나상진 포트폴리오

[드론 자율주행 구현 코드 (Python)]

라이브러리 활용, 스터디 중인 상황이라 코드가 깔끔하게 정리되어 있지는 않습니다.

제약조건: 드론 전방의 tof센서 1개만을 이용하여 복도형태의 ㅁ자 공간 자율주행

import robomaster

from robomaster import robot

from robomaster import camera

import time

import cv2

import pygame

import threading

import numpy as np

camera\_end = True   #camera\_display thread 종료 여부 변수

area = 0    #탐지한 색상 크기 (얼마나 떨어져있는가)

def init\_key():

    """ pygame 초기화

    """

    pygame.init()

    win = pygame.display.set\_mode((400,400))

def get\_key(keyName):

    """ 해당 키(키보드 입력)가 입력되었는지 확인

    Args: keyName (string): Key Name

    Returns: bool: keyName이 눌렸는지 반환

    """

    result = False

    for eve in  pygame.event.get():

        pass

    keyInput = pygame.key.get\_pressed()

    myKey = getattr(pygame, 'K\_{}'.format(keyName))

    if keyInput[myKey]:

        result = True

    pygame.display.update()

    return result

def get\_keyboard\_input():

    """ 현재 어떤 키입력들이 있는가

    Returns: List: [좌우, 앞뒤, 위아래, 좌우회전, 이륙, 착륙]

    좌우, 앞뒤, 위아래, 좌우회전: [-100,100]의 값 전달

    이륙, 착륙: bool

    """

    start = False

    end = False

    lr,fb,ud,ro = 0,0,0,0

    dis = 30

    if get\_key("LEFT"):

        lr = dis

    elif get\_key("RIGHT"):

        lr = -dis

    if get\_key("UP"):

        fb = dis

    elif get\_key("DOWN"):

        fb = -dis

    if get\_key("w"):

        ud = dis

    elif get\_key("s"):

        ud = -dis

    if get\_key("y"):

        start = True

    elif get\_key("u"):

        end = True

    return [lr, fb, ud, ro, start, end]

def takeoffDrone():

    """ 드론 이륙

    """

    # t1\_led.set\_led(r=255,g=0,b=0)

    # t1\_led.set\_led\_breath(freq=1, r=0, g=0, b=255)

    # time.sleep(3)

    # t1\_led.set\_led(r=0,g=255,b=0)

    # time.sleep(1)

    t1\_flight.takeoff().wait\_for\_completed()

    #.wait\_for\_completed()는 해당 동작이 마무리 될때까지 대기

def detectBlock():

    """ 장애물 유무 확인

    Returns: bool: 장애물 유무

    """

    dis = get\_distance()   #tof 거리센서로 거리측정

    print("distance: ",dis)

    if dis < 900:   #0.9m안에 장애물있는가

        return True

    return False

def battery\_display():

    """ 현재 배터리 잔량 확인후 매트릭스 LED에 표시

    """

    now\_battery = t1\_battery.get\_battery()

    format\_battery = now\_battery%8 + 1

    led\_value = "00000000"\*7 + "b"\*format\_battery + "r"\*(8-format\_battery)

    t1\_led.set\_mled\_boot(led\_value)

def get\_distance():

    """ 현재 거리 측정

    Returns:

        flaot: 거리 (mm 단위)

    """

    dis = t1\_sensor.get\_ext\_tof()

    return dis

def auto\_pilot():

    """ 자율주행

    단순히 전방으로 이동하면서 장애물을 만나면

    기체 좌,우를 살핀 후 더 넓은 구역으로 회전 후 다시 전방 주행

    """

    init\_key()    # key입력받기 초기화

    takeoffDrone()  #드론 이륙

    count = 0   #자율주행 진행상황, 동작 횟수 기입할 변수

    trap\_count = 0  #트랩(사방이 막힌상황)인지 확인하는 변수

    temp = 100

    pre\_rotate = -1 # -1: 좌, 1: 우

    global area     #특정 색상 영역의 넓이를 가져오는 변수

    d\_fblr = [1,0]     # 1,0: 앞으로  -1,0: 뒤로  0,1: 오른쪽으로  0,-1: 왼쪽으로

    move\_state = [0,0]  #출발지 0,0으로 부터 얼마나 떨어져 있는지 확인  +: 앞,우  -: 뒤, 왼

    while True:

        print(t1\_drone.get\_height())

        vals = get\_keyboard\_input()     #키보드 input 받아오기

        if vals[5]:    #u키를 꾹 눌러서 강제 종료

            break

        if count % 10 == 0:

            print("Current Battery: ", t1\_battery.get\_battery(), "%")

        if count > 15:

            #출발하고 조금 있다가 0,0(즉, 출발지로 다시 돌아오면 착륙, 오차를 감안해서 0,0 +-3을 둔다.)

            if abs(move\_state[0]) <= 3 and abs(move\_state[1]) <= 3:

                print("출발지로 돌아옴, 착륙")

                break

        if detectBlock():   # 장애물을 감지하면

            if temp >= 1 and temp <= 2:

                t1\_flight.rc(a=0,b=0,c=0,d=0)  # 앞으로 가던거 멈추기

                time.sleep(0.05)

                t1\_flight.rotate(-90\*pre\_rotate).wait\_for\_completed()   #원래 가던 방향 바라보기

                pre\_rotate = -pre\_rotate

                if abs(d\_fblr[0] == 1):

                    d\_fblr[0], d\_fblr[1] = d\_fblr[1], pre\_rotate\*d\_fblr[0]

                else:

                    d\_fblr[0], d\_fblr[1] = -pre\_rotate\*d\_fblr[1], d\_fblr[0]

                temp = 4

                continue

            t1\_flight.rc(a=0,b=0,c=0,d=0)  # 앞으로 가던거 멈추기

            time.sleep(0.05)

            trap\_count += 1

            temp = 0

            t1\_flight.rotate(-90).wait\_for\_completed() # 왼쪽으로 기체 90도 회전

            dis\_left = get\_distance()   # 진행하던 방향의 왼쪽 편의 거리 측정, 저장

            time.sleep(0.05)

            t1\_flight.rotate(180).wait\_for\_completed()  # 오른쪽으로 기체 90도 회전

            dis\_right = get\_distance()  # 진행하던 방향의 왼쪽 편의 거리 측정, 저장

            time.sleep(0.05)

            print("Left: ",dis\_left,",    Right: ",dis\_right)

            if dis\_left >= dis\_right:    # 더 트인 공간을 찾아 기체 회전

                print("Select Left, ", end=" ")

                pre\_rotate = -1

                t1\_flight.rotate(-180).wait\_for\_completed()

            else:

                print("Select Right", end=" ")

                pre\_rotate = 1

            if abs(d\_fblr[0] == 1):

                d\_fblr[0], d\_fblr[1] = d\_fblr[1], pre\_rotate\*d\_fblr[0]

            else:

                d\_fblr[0], d\_fblr[1] = -pre\_rotate\*d\_fblr[1], d\_fblr[0]

            print("End rotate ", count)

        else:   # 장애물이 없을 시

            # if area > 200:  # 특정 색상을 탐지했을때

            #     print("Red")

            #     trap\_count += 1

            #     t1\_flight.rotate(-90).wait\_for\_completed() # 왼쪽으로 기체 90도 회전

            #     print("End rotate")

            # else:

            count += 1

            temp += 1   #벽이 아닌 장애물이였을때 우회를 위해 다시 가던 방향을 확인해보는 변수 (일정 거리를 갔을 때 확인)

            trap\_count = 0  # trap\_count 초기화

            if temp == 3:

                t1\_flight.rc(a=0,b=0,c=0,d=0)  # 앞으로 가던거 멈추기

                time.sleep(0.05)

                t1\_flight.rotate(-90\*pre\_rotate).wait\_for\_completed()   #원래 가던 방향 바라보기

                if detectBlock():

                    t1\_flight.rotate(90\*pre\_rotate).wait\_for\_completed()    #다시 돌려서 가기

                else:

                    #좌표 갱신하는거 dxy 바꾸기

                    pre\_rotate = -pre\_rotate

                    if abs(d\_fblr[0] == 1):

                        d\_fblr[0], d\_fblr[1] = d\_fblr[1], pre\_rotate\*d\_fblr[0]

                    else:

                        d\_fblr[0], d\_fblr[1] = -pre\_rotate\*d\_fblr[1], d\_fblr[0]

            t1\_flight.rc(a=0,b=30,c=0,d=0)  # 전방으로 이동

            time.sleep(0.05)

            print("Go ", count, temp)

            print("좌표: ",move\_state)

            move\_state[0] += d\_fblr[0]

            move\_state[1] += d\_fblr[1]

        time.sleep(1)

    # 동작 종류후 착륙

    t1\_flight.land().wait\_for\_completed()

    t1\_drone.close()

def human\_controll():

    """ 키보드 입력을 통한 드론 직접 컨트롤

    방향키: 앞,뒤,좌,우

    y: 이륙, u: 착륙

    w: 위, s: 아래

    """

    init\_key()    # key입력받기 초기화

    flight\_status = False     # 현재 비행 상태인지 확인, 상태에 따라 입력값이 달라짐

    while True:

        if flight\_status:  #한번 날리고 착륙했으면 종료

            break

        #key 입력이 있을시 그 정보 받아오기

        vals = get\_keyboard\_input()

        if vals[4]:  #'y' key가 눌렸을때 이륙

            flight\_status = True  #비행상태 갱신

            takeoffDrone()

            time.sleep(0.05)

            while True:

                vals = get\_keyboard\_input()

                if vals[5]:  #'u' key가 눌렸을때 착륙

                    t1\_flight.land().wait\_for\_completed()

                    t1\_drone.close()

                    break

                print("Go:",vals[0],vals[1],vals[2], sep = " ")

                # 키입력대로 이동, 주행

                t1\_flight.rc(a=vals[0],b=vals[1],c=vals[2],d=vals[3])

                time.sleep(0.05)

def empty(a):

    """Trackbar 생성을 위한 동작이 없는 함수

    Args:

        a (none): 무시

    """

    pass

def camera\_display():

    """camera diplay

    camera를 통해 받아온 frame을 cv2로 display

    """

    global camera\_end

    global area

    t1\_camera.start\_video\_stream(display=False) #카메라 비디오 스트림 시작

    # hsvLower = np.array([165, 100, 50])

    # hsvUpper = np.array([179, 255, 255])

    # cv2.namedWindow("HSV")

    # cv2.resizeWindow("HSV", 640, 240)

    # cv2.createTrackbar("H Min", "HSV", 0 , 179, empty)

    # cv2.createTrackbar("H Max", "HSV", 179 , 179, empty)

    # cv2.createTrackbar("S Min", "HSV", 0 , 255, empty)

    # cv2.createTrackbar("S Max", "HSV", 255 , 255, empty)

    # cv2.createTrackbar("V Min", "HSV", 255 , 255, empty)

    # cv2.createTrackbar("V Max", "HSV", 255 , 255, empty)

    while camera\_end:

        img = t1\_camera.read\_video\_frame(timeout=10)  #프레임 받아오기

        # h\_min = cv2.getTrackbarPos("H Min", "HSV")

        # h\_max = cv2.getTrackbarPos("H Max", "HSV")

        # s\_min = cv2.getTrackbarPos("S Min", "HSV")

        # s\_max = cv2.getTrackbarPos("S Max", "HSV")

        # v\_min = cv2.getTrackbarPos("V Min", "HSV")

        # v\_max = cv2.getTrackbarPos("V Max", "HSV")

        # hsvLower = np.array([h\_min, s\_min, v\_min])

        # hsvUpper = np.array([h\_max, s\_max, v\_max])

        # hsv = cv2.cvtColor(img, cv2.COLOR\_BGR2HSV)

        # hsv\_mask = cv2.inRange(hsv, hsvLower, hsvUpper)

        # result = cv2.bitwise\_and(img, img, mask=hsv\_mask)

        # contours, \_ = cv2.findContours(hsv\_mask, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

        # tempArea = []

        # for contour in contours:

        #     tempArea.append(cv2.contourArea(contour))

        # if len(tempArea) != 0:

        #     area = max(tempArea) // 100 # 특정 색상의 영역 넓이를 area에 저장

        #     print("Area:", area)

        cv2.imshow("Drone Camera", img) #display

        cv2.waitKey(1)  #fps 조절

    t1\_camera.stop\_video\_stream()   #비디오 스트림 종료

def strart\_action(type):

    """비행 동작 실행

    type에 따른 드론 동작 실행

    Args:

        type (int): 비행 조종타입   1: auto, 2: human controll

    """

    global camera\_end

    if type == 1:

        # auto\_pilot thread 생성

        flight\_threading = threading.Thread(target=auto\_pilot)

    elif type == 2:

        # human\_controll thread 생성

        flight\_threading = threading.Thread(target=human\_controll)

    else:

        # Invalid Type

        print("error: 잘못된 타입 선택")

        return

    # 카메라 관련 thread 생성

    camera\_threading = threading.Thread(target=camera\_display)

    #각 스레드 시작

    flight\_threading.start()

    camera\_threading.start()

    # flght thread 종료까지 대기

    flight\_threading.join()

    # camera thread 종료를 위한 변수 변경

    camera\_end = False

    # camera thread 종료까지 대기

    camera\_threading.join()

################## main 실행영역 ##################

#드론, 변수 초기화

try:

    t1\_drone = robot.Drone()    # 드론 객체 생성

    t1\_drone.initialize()   # 드론 초기화

    t1\_flight = t1\_drone.flight     #드론 주행관련 변수

    t1\_led = t1\_drone.led   #드론 LED관련 변수

    t1\_camera = t1\_drone.camera     #드론 카메라관련 변수

    t1\_battery = t1\_drone.battery   #드론 배터리관련 변수

    t1\_sensor = t1\_drone.sensor     #드론 센서관련 변수

    t1\_led.set\_mled\_bright(bright=10)   # 매트릭스 LED 밝기 조절

    # 매트릭스 LED 세팅

    # t1\_led.set\_mled\_boot("0000000000r00r000r0rr0r000r00r00000rr000")

    print("연결 성공")

    print("Current Battery: ", t1\_battery.get\_battery(), "%")

    print(t1\_drone.get\_wifi())

except Exception as e:

    # 드론 연결 실패시 프로그램 종료

    print("연결 실패")

    exit()

#조종 타입 선택

type\_select = int(input("1. 자동비행\n2. 직접 조종\n비행 타입을 선택하세요:"))

strart\_action(type\_select) #타입에 맞는 동작 실행

# while True:

#     print(t1\_drone.get\_height())

#     time.sleep(1)

print("END")