프로그래머스 자율주행 데브코스 3기

2022.04.08. 차선인식 경진대회

Indigo

Team A1-1

Code Review

김대영 · 강수민 · 한은기

2022.04.13.수

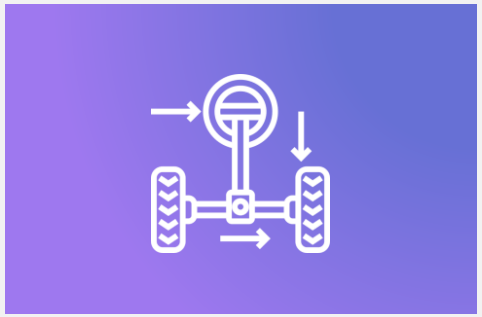
Contents

01 Strategy

Goals

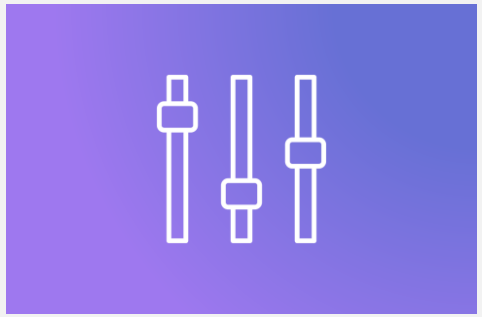
차선 인식
Lane Detection

- 조도의 변화
- 차선 인식 장애물 처리
- 왼·오른 차선 구분
- 기타 예외 상황 대응



모터 제어
Motor Control

- 차선 인식 결과 필터링
- 조향각 PID 제어
- 구간별 속도 조절
- 전·후진 기능 구현



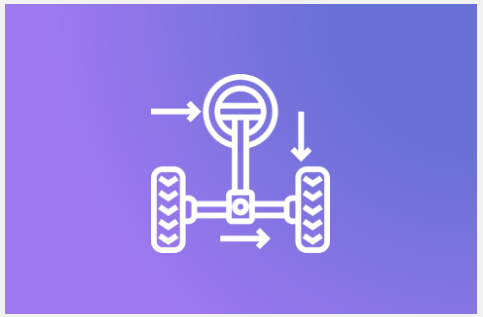
효율적 튜닝
Efficient Tuning

- 영상 처리 파라미터
- PID 계수
- 모터 속도

Strategies

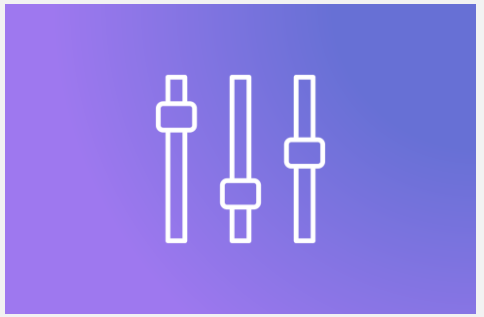
차선 인식
Lane Detection

- 히스토그램 정규화
  Histogram Normalization
- 캐니 에지 검출기
  Canny Edge Detection
- 허프변환 알고리즘
  Hough Transformation



모터 제어
Motor Control

- PID 제어 PID Control
  & 이동평균필터 MAF
- 곡선 구간 속도 감소
  Decelerate in Curve
- 후진 기능
  Reverse Function



효율적 튜닝
Efficient Tuning

- 트랙바 사용
  Using Trackbar

02

Code Review

연산 속도 향상을 위해

경기용 코드와 튜닝용 코드를 따로 관리하였습니다.

cv2.imshow(…), Trackbar 제거 등

보여드리는 내용은 더 나은 이해를 위해

둘을 조합하고 약간의 수정을 거친 코드임을 알려드립니다!

Codes: Import Modules, main


```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import numpy as np
import cv2, math, rospy
from cv_bridge import CvBridge
from sensor_msgs.msg import Image
from xycar_msgs.msg import xycar_motor
from std_msgs.msg import String
```

```python
if __name__ == '__main__':
    a = Houghline_Detect()
    a.start()
```

사용 클래스

PID
PID 제어

MAF
이동평균필터

Houghline_Detect
영상처리 및 차선 검출, 모터 조향

Codes: PID Class


```python
class PID():
    def __init__(self, kp=0.45, ki=0.0007, kd=0.15):
        self.Kp = kp
        self.Ki = ki
        self.Kd = kd
        self.p_error = 0.0
        self.i_error = 0.0
        self.d_error = 0.0

    def set(self,kp,ki,kd):
        self.Kp = kp
        self.Ki = ki
        self.Kd = kd
        self.p_error = 0.0
        self.i_error = 0.0
        self.d_error = 0.0

    def pid_control(self, cte):
        self.d_error = cte - self.p_error
        self.p_error = cte
        self.i_error += cte
        if abs(self.i_error) > 100000:
            self.i_error = 0

        return self.Kp * self.p_error + self.Ki * self.i_error + self.Kd * self.d_error
```

트랙바 설정 시
해당 값들을 변경 및 초기화

CTE 값을 입력받아
PID 제어값 결정

Codes: M.A.F Class


```python
class MAF:
    def __init__(self,n):
        self.n = n
        self.data = [0] * self.n

    def add_data(self, new_data):
        if len(self.data) < self.n:
            self.data.append(new_data)
        else:
            self.data = self.data[1:] + [new_data]

    def get_data(self):
        return float(sum(self.data))/len(self.data)
```

큐에 자리 남았으면 뒤에 추가
큐가 다 찼으면 하나를 지우고 추가

큐 내부 값을 평균해 돌려줌

Codes: Houghline_Detect


```
class Houghline_Detect:
    def __init__(self):
        self.m_a_f = MAF(5)
        self.pid=PID()

        self.image = np.empty(shape=[0])
        self.bridge = CvBridge()
        self.pub = None

        self.Width, self.Height = 640, 480   # raw image의 가로와 세로 픽셀 크기
        self.Offset = 380   # ROI를 설정할 세로 위치
        self.Gap = 55       # ROI의 높이
```

이동평균필터 클래스 MAF(큐 크기 5)와
PID 클래스 PID 호출

Codes: Houghline_Detect


```
class Houghline_Detect:
    def __init__(self):

        ...


        self.trackbar()
        self.pallete_title = 'pallete'
        #self.speed = 30
        #self.pid_p = 0.35
        #self.pid_i = 0.0005
        #self.pid_d = 0.005

        self.angle = 0.0  # 조향각
        self.count = 0    # 직선 미검출 횟수를 세는 변수

        self.prev_speed = 0  # 이전 속도(publish 기준)
        self.prev_rpos = 0   # 이전 오른 차선 위치
        self.prev_lpos = 0   # 이전 왼 차선 위치
        self.PURPOSE_SPEED = self.speed  # 기본 속도
```

트랙바 생성 함수와 그 이름 선언
실전에서는 직접 상수로 대체

Codes: Houghline_Detect


```
def img_callback(self, data):
    self.image = self.bridge.imgmsg_to_cv2(data, "bgr8")


def nothing(self, x):
    pass

def change_offset(self, x):
    self.Offset = x

def change_P(self, x):
    self.pid_p = float(x / 100)
    self.pid.set(self.pid_p,self.pid_i,self.pid_d)
def change_I(self, x):
    self.pid_i = float(x / 10000)
    self.pid.set(self.pid_p, self.pid_i, self.pid_d)
def change_D(self, x):
    self.pid_d = float(x / 1000)
    self.pid.set(self.pid_p, self.pid_i, self.pid_d)
```

Trackbar 변경 Callback 함수
(getTrackbarPos를 사용하므로 아무 작업 없음)

Offset 변수의 Trackbar Callback 함수

P, I, D 계수 Trackbar Callback 함수

Codes: Houghline_Detect

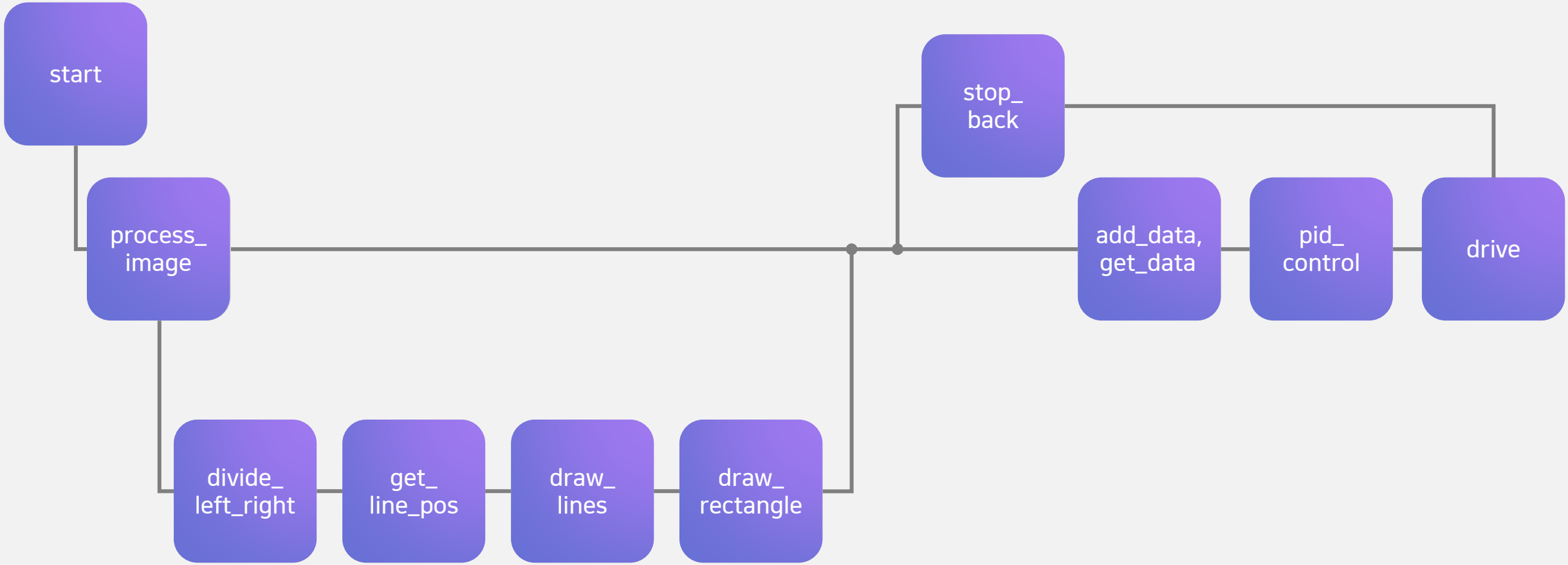

```
def trackbar(self):
    cv2.namedWindow("pallete")
    cv2.createTrackbar('Offset', "pallete", 380, 400, self.change_offset)
    cv2.createTrackbar('hough_threshold', "pallete", 30, 180, self.nothing)
    cv2.createTrackbar('hough_minLineLength ', "pallete", 0, 255, self.nothing)
    cv2.setTrackbarPos('hough_minLineLength ', "pallete", 30)
    cv2.createTrackbar('hough_maxLineGap', "pallete", 0, 255, self.nothing)
    cv2.setTrackbarPos('hough_maxLineGap', "pallete", 10)
    cv2.createTrackbar('canny_thr_min', "pallete", 0, 255, self.nothing)
    cv2.setTrackbarPos('canny_thr_min', "pallete", 60)
    cv2.createTrackbar('canny_thr_max', "pallete", 0, 180, self.nothing)
    cv2.setTrackbarPos('canny_thr_max', "pallete", 70)
    cv2.createTrackbar('PID_P', "pallete", 0, 99, self.change_P)
    cv2.setTrackbarPos('PID_P', "pallete", 45)
    cv2.createTrackbar('PID_I', "pallete", 0, 10, self.change_I)
    cv2.setTrackbarPos('PID_I', "pallete", 7)
    cv2.createTrackbar('PID_D', "pallete", 0, 100, self.change_D)
    cv2.createTrackbar('speed', "pallete", 0, 50, self.nothing)
    cv2.createTrackbar('middle_thresh', "pallete", 0, 90, self.nothing)
```

HoughLinesP(…) 함수 파라미터

Canny(…) 함수 파라미터

직선 필터링에 쓰이는 상수

Codes: Houghline_Detect



Codes: Houghline_Detect


```
def start(self):
    rospy.init_node('auto_drive')
    self.pub = rospy.Publisher("xycar_motor", xycar_motor, queue_size=1)
    rospy.Subscriber("/usb_cam/image_raw/", Image, self.img_callback)
    print("---------- We just go! ----------")

    for i in range(5):
        self.drive(0,0)

    while True:
        while not self.image.size == (640 * 480 * 3):
            continue
```

시작 전 바퀴 똑바로 정렬

Codes: Houghline_Detect


```
def start(self):

    ...
        왼/오른쪽 차선 위치(pixel)을 받아옴
        lpos, rpos = self.process_image(self.image)
        if lpos == -1 and rpos != self.Width + 1:
            lpos = rpos - 450
        elif rpos == self.Width + 1 and lpos != -1:
            rpos = lpos + 450
        elif rpos == self.Width + 1 and lpos == -1:
            self.count += 1
            if self.count >= 13:
                self.stop_back()
            else:
                self.drive(self.angle,self.speed)
            continue
        else:
            if rpos - lpos < 440:
                if abs(rpos-self.prev_rpos) < 100:
                    lpos = rpos - 450
                if abs(lpos-self.prev_lpos) < 100:
                    rpos = lpos + 450
```

한 쪽 차선만 검출 시, 반대쪽 차선은
450 pixel 만큼 떨어진 곳에 위치했다고 설정

Codes: Houghline_Detect


```
def start(self):

    ...


        lpos, rpos = self.process_image(self.image)
        if lpos == -1 and rpos != self.Width + 1:
            lpos = rpos - 450
        elif rpos == self.Width + 1 and lpos != -1:
            rpos = lpos + 450
        elif rpos == self.Width + 1 and lpos == -1:
            self.count += 1
            if self.count >= 13:
                self.stop_back()
            else:
                self.drive(self.angle,self.speed)
            continue
        else:
            if rpos - lpos < 440:
                if abs(rpos-self.prev_rpos) < 100:
                    lpos = rpos - 450
                if abs(lpos-self.prev_lpos) < 100:
                    rpos = lpos + 450
```

집념으로 찾은 값… 피땀눈물…

양쪽 차선 전부 미검출 시,
미검출이 13회 이상이면 후진,
아니면 그대로 속도와 조향각 유지한 채 전진

이동평균필터, PID 계산, 값 저장 없이 바로 Publish

Codes: Houghline_Detect

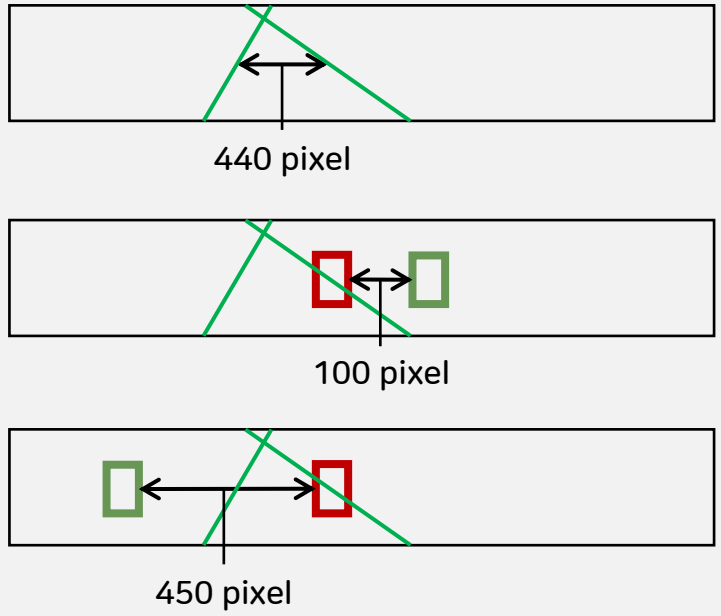

```
def start(self):

    ...


        lpos, rpos = self.process_image(self.image)
        if lpos == -1 and rpos != self.Width + 1:
            lpos = rpos - 450
        elif rpos == self.Width + 1 and lpos != -1:
            rpos = lpos + 450
        elif rpos == self.Width + 1 and lpos == -1:
            self.count += 1
            if self.count >= 13:
                self.stop_back()
            else:
                self.drive(self.angle,self.speed)
            continue
        else:
            if rpos - lpos < 440:
                if abs(rpos-self.prev_rpos) < 100:
                    lpos = rpos - 450
                if abs(lpos-self.prev_lpos) < 100:
                    rpos = lpos + 450
```

도로폭 최솟값인 450 pixel보다 약간 작은 수치

양쪽 차선 전부 검출 시에
양 차선의 위치 차가 440 pixel 이하로 꽤 좁다면, 직전 위치와 비교

100 pixel 이하의 차이라면, 왼/오른차선 ±450 pixel (도로폭)

➡ 도로 내부에서 차선 오검출을 일으키는 요소(콘센트 등) 제거
(+ finish 이후 합쳐지는 차선에서 알아서 차선 변경)



Codes: Houghline_Detect


```
def start(self):

    ...

        self.count = 0  # 차선 미검출 누적 횟수 초기화
        center = (lpos + rpos) / 2    # 양 차선의 평균을 중앙점으로 설정
        error = (center - (self.Width / 2-5))*0.8

        self.m_a_f.add_data(error)
        avg_angle = self.m_a_f.get_data()
        self.angle = self.pid.pid_control(avg_angle)

        self.drive(self.angle , self.speed) # 에러각과 속도를 바탕으로 모터로 publish

        self.prev_rpos = rpos
        self.prev_lpos = lpos
```

중앙과 틀어진 pixel 만큼 error_angle로 가정
이동평균필터, PID 적용해 최종 조향각(모터) 결정

후진이 없을 때의 왼쪽/오른쪽 차선 위치 저장

[중앙점 – {(도로 너비 ÷ 2) – 치우친 픽셀 }] × 예민성 완화

```
error = (error - error_min) * (steering_max - steering_min) / (error_max - error_min) + steering_min
```

```python
def process_image(self, frame):
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    kernel_size = 5
    norm = cv2.normalize(gray,None,0,255,cv2.NORM_MINMAX)
    blur_gray = cv2.GaussianBlur(norm, (kernel_size, kernel_size), 0)

    low_threshold = cv2.getTrackbarPos("canny_thr_min", self.pallete_title)
    high_threshold = cv2.getTrackbarPos("canny_thr_max", self.pallete_title)
    edge_img = cv2.Canny(np.uint8(blur_gray), low_threshold, high_threshold)

    threshold = cv2.getTrackbarPos("hough_threshold", self.pallete_title)
    minLineLength = cv2.getTrackbarPos("hough_minLineLength ", self.pallete_title)
    maxLineGap = cv2.getTrackbarPos("hough_maxLineGap", self.pallete_title)
    all_lines = cv2.HoughLinesP(roi, 1, math.pi / 180, \
                                threshold, minLineLength, maxLineGap)
```

Trackbar

Gray scale 변환

▼

히스토그램 정규화

▼

가우시안 필터

▼

Canny 에지 검출기

▼

Hough 라인 변환

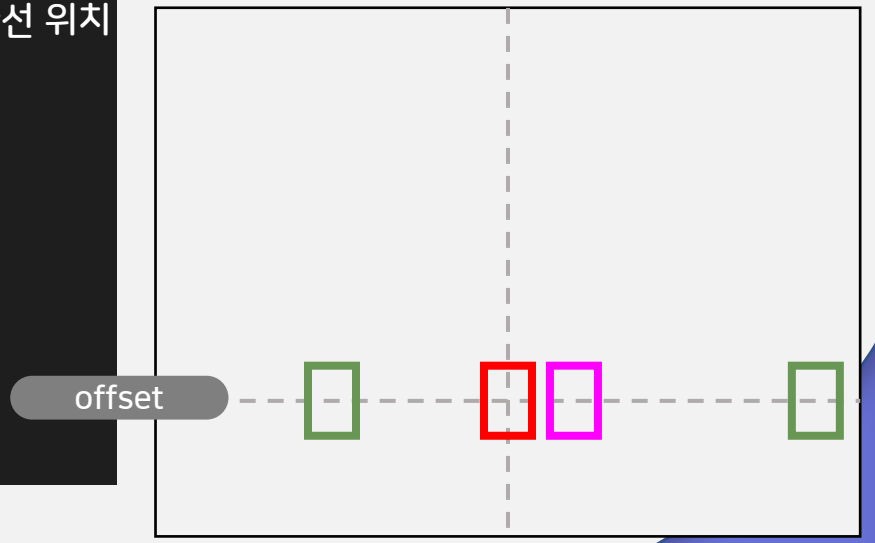
```python
def process_image(self, frame):

    ...

    if all_lines is None:
        return 0, 640
    left_lines, right_lines = self.divide_left_right(all_lines)

    frame, lpos = self.get_line_pos(frame, left_lines, left=True)
    frame, rpos = self.get_line_pos(frame, right_lines, right=True)

    frame = self.draw_lines(frame, left_lines)
    frame = self.draw_lines(frame, right_lines)

    frame = self.draw_rectangle(frame, lpos, rpos, offset=self.Offset)

    cv2.imshow('calibration', frame)
    cv2.imshow("canny", edge_img)

    return lpos, rpos
```

직선 미검출 시, 왼쪽을 0 & 오른쪽을 640

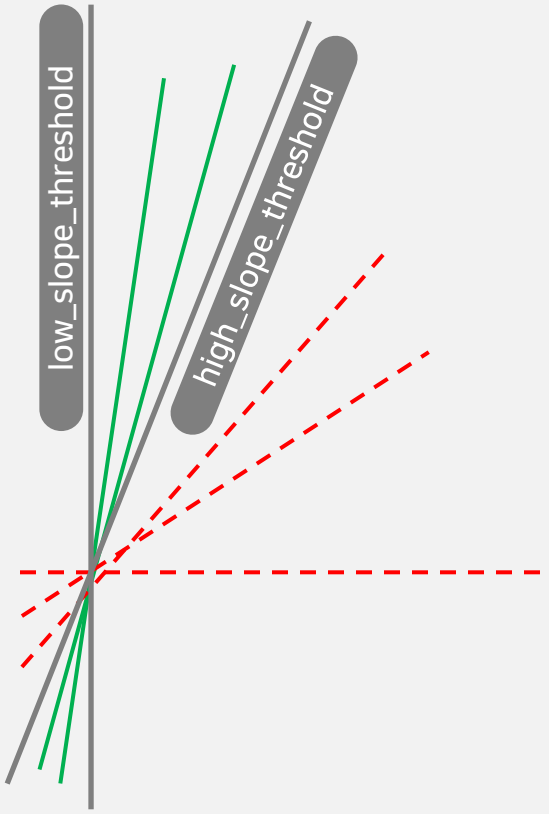
➡ 왼/오른차선 구분

➡ 대표 직선, 차선 위치

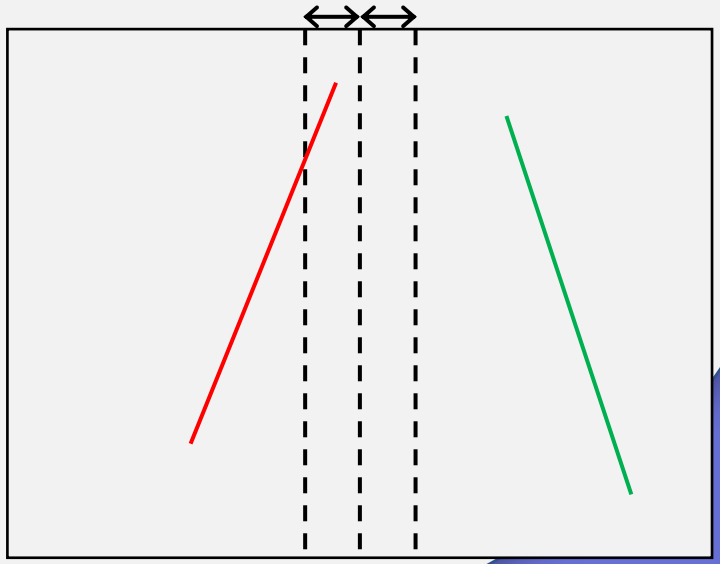
➡ 좌우 차선 그리기



```python
def divide_left_right(self, lines):
    low_slope_threshold = 0    # 직선의 최소 기울기
    high_slope_threshold = 10  # 직선의 최대 기울기

    slopes = []    # 기울기 조건에 해당하는 기울기 모음
    new_lines = [] # 기울기 조건에 해당하는 선(x1, y1, x2, y2) 모음

    for line in lines:
        x1, y1, x2, y2 = line[0]

        if x2 - x1 == 0:
            slope = 0 # x 값이 같다면 기울기를 0으로 함. DividedByZero 오류 방지
        else:
            slope = float(y2 - y1) / float(x2 - x1) # 기울기 계산

        if abs(slope) > low_slope_threshold and abs(slope) < high_slope_threshold:
            slopes.append(slope)
            new_lines.append(line[0])
```



```
def divide_left_right(self, lines):

    ...

    left_lines = []      # 왼쪽 차선들을 저장할 리스트
    right_lines = []     # 오른쪽 차선들을 저장할 리스트
    middle_thresh = 10   # 화면의 왼/오른쪽에 있는지 판단할 기준

    for j in range(len(slopes)):
        Line = new_lines[j]
        slope = slopes[j]

        x1, y1, x2, y2 = Line

        if (slope < 0) and (x2 < self.Width / 2 - middle_thresh):
            left_lines.append([Line.tolist()])
        elif (slope > 0) and (x1 > self.Width / 2 + middle_thresh):
            right_lines.append([Line.tolist()])

    return left_lines, right_lines
```



```
def get_line_pos(self, img, lines, left=False, right=False):
    m, b = self.get_line_params(lines)
    if m == 0 and b == 0:
        if left:
            pos = -1
        if right:
            pos = self.Width + 1
    else:
        y = self.Gap / 2
        pos = (y - b) / m
        b += self.Offset
        x1 = (self.Height - b) / float(m)
        x2 = ((self.Height / 2) - b) / float(m)
        cv2.line(img, (int(x1), self.Height), \
                 (int(x2), (self.Height / 2)), (255, 0, 0), 3)

    return img, int(pos)
```

➡ 모든 직선들을 대표하는 기울기와 절편을 리턴

수평선 처리

ROI 중간 지점에서 직선과의 교점을 차선 위치로 계산

ROI 내부에서의 직선을
확장하여 화면에 그림

```python
def get_line_params(self, lines):
    x_sum = 0.0
    y_sum = 0.0
    m_sum = 0.0

    size = len(lines)
    if size == 0:
        return 0, 0

    for line in lines:
        x1, y1, x2, y2 = line[0]
        x_sum += x1 + x2
        y_sum += y1 + y2
        m_sum += float(y2 - y1) / float(x2 - x1)

    x_avg = x_sum / (size * 2)
    y_avg = y_sum / (size * 2)
    m = m_sum / size
    b = y_avg - m * x_avg

    return m, b
```

모든 직선의 x, y, 기울기를 평균하고
직선의 절편을 구함

```
def drive(self, Angle, Speed):
    msg = xycar_motor()

    if Angle >= 50:
        Angle = 50
    if Angle <= -50:
        Angle = -50
    msg.angle = Angle

    if Speed != -10:
        if self.speed < self.PURPOSE_SPEED:
            self.speed += 0.75
            Speed = self.speed
        else:
            self.speed = self.PURPOSE_SPEED

    if abs(msg.angle) > 20 and Speed != -10:
        msg.speed = (Speed -(0.17 * abs(Angle)))
    else:
        msg.speed = (Speed -(0.1 * abs(Angle)))

    self.pub.publish(msg)
```

차선의 위치를 통해 구한 조향각을
모터 최대/최소 조향각 범위 내로 조정

후진이 아닐 때는
목표 속도까지 천천히 속도를 올림

→ 후진 후 급발진을 방지하여
차선 재이탈 위험을 줄이기 위함

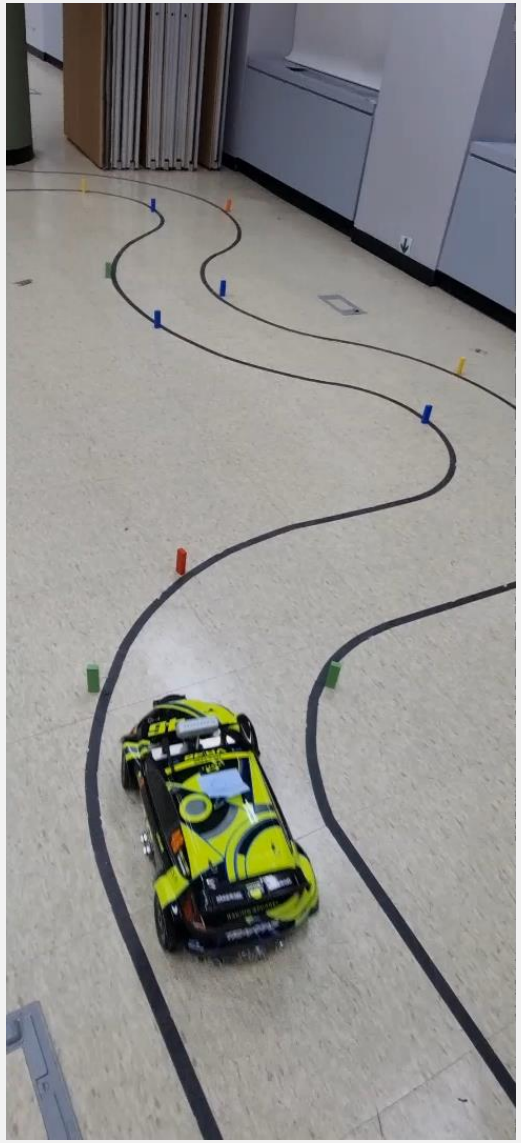
회전 구간에서 각도와 비례해 감속

```python
def stop_back(self):
    print("--------------stop---------------")
    rate = rospy.Rate(10)

    while True:
        if self.speed>=0:
            self.speed -= 5
            self.drive(self.angle,self.speed)
        else:
            break

    self.speed = -10 # 후진 속도
    for i in range(4):
        self.drive(-self.angle/4, self.speed)
        rate.sleep()

    self.speed = 1    # 속도를 양수로 만들어 drive 조건식에 이용
```

속도가 0이 될 때까지 서서히 감속

후진 속도로 변경 후 0.4초간 후진,
진행하던 반대 방향으로 후진

구현 장면

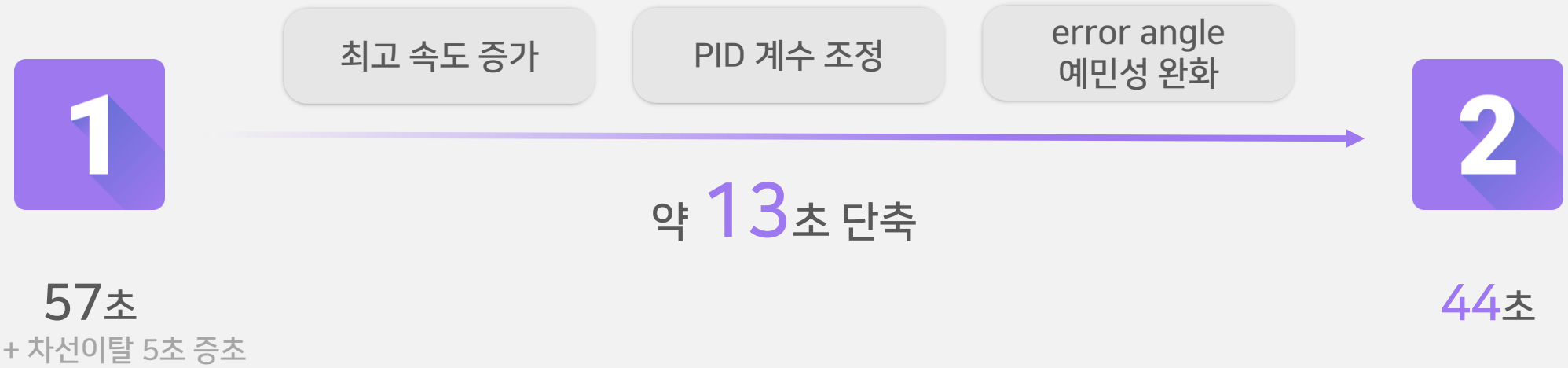


후진 작동 영상

차선 오검출 제거

03 Review

대회 결과


1
최고 속도 증가
PID 계수 조정
error angle
예민성 완화
2
약 13초 단축
57초
52초 + 차선이탈 5초 증초
44초

회고

영상처리

- **대영** Python으로 개발했기 때문에 ROI 설정 시 리스트 슬라이싱으로 구현이 가능했다.
  만약 C++을 썼다면 ROI 설정하고 Canny 에지 검출기를 적용하여 자른 부분에 가로선이 생겼다.

- **수민** 배웠던 다른 OpenCV 기능들을 다 활용해보지 못했다.

- **은기** 팀 합류 전 Birds-eye View에 슬라이딩 윈도우나 허프 직선 변환을 사용한 두 버전을 개발했었는데,
  제대로 된 수정을 해볼 기회가 없었고, 이를 제대로 활용해보지 못했다.

- **은기** 왼쪽 차선과 오른쪽 차선 여러 개가 검출되었을 때 하나의 대표 직선으로 추리는 함수
  힘수 get_line_pos(…)는 OpenCV 제공 함수인 cv2.fitLine(…)으로 대체될 수 있었다.
  합류했을 때는 이미 대부분의 함수 구조가 잡힌 터였고, 함부로 대체하였다가 복구하는 시간이 오래 걸릴
  수 있어 시도하진 않았다.

회고

영상처리

```
else:
    if rpos - lpos < 440:
        if abs(rpos-self.prev_rpos) < 100:
            lpos = rpos - 450
        if abs(lpos-self.prev_lpos) < 100:
            rpos = lpos + 450
```

- **대영** 이전 위치와 현재 위치를 비교하는 기준 픽셀값을 100으로 임의 설정한 부분이 아쉬웠다.
  위치의 변화량을 계산하여 다음 위치를 추정하거나, 오차 범위를 유연하게 특정하는 것이 좋았을 것 같다.

  rpos(오른 차선 위치)를 먼저 비교하는데, 위처럼 구현하면 오른쪽에서 왼쪽으로 차선이 합쳐질 때는 잘 작동하나, 왼쪽에서 오른쪽으로 합류해야 할 때는 오작동을 일으킨다. 구현이 잘못되었다.

회고

영상처리

```
else:
    if rpos - lpos < 440:
        if abs(rpos-self.prev_rpos) < 100 and abs(lpos-self.prev_lpos) >= 100:
            lpos = rpos - 450
        elif abs(lpos-self.prev_lpos) < 100 and abs(rpos-self.prev_rpos) >= 100:
            rpos = lpos + 450
        elif abs(rpos-self.prev_rpos) < 100 and abs(lpos-self.prev_lpos) < 100:
            # 차선이 좁아졌거나 합쳐지기 시작하는 구간
        else:
            # 차선을 잃고 다른 곳을 트래킹하는 경우, 차선 찾는 회복 기능
```

- **대영** 이전 코드 대신 위와 같은 식으로 진행되어야 맞을 것 같다.

회고

제어

- **대영** 특히 함수 drive(…) 에서 가·감속 계수들을 임의로 설정하였는데, 물리학이나 수학을 기반으로 구현하지 못한 것에 큰 아쉬움이 남았다.

- **대영** PID 계수 중 P 계수가 0.45 근방이면 직선 구간에서 흔들림이 적고, 0.35 근방이면 곡선 구간에서 이탈 없이 안정적이었다. 서로 Trade-off 관계였다. PID 제어를 하나만 사용해야 한다는 생각에 갇혔던 것 같다.

- **대영** 모터 속도 제한을 10,000으로 상향조정한 뒤, 후진 명령을 내리면 역토크가 발생해 모터에 부하가 갔다. 이에 대응하려 감속 후 후진을 하도록 구현했다.

- **은기** 속도에도 PID 제어를 도입했다면, 목표 속도까지 천천히 가속하고 유지하는 데 유용했을 것 같다. 막판에 시도는 해보았으나, 필요한 것은 천천히 가속을 하는 것보다 천천히 감속을 하는 것이었다. 따라서 시간의 제약으로 PID 제어를 상황에 맞게 변형하지 못한 아쉬움이 남는다.

회고

제어

- 수민 Trackbar를 이용해 launch 를 끄지 않고도 파라미터 등을 수정하며 변화 양상을 살펴볼 수 있었다. 따라서 튜닝의 시간과 번거로움을 줄일 수 있었다.
- 은기 마지막에는 Trackbar를 제거하고 상수로 값을 직접 할당했는데, 이 때는 일일이 코드 내 변수의 위치를 찾아가며 값을 조정했다. yaml 파일이나 *.launch 파일의 private 변수로 처리했다면 간편했을 것이다.
- 은기 초반 직선 구간에서 좌우로 크게 동요하는데, 이 현상을 줄이면 랩타임을 줄일 수 있을 듯해 아쉽다.
- 대영 차선을 찾지 못했을 때 트랙을 이탈했는지 여부를 판단하기 위한 수학적 방법이 있는지 알고 싶다.

회고

팀 활동

- **수민** 2회차 직전 정비 시간 중 속도를 올려 파라미터 재조정을 짧게 테스트했다. 자칫 무모한 도전일 수 있었으나, 랩타임 2위를 달성하는 데 성공했다.

- **수민** 파라미터 값과 설정값들을(PID 계수, 속도, MAF 큐 사이즈 등)의 변화와 그때의 랩타임을 기록해가며 튜닝을 진행하여, 최적의 값을 찾고 양상을 파악하기에 용이했다. 노션 만만세

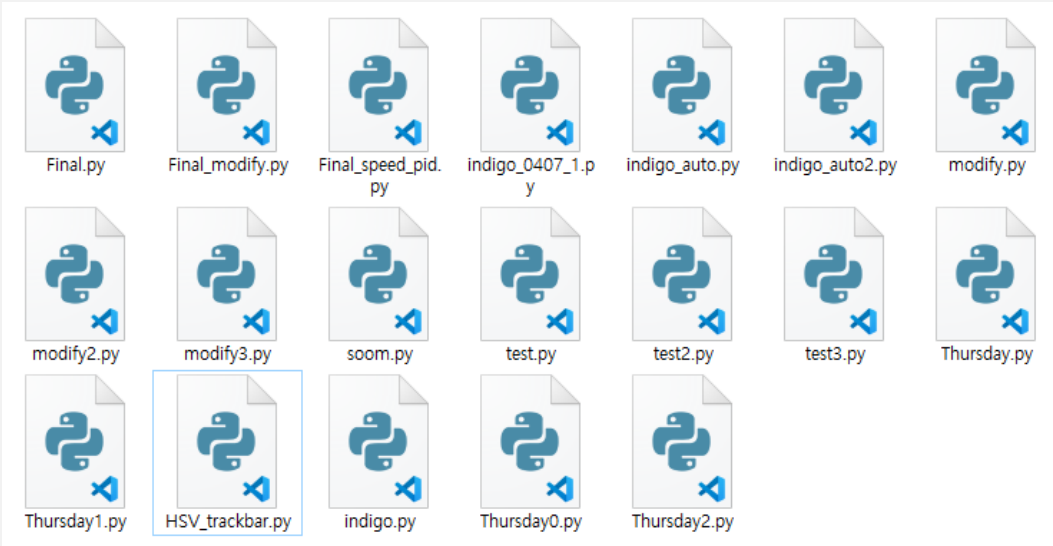
frame 중점	이동평균 필터 queue	Pid_p	Pid_i	Pid_d	speed	speed - angle * x	time	차선 침범	비고
-5	11	0.45	0.0005	0.03	25	0.3	1:05	7	
-5	11	0.45	0.0005	0.03	23	0.25	1:18	8	차선 완전 이탈 1회
						if angle > 10			
-5	11	0.45	0.0005	0.03	23	0.25			
						speed = speed - (abs(angle) - 10)			
-5	11	0.45	0.0005	0.02	23		0:50	8	차선 완전 이탈 1회
-5	11	0.45	0.0005	0.025	23		0:53	5	
-5	11	0.5	0.0005	0.03	23		0:55	7	

회고

팀 활동

- **은기** 대회가 모두 끝난 뒤 rosbag을 이용해 sensing 값을 rosbag으로 record하고 싶었으나 하지 못했다. 다행히 다른 날의 데이터가 있긴 하나, 해당 파일이 있었다면 자이카 없이도 알고리즘 수정을 할 수 있을 것이다.

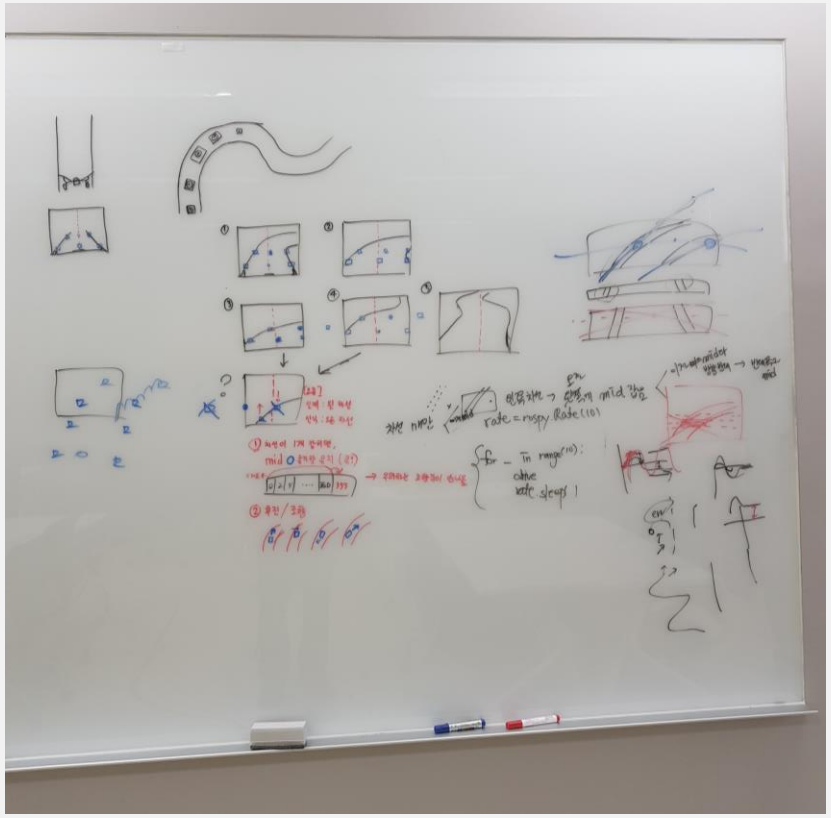
- **공통** Git을 뒤늦게 도입하였으며 제대로 commit 이력을 관리하지 못했다. 또한 Github에 협업자로 팀원을 등록해 자택에서도 수정 후 바로 공유할 수 있었으면 했다.

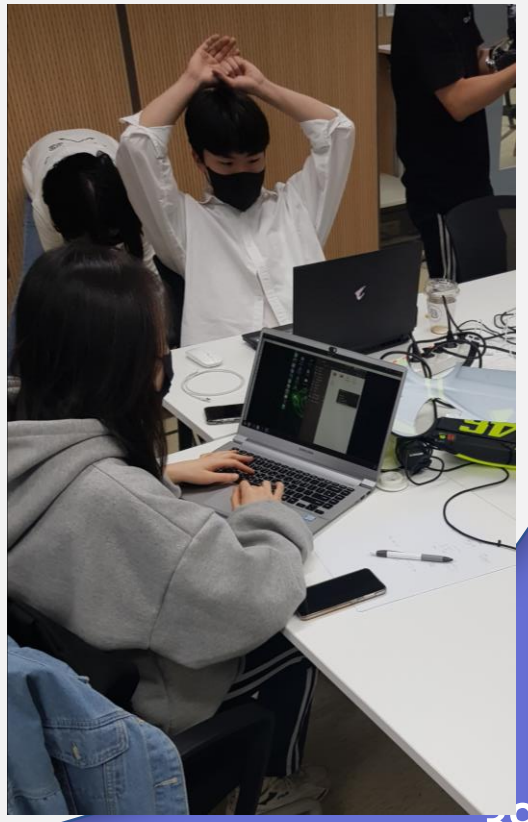


회고

팀 활동

- 은기 사진 조금 더 찍을 걸…!





팀 활동

- 은기 사진 조금 더 찍을 걸…!

자이카와 공감하는 중

은기 광선 발사

향후 계획

대영

- 임의의 픽셀값 등을 대입하지 않고, 수학적 모델이나 물리학 식을 공부하고 이를 활용해 구현하고 싶다.
- PID 외 제어 기술을 적용해보고 싶다.
- Class에서 가속(Accelerator)과 감속(Break) 기능을 따로 구별해 최고/최저 목표 속도로 가속/감속이 이뤄지도록 구현하고 싶다.
- ROI 두 개를 사용하는 아이디어를 구현하지 못했다. 도입 시 차선 검출의 정확도를 높일 수 있을 것이다.
- 차선 위치 인식에 차체의 현재 각도를 사용한다면 직선 및 곡선 판단과 제어에 유용할 것 같다.
- 순서도를 그리고 파트를 나누어 개발했다면 보다 체계적인 협업이 될 것 같다.

향후 계획

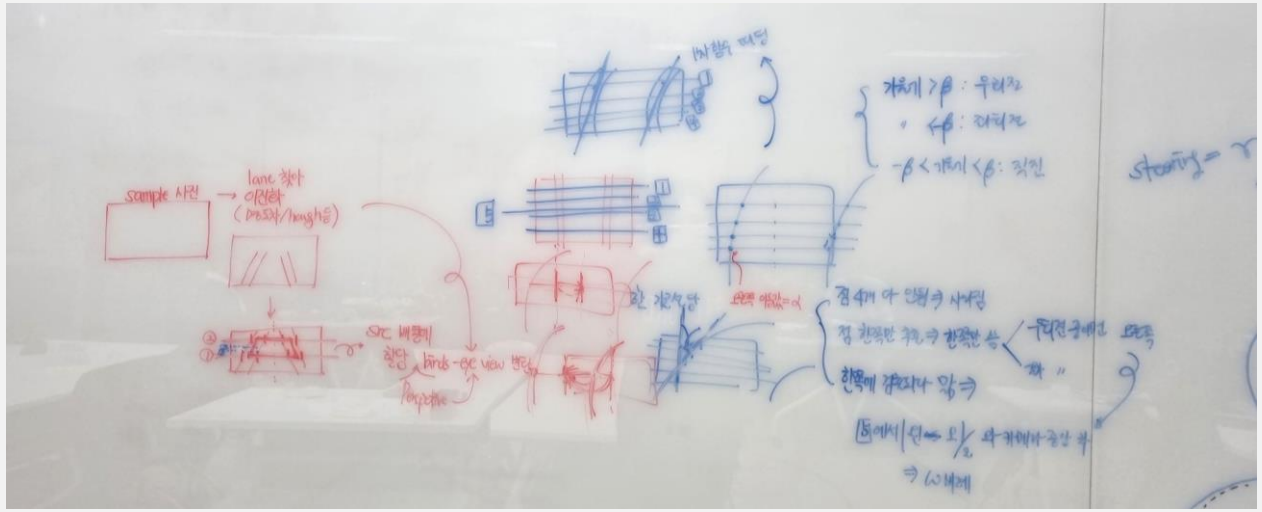
수민

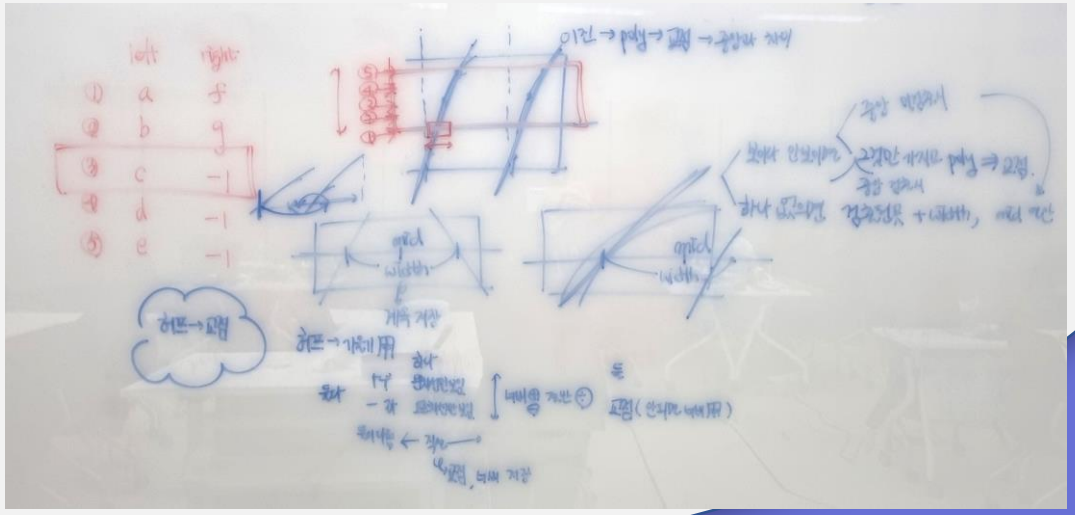
- 슬라이딩 윈도우나 버드아이뷰를 사용해 차선 검출에 도전해보고 싶다.
- 허프 변환, 슬라이딩 윈도우 외의 다른 차선 인식 기술을 알아볼 것이다.
- 튜닝하는 과정을 Git에 Commit으로 남긴다면, 되돌아가기도 편리할 것 같다.

향후 계획

은기

- PID 튜닝 방법도 다양하고, 개선된 PID 제어기도 있다. 이를 공부해보고 싶다.
- 추후에는 딥러닝이나 강화학습 기반 차선 추정을 할 수 있었으면 한다.
  꼭 기계학습이 아니더라도, 이전 차선의 위치들을 기록해 이들을 기반으로 다음 위치를 추정하는 기능을 구현해보고 싶다. 혹은 칼만 필터를 알았다면 이를 활용해도 좋았을 듯했다.
- 많은 알고리즘을 구상해보았으나 시간의 제약이 있어 아이디어로만 남았다. Bag 파일을 바탕으로 하나씩 구현해보아도 좋을 것 같다.





Thank You

Indigo
Let's Go
We Just Go

Team A1-1
김대영 · 강수민 · 한은기