스마트모빌리티설계

[12주차] PID 제어 과제

1. 코드

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import numpy as np
import cv2, math
import rospy, rospkg, time
from sensor_msgs.msg import Image
from cv_bridge import CvBridge
from xycar_msgs.msg import xycar_motor
from math import *
import signal
import sys
import os
import random

class Car():
    def __init__(self, speed=5, P=1, I=0, D=0):
        # ros init
        rospy.init_node('Xycar')
        self.image_sub = rospy.Subscriber("/usb_cam/image_raw/", Image, self.img_callback)

        # control
        self.motor_pub = rospy.Publisher('xycar_motor', xycar_motor, queue_size=1)
        self.speed = speed
        self.angle = None

        # image data
        self.image = np.empty(shape=[0])
        self.img_ready = None
        self.bridge = CvBridge()

        self.img = None
        self.display_img = None
        self.gray = None
        self.blur_gray = None
        self.edge_img = None

        self.WIDTH = 640
        self.HEIGHT = 480
        self.ROI_ROW = 250              # ROI row
        self.ROI_HEIGHT = self.HEIGHT - self.ROI_ROW
        self.L_ROW = self.ROI_HEIGHT - 120  # Row for position detection

        # line detection
        self.prev_x_left = 0
        self.prev_x_right = self.WIDTH

        # PID
        self.P = P
        self.I = I
        self.D = D
        self.i_error = 0.0
        self.prev_error = 0.0

        # time
        self.start_time = time.time()
```

```
        self.end_time = time.time()

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

    def motor_control(self, angle, speed):
        motor_msg = xycar_motor()
        motor_msg.angle = angle
        motor_msg.speed = speed

        self.motor_pub.publish(motor_msg)

    def PID(self, input_data, target_data=320):
        self.end_time = time.time()
        dt = self.end_time - self.start_time
        self.start_time = self.end_time

        error = target_data - input_data
        derror = error - self.prev_error

        p_error = self.P * error
        self.i_error = self.i_error + self.I * error * dt
        d_error = self.D * derror / dt if dt != 0 else 0

        output = p_error + self.i_error + d_error
        self.prev_error = error

        if output > 50:
            output = 50
        elif output < -50:
            output = -50

        return -output

    def image_preprocess(self):
        self.img = self.image.copy()
        self.display_img = self.img    # to show

        self.img_ready = False

        self.gray = cv2.cvtColor(self.img, cv2.COLOR_BGR2GRAY)
        self.blur_gray = cv2.GaussianBlur(self.gray, (5, 5), 0)
        self.edge_img = cv2.Canny(np.uint8(self.blur_gray), 50, 100)

    def track_line(self):
        # Find lines in ROI
        roi_img = self.img[self.ROI_ROW:self.HEIGHT, 0:self.WIDTH]
        roi_edge_img = self.edge_img[self.ROI_ROW:self.HEIGHT, 0:self.WIDTH]

        all_lines = cv2.HoughLinesP(roi_edge_img, 1, math.pi / 180, 30, 50, 20)
        if all_lines is None:
            return False

        # Slope filtering
        slopes = []
        filtered_lines = []
```

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

        if (x2 == x1):
            slope = 1000.0
        else:
            slope = float(y2 - y1) / float(x2 - x1)

        if 0.2 < abs(slope):
            slopes.append(slope)
            filtered_lines.append(line[0])

    # Separate left & right lines
    left_lines = []
    right_lines = []
    for j in range(len(slopes)):
        line = filtered_lines[j]
        slope = slopes[j]

        x1,y1, x2, y2 = line
        if (slope < 0) and (x2 < self.WIDTH / 2):
            left_lines.append(line.tolist())
        elif (slope > 0) and (x1 > self.WIDTH / 2):
            right_lines.append(line.tolist())

    # Left: Red & Right: Yellow
    line_draw_img = roi_img.copy()
    for line in left_lines:
        x1,y1, x2, y2 = line
        cv2.line(line_draw_img, (x1,y1), (x2, y2), (0, 0, 255), 2)     # RED
    for line in right_lines:
        x1,y1, x2, y2 = line
        cv2.line(line_draw_img, (x1,y1), (x2, y2), (0, 255, 255), 2)    # YELLOW

    # Major LEFT lines
    m_left, b_left = 0.0, 0.0
    x_sum, y_sum, m_sum = 0.0, 0.0, 0.0
    size = len(left_lines)
    if size != 0:
        for line in left_lines:
            x1, y1, x2, y2 = line
            x_sum += x1 + x2
            y_sum += y1 + y2
            if (x2 != x1):
                m_sum += float(y2 - y1) / float(x2 - x1)
            else:
                m_sum += 0
        x_avg = x_sum / (size * 2)
        y_avg = y_sum / (size * 2)
        m_left = m_sum / size
        b_left = y_avg - m_left * x_avg

        if m_left != 0.0:
            x1 = int((0.0 - b_left) / m_left)
            x2 = int((self.ROI_HEIGHT - b_left) / m_left)
            cv2.line(line_draw_img, (x1, 0), (x2, self.ROI_HEIGHT), (255, 0, 0), 2)    # BLUE
```

```
        # Major RIGHT lines
        m_right, b_right = 0.0, 0.0
        x_sum, y_sum, m_sum = 0.0, 0.0, 0.0
        size = len(right_lines)
        if size != 0:
            for line in right_lines:
                x1, y1, x2, y2 = line
                x_sum += x1 + x2
                y_sum += y1 + y2
                if (x2 != x1):
                    m_sum += float(y2 - y1) / float(x2 - x1)
                else:
                    m_sum += 0
            x_avg = x_sum / (size * 2)
            y_avg = y_sum / (size * 2)
            m_right = m_sum / size
            b_right = y_avg - m_right * x_avg

            if m_right != 0.0:
                x1 = int((0.0 - b_right) / m_right)
                x2 = int((self.ROI_HEIGHT - b_right) / m_right)
                cv2.line(line_draw_img, (x1, 0), (x2, self.ROI_HEIGHT), (255, 0, 0), 2)    # BLUE

        if m_left == 0.0:
            x_left = self.prev_x_left
        else:
            x_left = int((self.L_ROW - b_left) / m_left)
            self.prev_x_left = x_left

        if m_right == 0.0:
            x_right = self.prev_x_right
        else:
            x_right = int((self.L_ROW - b_right) / m_right)
            self.prev_x_right = x_right

        self.prev_x_left = x_left
        self.prev_x_right = x_right

        x_midpoint = (x_left + x_right) // 2

        view_center = self.WIDTH // 2

        cv2.line(line_draw_img, (0,self.L_ROW), (self.WIDTH, self.L_ROW), (0, 255, 255), 2)
        cv2.rectangle(line_draw_img, (x_left - 5, self.L_ROW - 5), (x_left + 5, self.L_ROW + 5),
(0,255,0), 4)
        cv2.rectangle(line_draw_img, (x_right - 5, self.L_ROW - 5), (x_right + 5, self.L_ROW +
5), (0,255,0), 4)
        cv2.rectangle(line_draw_img, (x_midpoint - 5, self.L_ROW - 5), (x_midpoint + 5,
self.L_ROW + 5), (255,0,0), 4)
        cv2.rectangle(line_draw_img, (view_center - 5, self.L_ROW - 5), (view_center + 5,
self.L_ROW + 5), (0,0,255), 4)

        self.display_img[self.ROI_ROW : self.HEIGHT, 0 : self.WIDTH] = line_draw_img

        # control
        self.angle = self.PID(x_midpoint)
```

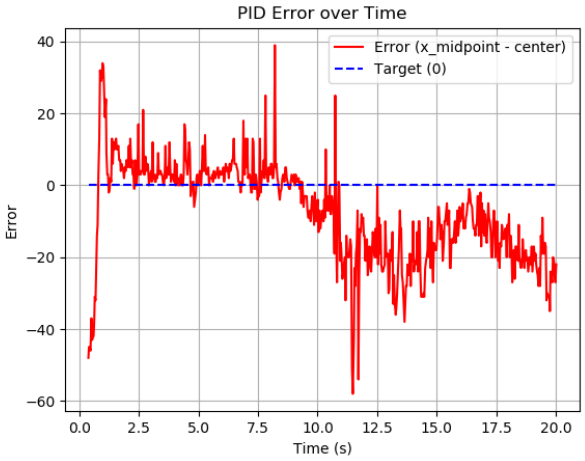
```
        self.motor_control(self.angle, self.speed)

        return True

    def drive(self):
        print("Self-driving start!")

        while not self.image.size == (self.WIDTH * self.HEIGHT * 3):
            continue

        while not rospy.is_shutdown():
            while self.img_ready == False:
                continue

            # Image preprocessing
            self.image_preprocess()

            # etc situations

            # LINE DETECTION
            if self.track_line() == False:
                continue

            cv2.imshow("Self-driving", self.edge_img)
            cv2.waitKey(1)

if __name__ == "__main__":
    car = Car()
    car.drive()
```

2. 코드 설명
   A. signal_handler()
      i. CTRL+C(interrupt) 발생 시 시스템을 안전하게 종료하는 함수
   B. img_callback()
      i. 이미지 관련 subscriber에 등록되어 ROS topic에서 이미지 데이터를 수신할 때마다 호출되는 함수로, OpenCV 이미지로 변환하는 함수
   C. drive()
      i. 조향각과 속도를 인자로 받아 자동차를 제어하는 메시지를 publish하는 함수
   D. PID()
      i. 조향할 때 진동을 최소화하기 위해 조향에 PID 제어를 적용해주는 함수
      ii. 현재값을 입력으로 받아 목표값과의 오차를 이용하여 PID 제어를 수행
   E. image_preprocess()
      i. 이미지를 전처리하는 함수
      ii. topic으로 전달 받은 이미지를 흑백으로 변환하고 blur 처리한 후 Canny edge detection을 수행
   F. track_line()
      i. 차선 인식 기반 자율주행 전체 로직을 수행하는 함수
      ii. 전처리한 이미지에서 차선을 찾아낸 후, 차선의 중앙 위치와 이미지의 중앙 위치를 비교

하여 차량이 차선을 잘 따라가도록 하고 인식한 차선에 대해서 실시간으로 시각화하는 함수

3. 결과 분석

동일한 주행 구간(직진 구간 후 곡선 구간)과 20초의 동일한 작동 시간을 설정한 후 PID 제어 값을 변경해가며 주행하였고 주행 영상을 촬영함과 동시에 이미지 중앙 pixel과 차선의 중앙 pixel 간 시간에 따른 오차를 시각화하여 비교하였다.

P I D	
1 / 0 / 0	

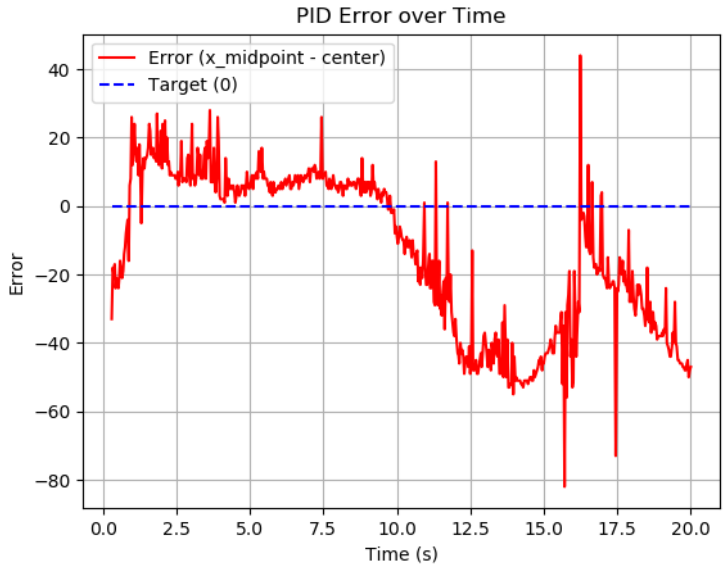
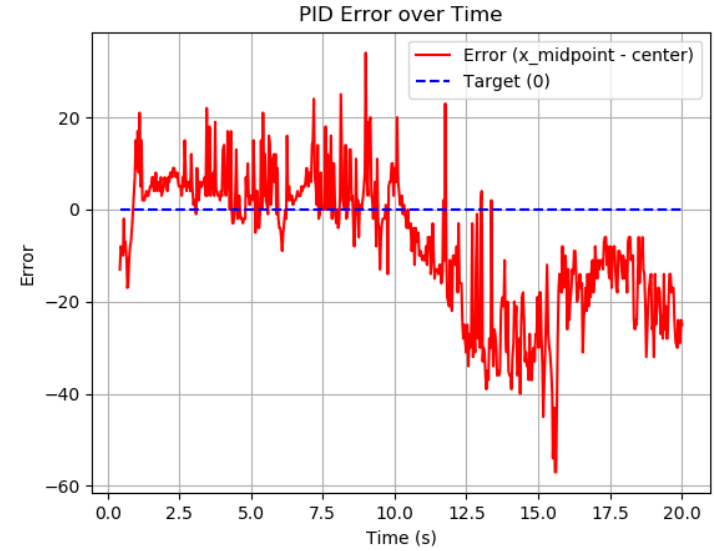
PID의 초기값은 1/0/0으로 설정한 후 실험을 시작하였으며, 초기 상태에서 P값, I값, D값을 조금씩 변화시켜 보았다.

첫 번째 그래프를 보면 0~20 내외에서 진동하는 오차가 보이는 구간이 직선주행 구간, 이후 오차가 증가하여 -40 이상까지 좀 더 불규칙적으로 그래프가 오르내리는 구간이 곡선 구간이라고 보여진다. 또한 10초 부근을 기점으로 직선 구간에서 곡선 구간으로 바뀌는 것을 추측할 수 있다. 이후 다른 PID 값을 사용한 그래프에서도 그래프의 개형 자체는 직선 구간에서 진동하며 횡보하고 곡선 구간에서 좀 더 불규칙해지는 양상을 따른다.

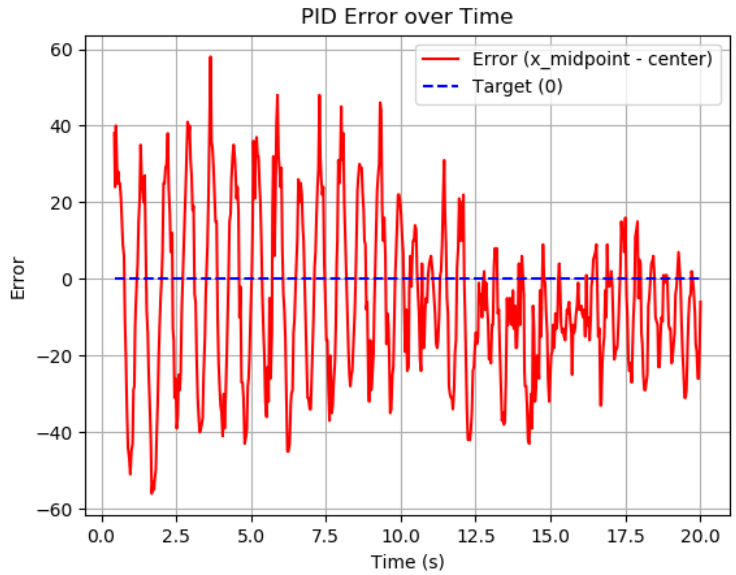
그래프만 볼 때 진동수 자체는 많은 것으로 보이지만, 첨부한 영상을 통해 그래프에서의 진동이 실제 자동차의 조향으로 전달되지는 않고 부드럽게 직진하는 것을 확인할 수 있다. 따라서 소프트웨어에서 전달하는 조향각이 작은 진폭과 높은 진동수로 진동할 경우 하드웨어적으로는 이러한 진동이 크게 반영되지 않는 것으로 추측된다.

또한 직진 구간에서 조향이 완벽하게 중앙이라면 이론상으로는 오차가 0 부근에서 양수, 음수쪽으로 고루 진동해야 하지만, 실험결과 0~20 사이의 양수 부근에서 진동하는 것을 확인할 수 있다. 따라서 차량 제어에 오차가 있을 것으로 추측된다.

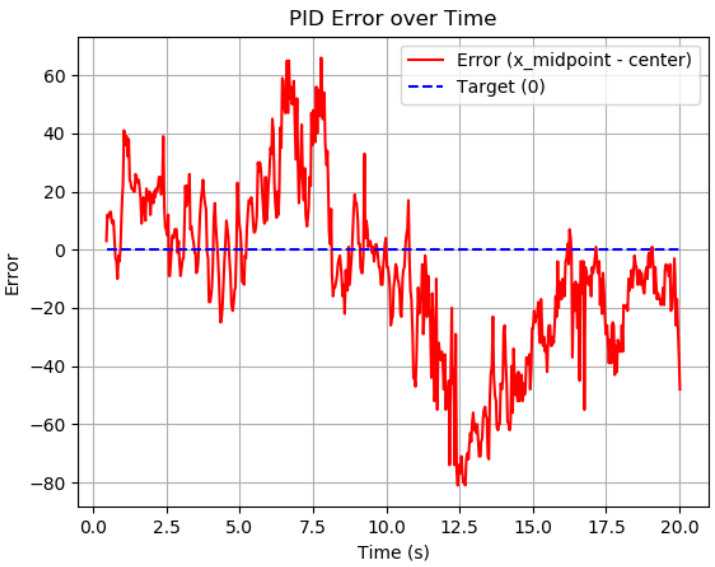
곡선 구간의 경우 영상을 확인하였을 때 차량이 차선의 중앙부근을 유지하며 도는 것이 아니라 인코스를 그리며 곡선 구간을 도는 것을 확인할 수 있는데, 이는 ROI 상으로 자동차가 미리 조향을 하기 때문에 발생하며, 또한 PID 제어 시스템을 통한 조향 가중치의 증가로 인한 반응속도의 향상 또한 원인으로 보인다. 추후 이러한 인코스 주행 원인을 최종 주행에서 기록 단축을 위한 기법으로 사용할 수 있을 것으로 고려된다.

0.5 / 0 / 0	
0.8 / 0 / 0	

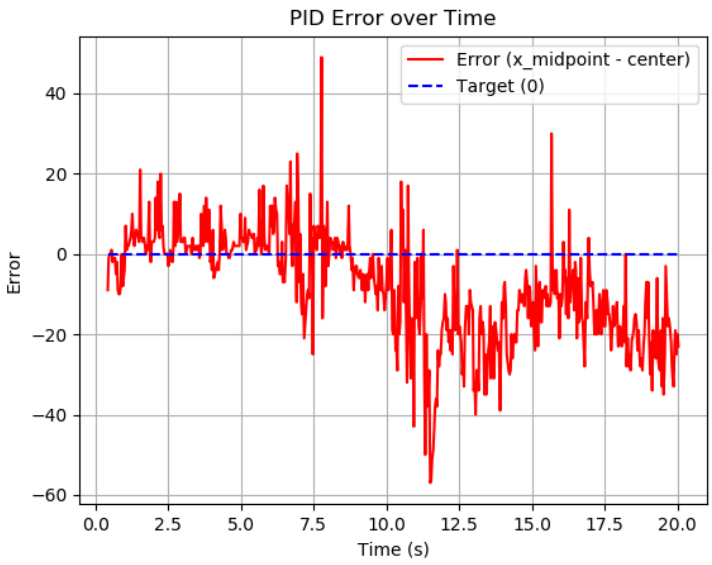
실제로 P값을 감소시킨 PID 값 0.5 / 0 / 0, 0.8 / 0 / 0 의 그래프를 1 / 0 / 0의 그래프와 비교하면 반응속도의 하락으로 인해 곡선 구간에서 평균오차가 증가하는 것을 확인할 수 있다. 또한 빠르게 주어진 코스를 완주하는 것을 목표로 하는 만큼 인코스로 주행하는 것은 바람직하다고 생각되어 이를 유지하기로 하였다.

2 / 0 / 0	

반면 2 / 0 / 0 과 같이 P값을 지나치게 증가시킬 경우 오버슈팅이 발생하고 그만큼 오차의 주기도 증가하는 것을 확인할 수 있다. 실제로 촬영한 영상에서도 바퀴가 심하게 진동하는 것을 확인할 수 있다.

1 / 0 / 0.5	

다섯 번째 그래프의 경우 PID값 중 D값을 증가시킨 주행 데이터이다. 이론대로 오차의 진동수가 줄어든 것을 확인할 수 있었으나 실제 주행 영상을 확인해 보면 오차의 진동수가 줄어든 것이 오히려 소프트웨어의 오차의 진동이 하드웨어에 더 잘 반영되는 결과를 초래하여 차량이 전진 상황에서 진동하며 주행하는 것을 확인할 수 있다. 또한 곡선 구간에서도 반응속도가 감소하여 평균오차가 증가한 것을 확인할 수 있었다.

1 / 0.0004 / 0	

I값을 0.0004로 증가시킨 경우 그래프상으로는 큰 차이가 없었고 주행 영상 상으로 1 / 0 / 0에 비해 많은 진동이 발생함을 확인할 수 있었다.

결론적으로 실험결과 PID의 초기값인 1 / 0 / 0이 최적의 PID 값이라고 판단하여 초기값을 유지하기로 하였다.