결과 보고서

임베디드응용및실습.
11주차 실습 과제

과 목 명	임베디드응용및실습
학 번	2019161068
이 름	백승진
제 출 일	2024년 11월 15일

1. 1m 내외의 코스에서 직진 및 좌/우회전 라인 트레이싱이 가능하도록 코딩하시오.

- 노란색 선/ 흰색 선 관계없음
- 도로를 완전히 벗어나는 경우가 없도록 함
  - 선을 밟거나 넘는 경우는 상관없음

1) 코드

```
import cv2 as cv
import numpy as np
import threading, time
import SDcar

def func_thread():
    i = 0
    while True:
        print("alive!!")
        time.sleep(1)
        i = i + 1
        if is_running is False:
            break

def key_cmd(which_key):
    print('which_key', which_key)
    is_exit = False
    global enable_linetracing
    if which_key & 0xFF == 82:
        print('up')
        car.motor_go(100)
    elif which_key & 0xFF == 84:
        print('down')
        car.motor_back(100)
    elif which_key & 0xFF == 81:
        print('left')
        car.motor_left(100)
    elif which_key & 0xFF == 83:
        print('right')
        car.motor_right(100)
    elif which_key & 0xFF == 32:
        car.motor_stop()
        print('stop')
    elif which_key & 0xFF == ord('q'):
        car.motor_stop()
        print('exit')
        is_exit = True
    elif which_key & 0xFF == ord('e'):
        enable_linetracing = True
        print('enable_linetracing : ', enable_linetracing)
    elif which_key & 0xFF == ord('w'):
        enable_linetracing = False
        car.motor_stop()
        print('enable_linetracing 2: ', enable_linetracing)
```

```
    return is_exit

def detect_maskY_HSV(frame):
    crop_hsv = cv.cvtColor(frame, cv.COLOR_BGR2HSV)
    crop_hsv = cv.GaussianBlur(crop_hsv, (5, 5), cv.BORDER_DEFAULT)
    # need to tune params
    mask_Y = cv.inRange(crop_hsv, (25, 50, 100), (35, 255, 255))
    return mask_Y

def detect_maskY_BGR(frame):
    B = frame[:, :, 0]
    G = frame[:, :, 1]
    R = frame[:, :, 2]
    Y = np.zeros_like(G, np.uint8)
    # need to tune params
    Y = G * 0.5 + R * 0.5 - B * 0.7
    Y = Y.astype(np.uint8)
    Y = cv.GaussianBlur(Y, (5, 5), cv.BORDER_DEFAULT)
    # need to tune parmas
    _, mask_Y = cv.threshold(Y, 100, 255, cv.THRESH_BINARY)
    return mask_Y

def line_tracing(cx):
    # print('cx : ', cx)
    # pirnt('v_x_grid : ', v_x_grid)
    global moment
    global v_x
    tolerance = 0.1
    diff = 0

    if moment[0] != 0 and moment[1] != 0 and moment[2] != 0:
        avg_m = np.mean(moment)
        diff = np.abs(avg_m - cx) / v_x

    print('diff = {:.4f}' .format(diff))

    if diff <= tolerance:
        moment[0] = moment[1]
        moment[1] = moment[2]
        moment[2] = cx

        if v_x_grid[0] <= cx < v_x_grid[1]:
            car.motor_left(speed)
            print('turn right')
        elif v_x_grid[1] <= cx < v_x_grid[2]:
            car.motor_go(speed)
            print('go')
        elif v_x_grid[2] <= cx:
            car.motor_right(speed)
            print('turn left')
        else:
            # 선을 벗어난 경우
```

```
                car.motor_go(speed)
                print('go')
                moment = [0, 0, 0]

def show_grid(img):
    h, _, _ = img.shape
    for x in v_x_grid:
        # print('show_gird', x)
        cv.line(img, (x, 0), (x, h), (0, 255, 0), 1, cv.LINE_4)

def main():
    camera = cv.VideoCapture(0)
    camera.set(cv.CAP_PROP_FRAME_WIDTH,v_x)
    camera.set(cv.CAP_PROP_FRAME_HEIGHT,v_y)

    try:
        while(camera.isOpened()):
            ret, frame = camera.read()
            frame = cv.flip(frame, -1)
            cv.imshow('camera', frame)

            # image processing start here
            crop_img = frame[120:, :]

            maskY = detect_maskY_BGR(crop_img)
            # maskY = detect_maskY_HSV(crop_img)

            contours, _ = cv.findContours(maskY, cv.RETR_TREE, cv.CHAIN_APPROX_SIMPLE)
            # print('len(contours), ', len(contours))

            if len(contours) > 0:
                c = max(contours, key=cv.contourArea)
                m = cv.moments(c)

                cx = int(m['m10']/(m['m00'] + epsilon))
                cy = int(m['m01']/(m['m00'] + epsilon))
                cv.circle(crop_img, (cx, cy), 3, (0, 255, 0), 1)

                cv.putText(crop_img, str(cx), (10, 10), cv.FONT_HERSHEY_DUPLEX, 0.5, (0, 255, 0))
                # print('enable_lintracing', enable_linetracing)

                if enable_linetracing == True:
                    line_tracing(cx)
            show_grid(crop_img)

            cv.imshow('maskY', maskY)
            cv.imshow('crop_img', cv.resize(crop_img, dsize=(0, 0), fx=2, fy=2))

            # image processing end here
            is_exit = False
            which_key = cv.waitKey(20)
            if which_key > 0:
```

```
                    is_exit = key_cmd(which_key)
                if is_exit is True:
                    cv.destroyAllWindows()
                    break
        except Exception as e:
            print(e)
            global is_running
            is_running = False

if __name__ == '__main__':
    epsilon = 0.01
    speed = 30

    v_x = 320
    v_y = 240
    v_x_grid = [int(v_x  * i / 10) for i in range(1, 10)]

    moment = np.array([0, 0, 0])

    print(v_x_grid)

    t_task1 = threading.Thread(target = func_thread)
    t_task1.start()

    car = SDcar.Drive()

    is_running = True
    enable_linetracing = False
    main()
    is_running = False
    car.clean_GPIO()
    print('end vis')
```

2) 결과

영상 참조

3) 해석

이번 과제에서는 강의 자료에 있는 코드를 사용했기 때문에 크게 다른 점은 없다.

우선 필요한 모듈을 모두 import 한 후, 연결이 잘 되어 있는지 확인하기 위해 1초마다 "alive!!" 메시지를 출력한다. 다음으로, 키보드 입력을 받아 자동차가 움직이도록 하는 key_cmd 함수를 정의한다. 해당 함수에서는 조건문을 통해 킷값을 확인하고, 이를 통해 과제에서는 사용하지 않지만, 방향키로 자동차를 제어할 수 있다. 또한, q를 입력하면 프로그램을 종료하고, e를 입력하면 과제가 요구하는 내용인 라인 트레이싱을 수행한다. 라인 트레이싱을 종료하기 위해서는 w를 입력하면 된다.

다음으로 노란색 선을 추출하기 위해 detect_maskY_BGR 함수를 정의한다(detect_maskY_HSV 함수도 있지만, 이번 과제에서는 사용하지 않았다). 해당 함수에서는 실시간으로 받은 3채널 영상을 채널별로 분리하여 얻은 3개의 값을 적절히 연산하여 해당 값이 노란색인지 아닌지 판단하고 노란색이라면 흰색, 아니라면 검은색으로 이진화한다. 여기서 노이즈 개선을 위해 가우시안 블러 처리해주며, 이진화 임곗값은 100이다.

다음, 본격적으로 과제 수행을 위한 line_tracing 함수를 정의한다. 여기서 인자인 cx는 Contour의 중심점으로, moment라고 불리는데 이는 뒤쪽에서 계산한다. 과제에서는 이전 값들과 현재 값을 비교하여 조금 더 자연스러운 움직임을 구현했다. 코드는 이전 값 2개와 현재 값의 평균을 계산한 후, 이 값과 현재 값의 차이를 계산한다. 이 오차가 허용 범위 내에 있을 때 즉, 선을 벗어나지 않은 상태라면 line_tracing을 수행하고, 그렇지 않은 경우에는 그냥 직진하도록 했다. 여기서 line_traceing은 moment의 위치에 따라 다른 동작을 수행하는데, 만약 너무 좌측으로 치우쳐져 있다면 우회전, 너무 우측으로 치우쳐져 있다면 좌회전하는 간단한 동작이다. v_x_grid는 원 영상을 9 등분한 x 좌표들의 집합이며, 강의 자료보다 작은 값들로 기준을 설정하였는데, 이유는 최대한 좌측 중앙선을 침범하지 않기 위해서이다.

line_tracing 함수 다음은 show_grid 함수로, 이미지에 9 등분의 grid를 그려주는 함수이다.

다음은 main 함수로, 실시간으로 영상을 받아오고, 이를 보기 편하게 상하좌우 반전시킨다. 다음으로 line tracing을 위한 차선은 이미지의 아래쪽에만 있기 때문에, 영상의 아래쪽만 연산하도록 crop해준다. 다음으로 영상에서 노란색 선을 추출하기 위해 위에서 언급한 detect_maskY_BGR 함수를 호출하고, contours를 계산한다. 만약 contours가 존재한다면, contours의 무게중심점을 c로 설정하고 해당 점의 x 좌표와 y 좌표는 각각 cx, cy로 설정한다. 또한, 해당 점은 초록색으로 표기하고, 해당 점을 기준으로 line_tracing을 진행한다. 다음으로 카메라의 프레임을 확인하기 위해 영상을 출력한다.

지금까지는 import 하거나 함수들만 정의했으므로, 이 파일을 실행한다고 해서 어떤 동작도 하지 않는다. 실행을 위해서는 main 함수를 호출해야 하는데, 아무 파일에서나 동작하지 않고, main 모듈에서만 동작하도록 조건문을 사용한다. 만약 해당 파일이 main 모듈이라면 전역 변수와 영상의 해상도를 초기화한 후, 화면 크기에 맞게 grid를 분할한다. 다음으로 moment 배열을 초기화한 후, func_thread 쓰레드를 활성화한다. 여기까지 진행하고 나서야 main 함수를 호출한다. 만약 q를 입력하여 프로그램이 종료되면 자원을 해제한다.

위의 과제 결과와 코드는 모두 강의 자료에 있는 내용이고, 변수의 값 정도만 조정했을 뿐이다. 해당 코드는 노란 선을 기준으로 라인 트레이싱을 진행하기 때문에, 선을 침범할 수밖에 없다. 그래서 선을 침범하지 않도록 흰 선을 기준으로 라인 트레이싱을 진행하려고 시도했다. 하지만 흰 선은 중간중간 끊겨 있기에 핵심 moment를 찾기 어려울 수 있다. 따라서 선을 이어줄 필요가 있는데, 이는 모폴로지 연산을 통해 해결할 수 있다. 흰 선을 잇기 위해서는 팽창 연산을 수행하면 되고, 팽창된 흰 선을 다시 감소하기 위해 이어서 침식 연산을 수행한다. 실제로 이러한 과정을 통해 흰 선만을 검출하고 moment를 흰 선에 위치하는 것까지 성공했다. 하지만, 문제는 90도 좌회전 혹은 우회전 코스에서 연산을 수행하는 속도보다 자동차의 이동속도가 더욱 빨라 흰 선을 지나치고 말았다. 자동차의 속도를 줄여도 문제는 해결되지 않았다. 이유는 crop 한 이미지의 크기와 흰색 마스크의 크기가 작아, moment가 영상 밖으로 쉽게 나갈 수밖에 없었기 때문인데, 이미지의 크기를 줄인 이유는 트랙의 바깥쪽 흰 영역을 인식하지 않기 위해서이다. 아무튼 아쉽게도 선을 침범하지 않고, 노란 선 내에서만 주행하는 코드는 완성하지 못했다. 하지만 거의 마무리 단계까지 도달했다고 생각하며, 시간이 조금만 있다면 완성할 수 있기 때문에 이후 시간이 날 때면 완성해보고 싶다.