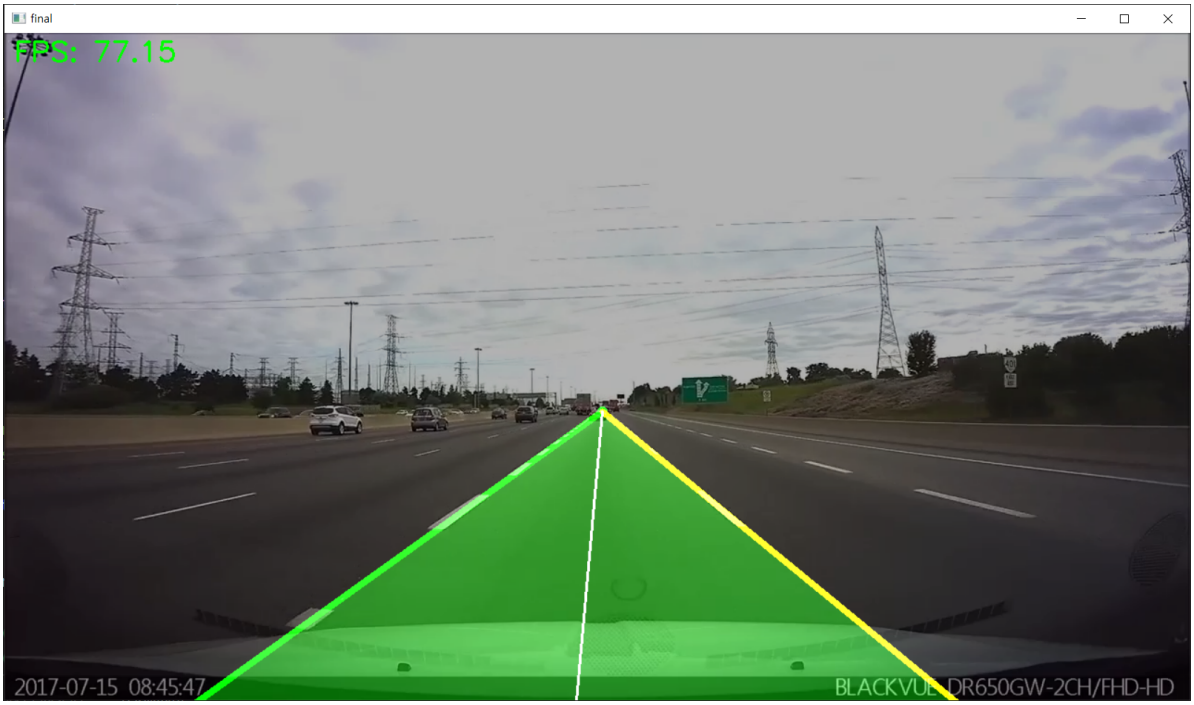


Figure 7. Going Straight

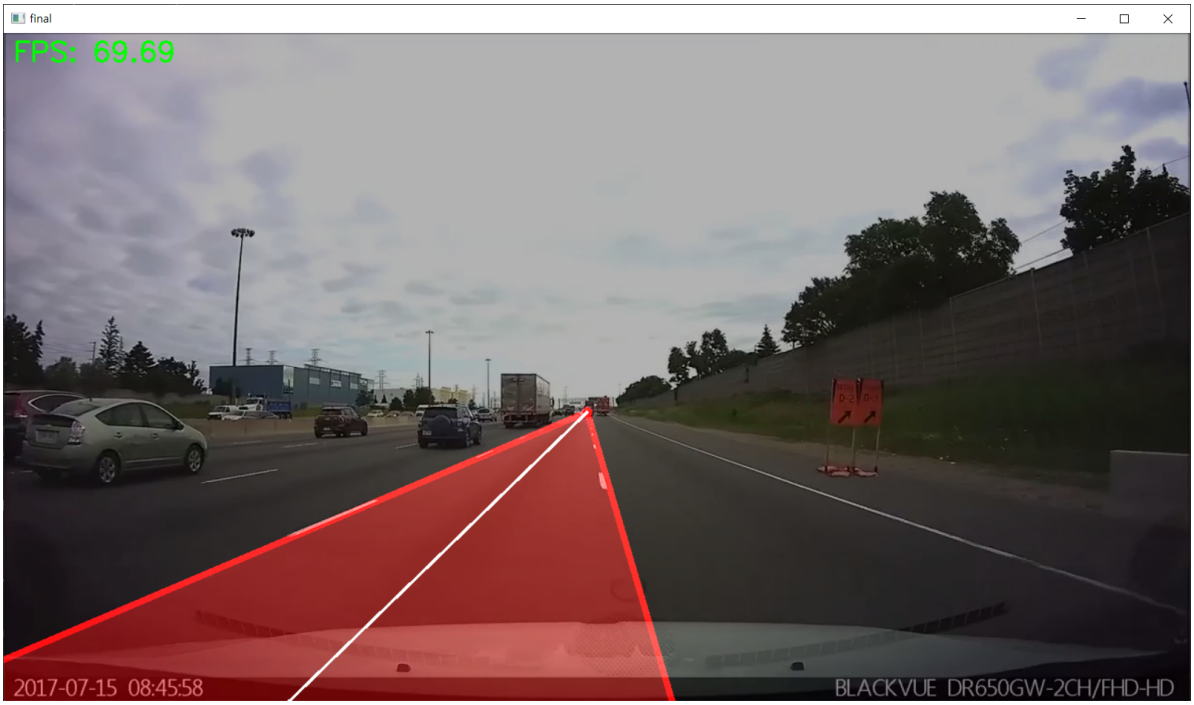


Figure 8. Lane Change to Right

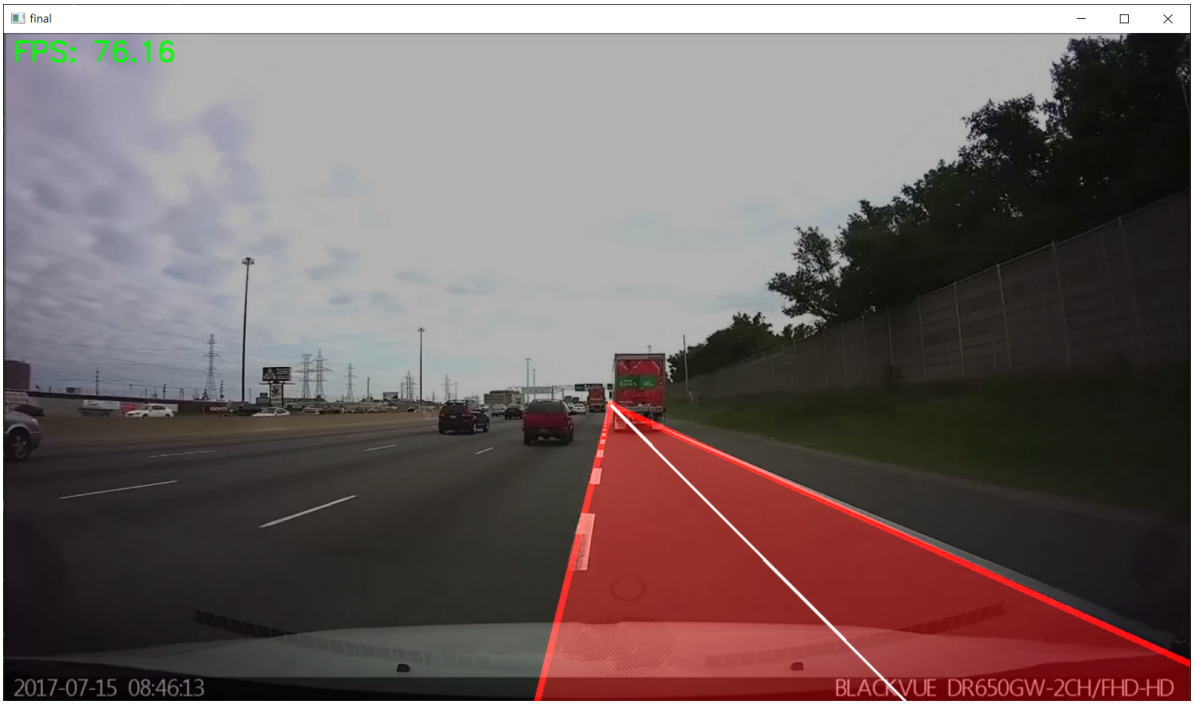


Figure 8. Lane Change to Left

During straight driving, the road color inside the line is filled with green according to procedure 4.6. However, when changing lanes, the road and line colors are painted red. Lane changes are detected based on the middle line by measuring the angles of both lines. When the difference between the two angles increases, it is recognized as a lane change.

4.8 Put Warning Message

Figure 9. Warning Message

```
class Linedetect:
    def getLines(self, _frame):
        self.lines = cv.HoughLines(self.roi, self.hough_rho, self.hough_theta,
self.hough_thr)

        # 넘파이는 height 다음 width  입니다!!!
        self.line_img = np.zeros((self.height, self.width, 3), dtype=np.uint8)
        self.lines_pos = []
        self.lines_neg = []
        if self.lines is not None and len(self.lines) > 0:
            for i in range(len(self.lines)):
                for rho, theta in self.lines[i]:

                    self.slope = np.tan(theta)
                    if(self.slope > 0):
                        self.lines_pos.append((rho, theta))
                    else:
                        self.lines_neg.append((rho, theta))

            if len(self.lines_pos) > 0:
                self.avg_rho_pos, self.avg_theta_pos = np.mean(self.lines_pos,
axis = 0)
                self.prev_line_pos = (self.avg_rho_pos, self.avg_theta_pos)
                self.color_pos = color_g

            elif self.prev_line_pos is not None:
                self.avg_rho_pos, self.avg_theta_pos = self.prev_line_pos
                self.color_pos = color_y

            if len(self.lines_neg) > 0:
                self.avg_rho_neg, self.avg_theta_neg = np.mean(self.lines_neg,
axis = 0)
                self.prev_line_neg = (self.avg_rho_neg, self.avg_theta_neg)
                self.color_neg = color_g
```

```
            elif self.prev_line_neg is not None:
                self.avg_rho_neg, self.avg_theta_neg = self.prev_line_neg
                self.color_neg = color_y

        self.intersection = self.Intersection(self.prev_line_pos,
self.prev_line_neg)
        self.line_point_pos, self.line_point_neg, self.line_point_cen =
self.drawLines(self.prev_line_pos, self.prev_line_neg)

        if self.intersection is not None:

            cv.circle(self.line_img, self.intersection, 3, self.mask_road, 4)
            cv.line(self.line_img, self.intersection, self.line_point_pos,
self.color_pos, 5)
            cv.line(self.line_img, self.intersection, self.line_point_neg,
self.color_neg, 5)
            cv.line(self.line_img, self.intersection, self.line_point_cen,
color_w, 2)

            mask = np.zeros_like(_frame)
            pts = np.array([self.intersection, self.line_point_pos,
self.line_point_neg], dtype=np.int32)
            cv.fillConvexPoly(mask, pts, self.mask_road)

            self.result = cv.addWeighted(_frame, 0.7, mask, 0.4, 0)
            self.result = cv.addWeighted(self.result, 1, self.line_img, 1, 0)

            # 차선 이동 여부 출력
            cv.putText(self.result, self.line_dir, (int(self.width/5),
int(self.height/2)), cv.FONT_HERSHEY_SIMPLEX, 3, (255, 0, 255), 5)
            diff_deg = f"Angle Diffrent: {(self.diff_deg):.2f}"
            cv.putText(self.result, diff_deg, (10, 60), cv.FONT_HERSHEY_SIMPLEX,
1, (0, 255, 0), 2)

    def Intersection(self, line_pos, line_neg):

        rho1, theta1 = line_pos
        rho2, theta2 = line_neg

        A = np.array([
            [np.cos(theta1), np.sin(theta1)],
            [np.cos(theta2), np.sin(theta2)]
        ])
        b = np.array([[rho1], [rho2]])
        x0, y0 = np.linalg.solve(A, b)
        x0, y0 = int(np.round(x0)), int(np.round(y0))

        return [x0, y0]

    def drawLines(self, line_pos, line_neg):
        extended_y = self.height + 100

        rho1, theta1 = line_pos
        m1 = -np.cos(theta1) / np.sin(theta1)
        b1 = rho1 / np.sin(theta1)

        if not np.isinf(m1) and not np.isnan(m1):
            x1 = int((extended_y - b1) / m1)
```

```
            y1 = extended_y
            self.prev_x1 = x1
            self.prev_y1 = y1
        else:
            x1 = self.prev_x1
            y1 = self.prev_y1

        self.slope_pos = m1

        rho2, theta2 = line_neg
        m2 = -np.cos(theta2) / np.sin(theta2)
        b2 = rho2 / np.sin(theta2)

        if not np.isinf(m2) and not np.isnan(m2):
            x2 = int((extended_y - b2) / m2)
            y2 = extended_y
            self.prev_x2 = x2
            self.prev_y2 = y2
        else:
            x2 = self.prev_x2
            y2 = self.prev_y2

        self.slope_neg = m2

        xm, ym = self.getBias(x1, x2, extended_y)

        return (x1, y1), (x2, y2), (xm, ym)

    def getBias(self, x1, x2, extended_y):

        x0, y0 = self.intersection
        xm = int((x1+ x2) / 2)
        ym = extended_y

        if x0 != xm:
            self.slope_mid = -(y0 - ym) /(x0 - xm)
            self.prev_slope_mid = self.slope_mid
        else:
            self.slope_mid = self.prev_slope_mid

        self.theta_mid      = np.degrees(np.arctan(self.slope_mid))

        if(self.theta_mid < 0):
            self.theta_mid = -self.theta_mid
        elif(self.theta_mid > 0):
            self.theta_mid = 180 - self.theta_mid

        self.diff_deg_pos = -np.degrees(np.arctan(self.slope_pos)) + 90 -
self.theta_mid
        self.diff_deg_neg = self.theta_mid -
np.degrees(np.arctan(self.slope_neg))

        self.diff_deg = self.diff_deg_pos - self.diff_deg_neg

        # print('diff_deg: ' + str(self.diff_deg))

        if self.diff_deg > 75 :
            self.line_dir = 'WARNING!! Go Left'
```

```
            self.mask_road = color_r
            self.color_pos = color_r
            self.color_neg = color_r

        elif self.diff_deg < - 75:
            self.line_dir = 'WARNING!! Go Right'
            self.mask_road = color_r
            self.color_pos = color_r
            self.color_neg = color_r

        else:
            self.line_dir = ''
            if len(self.lines_pos) > 0:
                self.color_pos = color_g

            elif self.prev_line_pos is not None:
                self.color_pos = color_y

            if len(self.lines_neg) > 0:
                self.color_neg = color_g

            elif self.prev_line_neg is not None:
                self.color_neg = color_y

            self.mask_road = color_g

        return (xm, ym)

def main():


    while(cap.isOpened()):

    # procedure 4.1
    # procedure 4.2
    # procedure 4.2

        LD.getLines(frame)

        LD.FPS()

        LD.showVid("final", LD.result)

        LD.outResult()

        k = cv.waitKey(1) & 0xFF
        if k == ord('q')   :   break
        elif k == ord('s') :   cv.waitKey()

    LD.closeAll()
```

Result

result video

Conclusion

In the present investigation, an advanced lane departure warning system has been meticulously designed and implemented for the accurate detection of road lanes and prompt notification of the driver upon unintended lane deviation. The proposed system is built upon the integration of OpenCV for sophisticated image processing tasks, the HoughLines algorithm for precise line detection, and NumPy for performing essential mathematical operations.

A notable challenge encountered in this study was the intricate process of converting angles between the Hough coordinate system and the Cartesian coordinate system. This task necessitated a considerable amount of experimentation and iterative refinement to achieve the desired level of accuracy. The successful calculation of the normal route change was contingent upon the unification of these two coordinate systems.

The results are rendered in real-time, with a calculated frame rate, thereby ensuring an efficient and reliable solution for the improvement of vehicular safety on roadways.