Assignment 3

กลุ่ม 4 Square

6610110214 พีรณัฐ ปถมกุล (กัส)

6610110126 ธัญพิสิษฐ์ แสงส่อง (ปริ้น)

6610110492 ทักษลักษณ์ แย้มมยาสุจริต (อู๋)

6610110475 ณัฐดนัย ชูกูล (เกมส์)

อาจารย์ผู้สอน

อาจารย์ธเนศ เคารพาพงศ์

อาจารย์รัฐชัย วงศ์ธนวิจิต

อาจารย์อนันท์ ชกสุริวงค์

รายงานฉบับนี้เป็นส่วนหนึ่งของการเรียนวิชา 241-251 AI for robot controlling module

สาขาวิชา ปัญญาประดิษฐ์ มหาวิทยาลัย สงขลานครินทร์ ภาคเรียนที่ 1 ปีการศึกษา 2567

คำนำ

การพัฒนาเทคโนโลยีหุ่นยนต์ในปัจจุบันมีความก้าวหน้าอย่างรวดเร็ว โดยเฉพาะอย่างยิ่งในการใช้ระบบเซนเซอร์และกล้องในการแก้ไขปัญหาต่าง ๆ รายงานฉบับนี้จัดทำขึ้นเพื่ออธิบายวิธีการแก้ปัญหาการเดินในเขาวงกต โดยการสร้างหุ่นยนต์อัจฉริยะที่สามารถตรวจจับกำแพงและเก็บข้อมูลตำแหน่งสิ่งของได้อย่างมีประสิทธิภาพ

สารบัญ

บทนำ

รายงานฉบับนี้จัดทำขึ้นเพื่ออธิบายการพัฒนาวิธีการแก้ปัญหาการเดินเพื่อเก็บข้อมูลภายในเขาวงกต โดยใช้เทคโนโลยีระบบเซนเซอร์และกล้องร่วมกับการโปรแกรมหุ่นยนต์อัจฉริยะเพื่อให้หุ่นยนต์สามารถตรวจจับกำแพงและเก็บข้อมูลตำแหน่งสิ่งของได้อย่างแม่นยำ นอกจากนี้ การพัฒนานี้ยังมุ่งหวังที่จะเพิ่มประสิทธิภาพในการเลือกเส้นทางที่ดีที่สุดสำหรับการเดินในสภาพแวดล้อมที่ซับซ้อน โดยการทดสอบและวิเคราะห์ผลการทำงานของหุ่นยนต์ในสถานการณ์จริง นับเป็นก้าวสำคัญในการประยุกต์ใช้เทคโนโลยีหุ่นยนต์ในการแก้ไขปัญหาในโลกแห่งความเป็นจริง

วัตถุประสงค์

- เพื่อให้หุ่นยนต์สามารถเก็บข้อมูลต่างๆในเขาวงกตได้
- เพื่อให้หุ่นยนต์สามารถทำภารกิจในเขาวงกตโดยใช้เวลาน้อยที่สุด

อุปกรณ์

- หุ่นยนต์ Robomaster
- กล้องบน Robomaster
- TOF sensor
- ADC sensor
- Sub_position
- Sub_adtitute
- Visual Studio Code

1.import library

import robomaster

from robomaster import robot, blaster, camera

import math

import matplotlib.pyplot as plt

import time

import cv2

import numpy as np

2.กำหนด grid ,ขอบเขตกำแพง และ ตำแหน่งเริ่มต้น

```
#[North(front), West(left), East(right), South(back),Visited,chicken,thive]
grid = [
    [[2,2,0,0,0,0,0],[2,0,0,0,0,0,0],[2,0,0,0,0,0,0],[2,0,0,0,0,0,0],[2,0,0,0,0,0,0],[2,0,2,0,0,0,0]],
    [[0,2,0,0,0,0,0],[0,0,0,0,0,0,0],[0,0,0,0,0,0,0],[0,0,0,0,0,0,0],[0,0,0,0,0,0,0],[0,0,2,0,0,0,0]],
    [[0,2,0,0,0,0,0],[0,0,0,0,0,0,0],[0,0,0,0,0,0,0],[0,0,0,0,0,0,0],[0,0,0,0,0,0,0],[0,0,2,0,0,0,0]],
    [[0,2,0,0,0,0,0],[0,0,0,0,0,0,0],[0,0,0,0,0,0,0],[0,0,0,0,0,0,0],[0,0,0,0,0,0,0],[0,0,2,0,0,0,0]],
    [[0,2,0,0,0,0,0],[0,0,0,0,0,0,0],[0,0,0,0,0,0,0],[0,0,0,0,0,0,0],[0,0,0,0,0,0,0],[0,0,2,0,0,0,0]],
    [[0,2,0,2,0,0,0],[0,0,0,2,0,0,0],[0,0,0,2,0,0,0],[0,0,0,2,0,0,0],[0,0,0,2,0,0,0],[0,0,2,2,0,0,0]]
]

# start map
row = 0
col = 5
cerrent_po = [row,col]
```

- กำหนดขอบกำแพงไว้เพื่อไม่ให้หุ่นเดินออกจากเขาวงกต
- กำหนดตำแหน่งเริ่มต้น

โดยใช้เป็นพิกัด row ,column โดยเริ่มจากมุมบนขวา เป็นตำแหน่ง 0,0 โดยใน 1 ช่อง จะเก็บข้อมูล

ดังนี้ [North(front), West(left), East(right), South(back),Visited,chicken,thive]

และให้

0 คือ ยังไม่เคยไป

1 คือ เคยไป

2 คือ มีกำแพง หรือขอบแมพ

3. สร้าง **คลาส `Marker` ไว้เก็บตัวแปรของภาพ**

```
class Marker:
    def __init__(self, x, y, w, h):
        self._x = x
        self._y = y
        self._w = w
        self._h = h

    @property
    def pt1(self):
        return int(self._x), int(self._y)
    @property
    def pt2(self):
        return int(self._x + self._w), int(self._y + self._h)
    @property
    def center(self):
        return int(self._x + self._w / 2), int(self._y + self._h / 2)
```

- **วัตถุประสงค์**: แทนที่ตำแหน่งและขนาดของเครื่องหมายที่ตรวจจับได้ (มักเป็นวงกลม)
- **Attributes**
    - _x, _y: ค่าพิกัดของเครื่องหมาย
    - _w, _h: ความกว้างและความสูงของเครื่องหมาย
- **Methods**
    - pt1: คืนค่าพิกัดของมุมซ้ายบนของเครื่องหมาย
    - pt2: คืนค่าพิกัดของมุมขวาล่าง
    - center: คืนค่าพิกัดของจุดศูนย์กลางของเครื่องหมาย

4.ฟังก์ชัน `process_image`

- **วัตถุประสงค์**: ประมวลผลภาพเพื่อค้นหาขอบโดยใช้เทคนิค Canny edge detection และขยายขอบให้เด่นชัดขึ้น
- **ขั้นตอน**:

```
image = cv2.imread(image_path)[250:480,570:720]
```

- อ่านภาพจากไฟล์และตัดเฉพาะส่วนที่สนใจ (crop) โดยเลือกพื้นที่ระหว่างพิกัด y 250-480 และ x 570-720

```
b, g, r = cv2.split(image)
```

- แยกช่องสี BGR ของภาพ

```
b_contrasted = cv2.convertScaleAbs(b, alpha=alpha, beta=beta)
```

- ปรับคอนทราสต์ของช่องสีน้ำเงิน (B) โดยใช้ค่า alpha และ beta

```
edges_b = cv2.Canny(b_contrasted, canny_threshold1, canny_threshold2)
```

- ใช้ Canny edge detection เพื่อตรวจจับขอบในภาพที่ปรับคอนทราสต์แล้ว

```
kernel = np.ones(dilation_kernel_size, np.uint8)
```

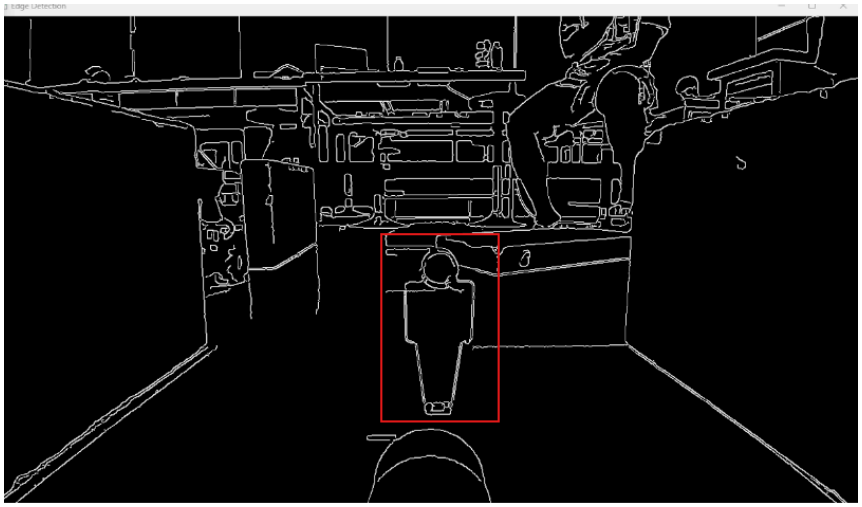
- สร้าง kernel สำหรับการทำ dilation

```
edges_dilated = cv2.dilate(edges_b, kernel,
   iterations=dilation_iterations)
```

- ทำ dilation บนภาพขอบที่ได้จาก Canny เพื่อขยายเส้นขอบให้หนาขึ้น

```
cv2.imwrite(output_path, edges_dilated)
```

- บันทึกภาพที่ผ่านการประมวลผลแล้วลงในไฟล์



5.ฟังก์ชัน `detect_blue_circles` นี้ทำหน้าที่ตรวจจับวงกลมสีฟ้าในภาพ โดยมีขั้นตอนดังนี้:

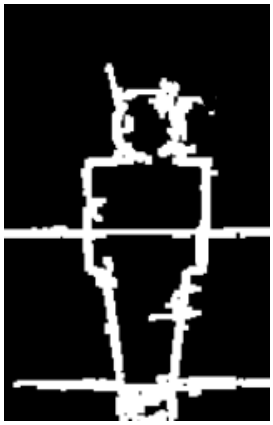
1. `hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)`: แปลงภาพจาก BGR เป็น HSV
2. `mask = cv2.inRange(hsv, lower_blue, upper_blue)`: สร้าง mask สำหรับสีฟ้า
3. `result = cv2.bitwise_and(frame, frame, mask=mask)`: ใช้ mask กับภาพต้นฉบับ
4. `gray = cv2.cvtColor(result, cv2.COLOR_BGR2GRAY)`: แปลงภาพเป็นโทนสีเทา
5. `gray_blurred = cv2.GaussianBlur(gray, (9, 9), 2)`: ทำ Gaussian blur
6. `circles = cv2.HoughCircles(...)`: ใช้ Hough Circle Transform เพื่อตรวจจับวงกลม
7. ถ้าพบวงกลม:
    - วนลูปผ่านแต่ละวงกลมที่พบ
    - ตรวจสอบความเป็นวงกลมด้วยการคำนวณ circularity
    - ถ้า circularity > 0.6:
        - วาดวงกลมและสี่เหลี่ยมรอบวงกลมที่ตรวจพบ
        - สร้างวัตถุ Marker เก็บข้อมูลตำแหน่งและขนาด
8. วาดเส้นกากบาทที่จุดกึ่งกลางภาพ
9. แสดงภาพผลลัพธ์ด้วย `cv2.imshow`
10. ส่งคืนวัตถุ marker ที่เก็บข้อมูลวงกลมที่ตรวจพบ

วัตถุประสงค์: ตรวจจับวงกลมสีฟ้าในเฟรมภาพ

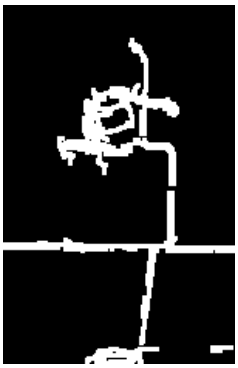
วิธีการ: ใช้การแปลงสี HSV, mask, และ HoughCircles

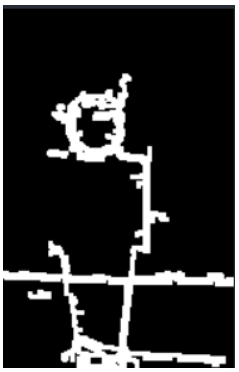
6.ฟังก์ชัน `matches_template()` นี้ใช้สำหรับเปรียบเทียบภาพที่ตรวจจับได้กับภาพต้นแบบ (template) โดยมีขั้นตอนดังนี้:

1. `cv2.imread()`: อ่านภาพต้นแบบ ('template.png') และภาพปัจจุบัน ('current_detection.png')
2. `cv2.ORB_create()`: สร้างตัวตรวจจับ ORB (Oriented FAST and Rotated BRIEF) สำหรับหา keypoints และ descriptors
3. `orb.detectAndCompute()`: ใช้ ORB เพื่อหา keypoints และ descriptors ของทั้งสองภาพ
4. `cv2.BFMatcher()`: สร้าง Brute-Force Matcher สำหรับจับคู่ descriptors
5. `bf.match()`: จับคู่ descriptors ระหว่างสองภาพ
6. `sorted()`: เรียงลำดับ matches ตามระยะห่าง (น้อยไปมาก)
7. คำนวณ similarity_score โดยนับจำนวน matches ที่มีระยะห่างน้อยกว่า 50
8. แสดงคะแนนความคล้ายคลึง
9. ถ้า similarity_score >= 10:
    - สั่งให้ blaster ยิง 3 ครั้ง



Template





ภาพที่เอามาเช็คความเหมือน

7.ฟังก์ชัน `detect_chicken` นี้ใช้สำหรับตรวจจับไก่ในภาพ โดยมีขั้นตอนดังนี้:

1. `cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)`: แปลงภาพจาก BGR เป็น HSV color space
2. กำหนดช่วงสีสำหรับไก่ในพื้นที่ HSV:
    - `lower_chicken = np.array([29, 235, 85])`
    - `upper_chicken = np.array([37, 255, 255])`
3. `cv2.inRange()`: สร้าง mask สำหรับสีของไก่
4. `cv2.findContours()`: หา contours จาก mask ที่ได้
5. ถ้าพบ contours:
    - หา contour ที่มีพื้นที่ใหญ่ที่สุด
    - ใช้ `cv2.boundingRect()` เพื่อหากรอบสี่เหลี่ยมที่ล้อมรอบ contour นั้น
6. ตรวจสอบขนาดและสัดส่วนของกรอบสี่เหลี่ยม:
    - ถ้าความกว้างอยู่ระหว่าง 50-60 หรือ

7. ปรับขนาดกรอบสี่เหลี่ยมให้ใหญ่ขึ้น:
    - เพิ่มความกว้างและความสูง
    - ปรับตำแหน่งของกรอบ
8. วาดกรอบสี่เหลี่ยมรอบไก่ที่ตรวจพบด้วย `cv2.rectangle()`
9. เขียนข้อความแสดงตำแหน่งและขนาดของไก่ด้วย `cv2.putText()`
10. พิมพ์ '>>>>>chick<<<<<<' เพื่อแสดงว่าตรวจพบไก่

8.ฟังก์ชัน `detect()` นี้เป็นฟังก์ชันหลักสำหรับการตรวจจับและตอบสนองต่อวัตถุในภาพ โดยมีขั้นตอนดังนี้:

1. วนลูปอ่านภาพจากกล้องของหุ่นยนต์
2. เรียกใช้ `detect_blue_circles()` เพื่อตรวจจับวงกลมสีฟ้า
3. เรียกใช้ `detect_chicken()` เพื่อตรวจจับไก่
4. ตรวจสอบเงื่อนไขของ marker และ count เพื่อกำหนดค่า check
5. ถ้า check เป็น False:
    - หยุดการเคลื่อนที่ของหุ่นยนต์
    - ถ้าพบ marker:
        - คำนวณค่าความผิดพลาดในแนวแกน x และ y
        - คำนวณความเร็วสำหรับการปรับ gimbal
        - ควบคุม gimbal ให้หันไปทาง marker
        - บันทึกข้อมูลการปรับ gimbal
    - ถ้าขนาดของ marker >= 45:
        - เพิ่มค่า count
        - เปิดไฟ LED ของ blaster ในบางช่วง count
6. เมื่อ count เท่ากับ 10:
    - บันทึกภาพปัจจุบัน
    - เรียกใช้ `process_image()` เพื่อประมวลผลภาพ
7. เมื่อ count เท่ากับ 15:
    - เรียกใช้ `matches_template()` เพื่อเปรียบเทียบภาพกับ template
8. เมื่อ count >= 20:
    - ปรับ gimbal กลับสู่ตำแหน่งกลาง
    - ปิดไฟ LED ของ blaster
    - รีเซ็ตค่า check และ count
    - ออกจากลูป
9. ถ้าไม่เข้าเงื่อนไขใดๆ ให้หยุดการเคลื่อนไหวของ gimbal

9.อ่านค่า TOF Sensor

```
status_tof = None

tof_distance = 0

def tof_data_handler(sub_info):

   global tof_distance, status_tof

   tof_distance = sub_info[0]

   # print(f"ToF distance: {tof_distance} mm")

   if 300 >= tof_distance >= 100:

      status_tof = True

   else:

      status_tof = False
```

วัตถุประสงค์: จัดการข้อมูลจากเซ็นเซอร์ Time of Flight (ToF) การทำงาน:

- รับค่าระยะทางจาก ToF sensor
- กำหนดสถานะ status_tof เป็น True ถ้าระยะทางอยู่ระหว่าง 100-300 มม.

10.อ่านค่า SHARP Sensor

```
def sharp_sen_data():
    global
adc_r,adc_l,adc_r_new,adc_l_new,status_ss_l,status_ss_r
    adc_r = ep_sensor_adaptor.get_adc(id=2, port=2)
    adc_r_cm = (adc_r * 3) / 1023  # process to cm unit
    adc_l = ep_sensor_adaptor.get_adc(id=3, port=1)
    adc_l_cm = (adc_l * 3) / 1023  # process to cm unit
    # print(f"adc_r_cm: {adc_r_cm} volt")
    # print(f"adc_l_cm: {adc_l_cm} volt")
    if adc_r_cm > 1.4:
        adc_r_new = ((adc_r_cm - 4.2) / -0.31)
    elif 1.4 >= adc_r_cm >= 0.6:
        adc_r_new = ((adc_r_cm - 2.03) / -0.07)
```

```
    elif 0 <= adc_r_cm < 0.6:
        adc_r_new = ((adc_r_cm - 0.95) / -0.016)


    if adc_l_cm > 1.4:
        adc_l_new = ((adc_l_cm - 4.2) / -0.31)
    elif 1.4 >= adc_l_cm >= 0.6:
        adc_l_new = ((adc_l_cm - 2.03) / -0.07)
    elif 0 <= adc_l_cm < 0.6:
        adc_l_new = ((adc_l_cm - 0.95) / -0.016)

    if 46 > adc_r_new > 2:
        status_ss_r = True
    else:
        status_ss_r = False


    if 40 > adc_l_new > 2:
        status_ss_l = True
    else:
        status_ss_l = False
```

วัตถุประสงค์: จัดการข้อมูลจากเซ็นเซอร์ Sharp การทำงาน:

- อ่านค่า ADC จากเซ็นเซอร์ Sharp ทั้งซ้ายและขวา
- แปลงค่า ADC เป็นระยะทาง (cm) โดยใช้สมการเฉพาะ โดยเริ่มจากแปลงเป็น volt แล้วกำหนดช่วง เป็น 3ช่วง เพื่อแปลงค่าแต่ละช่วงให้เป็นเซนติเมตร
- กำหนดสถานะ status_ss_r และ status_ss_l ตามระยะทางที่คำนวณได้

  -SHARP ขวา หากอยู่ในช่วง 2 -46 : status_ss_r เป็น TRUE

  -SHARP ซ้าย หากอยู่ในช่วง 2 -40 : status_ss_l เป็น TRUE

  ถ้าไม่ จะเป็น False

11.อ่านค่าตำแหน่งของหุ่น

```
x = 0
y = 0
# ฟังก์ชันสำหรับจัดการตำแหน่งของหุ่นยนต์
def sub_position_handler(position_info):
   global x,y
   x, y, z = position_info
```

วัตถุประสงค์: จัดการข้อมูลตำแหน่งของหุ่นยนต์ การทำงาน:

- **รับค่าตำแหน่ง x, y, z จากหุ่นยนต์**
- **อัพเดตค่าตำแหน่ง x และ y ของหุ่นยนต์**

12.อ่านค่ามุมของหุ่น

```
''' ----- sub_attitude_info_handler ----- '''
yaw = 0


# ฟังก์ชันสำหรับจัดการท่าทางของหุ่นยนต์
def sub_attitude_info_handler(attitude_info):
    global yaw
    yaw, pitch, roll = attitude_info
    # print(f"chassis attitude: yaw:{yaw}")
```

วัตถุประสงค์: จัดการข้อมูลท่าทางของหุ่นยนต์ การทำงาน:

- **รับค่ามุม yaw, pitch, roll จากหุ่นยนต์**
- **อัพเดตค่ามุม yaw ของหุ่นยนต์ จาก callback โดยใช้ความถี่ 10**

13.ปรับหุ่นให้ไม่ชิดกำแพงมากเกินไป

```
def adjust_wall_l():


    if adc_l_new < 40:
        if adc_l_new < 25:
            walk_y = (abs(25 - adc_l_new)/100)-0.035
            # print("Move right")
            ep_chassis.move(x=0, y=walk_y, z=0,
```

```
xy_speed=MAX_SPEED).wait_for_completed()
        elif adc_l_new > 35:
            walk_y = (abs(35 - adc_l_new)/100)-0.035
            # print("Move left")
            ep_chassis.move(x=0, y=-walk_y, z=0,
xy_speed=MAX_SPEED).wait_for_completed()


def adjust_wall_r():


    if adc_r_new < 40:
        if adc_r_new < 25 :
            walk_y = (abs(25 - adc_r_new)/100)-0.035
            # print("Move left")
            ep_chassis.move(x=0, y=-walk_y, z=0,
xy_speed=MAX_SPEED).wait_for_completed()

        elif adc_r_new > 35:
            walk_y = (abs(35 - adc_r_new)/100)-0.035
            # print("Move right")
            ep_chassis.move(x=0, y=walk_y, z=0,
xy_speed=MAX_SPEED).wait_for_completed()


def adjust_wall_f():

    if tof_distance <100:
        walk_x = (abs(100-tof_distance)/1000)+0.02
        ep_chassis.move(x=-walk_x, y=0, z=0,
xy_speed=MAX_SPEED).wait_for_completed()

    if 450>=tof_distance>300:
        walk_x = (abs(tof_distance -300)/1000)+0.02
        ep_chassis.move(x=walk_x, y=0, z=0,
xy_speed=MAX_SPEED).wait_for_completed()
```

วัตถุประสงค์: ปรับตำแหน่งของหุ่นยนต์ให้ไม่ชิดกับกำแพงทั้งสามด้าน
โดยกำหนดค่าชิดจากกำแพง และปรับมาระยะหนึ่ง

a) adjust_wall_lปรับตำแหน่งของหุ่นยนต์ให้ขนานกับกำแพงด้านซ้าย

การทำงาน:

- **ตรวจสอบระยะห่างจากกำแพงด้านซ้าย**
- **เคลื่อนที่ซ้ายหรือขวาเพื่อรักษาระยะห่างที่เหมาะสม**

b) adjust_wall_r ปรับตำแหน่งของหุ่นยนต์ให้ขนานกับกำแพงด้านขวา

การทำงาน:

- ตรวจสอบระยะห่างจากกำแพงด้านขวา
- เคลื่อนที่ซ้ายหรือขวาเพื่อรักษาระยะห่างที่เหมาะสม

c) adjust_wall_f ปรับตำแหน่งของหุ่นยนต์ให้ห่างจากกำแพงด้านหน้าอย่างเหมาะสม

การทำงาน:

- ตรวจสอบระยะห่างจากกำแพงด้านหน้า
- เคลื่อนที่เข้าหาหรือถอยห่างจากกำแพงเพื่อรักษาระยะห่างที่เหมาะสม

14. การเคลื่อนที่ทั้งหมด

```
''' ----- movement ----- '''
def move_stop():
    ep_chassis.drive_speed(x=0, y=0, z=0, timeout=0.75)
    time.sleep(0.7)


def move_forward():
    print("Drive forward")
    ep_chassis.move(x=0.57, y=0, z=0,
xy_speed=MAX_SPEED).wait_for_completed()
    ep_gimbal.recenter(pitch_speed=400,
yaw_speed=400).wait_for_completed()



def turn_back():
    print("Turn Back")

    ep_chassis.move(x=0, y=0, z=180,
xy_speed=MAX_SPEED).wait_for_completed()
    ep_gimbal.recenter(pitch_speed=400,
yaw_speed=400).wait_for_completed()



def turn_left():
    print("Turn Left")

    ep_chassis.move(x=0, y=0, z=90,
xy_speed=MAX_SPEED).wait_for_completed()
    ep_gimbal.recenter(pitch_speed=400,
```

```
yaw_speed=400).wait_for_completed()

def turn_right():
    print('Turn Right')

    ep_chassis.move(x=0, y=0, z=-90,
xy_speed=MAX_SPEED).wait_for_completed()
    ep_gimbal.recenter(pitch_speed=400,
yaw_speed=400).wait_for_completed()
```

ควบคุมการเคลื่อนที่พื้นฐานของหุ่นยนต์ การทำงาน:

- **ใช้คำสั่ง ep_chassis.move() ep_chassis.drive_speed() เพื่อควบคุมการเคลื่อนที่**
- **ep_chassis.drive_speed() เพื่อหยุดรถ**
- **ใช้ ep_gimbal.recenter() เพื่อปรับตำแหน่งกล้องกลับสู่ตำแหน่งกลาง(ทิศหน้ารถ)**
- **ทำเป็น Function เพื่อใว้เรียกใช้งานได้ง่าย และสะดวก**

15.กำหนดทิศทางปัจจุบันของหุ่นยนต์

```
''' ----- DirectionFacing ----- '''
robo_status_now = None
def getDirectionFacing():
   global robo_status_now
   degrees = yaw
   if -45 <= degrees < 0 or 45>=degrees >= 0:
      robo_status_now = 'N'
   if 45 < degrees <= 135:
      robo_status_now = 'E'
   if 135 < degrees <=180 or -180<= degrees <-135 :
      robo_status_now = 'S'
   if -135 <= degrees < -45:
      robo_status_now = 'W'
```

วัตถุประสงค์:

- **กำหนดทิศทางปัจจุบันของหุ่นยนต์**

การทำงาน:

- **ใช้ค่า yaw เพื่อกำหนดทิศทาง robo_status_now = (N, E, S, W)โดยกำหนดเป็นช่วงองศาของหุ่น เพื่อนำไปใช้ใน function อื่นๆ**

16.อัพเดตตำแหน่งและข้อมูลกำแพงในแผนที่เสมือน

```
''' ----- Update_Maze ----- '''
status_logic = None
def update_position():
    global cerrent_po,status_logic
    getDirectionFacing()
    logic()
    if robo_status_now =='N':
        if status_logic == 'move_forward':
            grid[cerrent_po[0]][cerrent_po[1]][2] = 2 #grid[cerrent_po[0]][cerrent_po[1]] have right
wall
            grid[cerrent_po[0]-1][cerrent_po[1]][4] = 1
            # grid[cerrent_po[0]+1][cerrent_po[1]][3] = 1




            cerrent_po = [cerrent_po[0]-1,cerrent_po[1]]

        if status_logic == 'turn right': #turn right and move forward
            # grid[cerrent_po[0]][cerrent_po[1]][2] = 2 #grid[cerrent_po[0]][cerrent_po[1]] have right
wall
            grid[cerrent_po[0]][cerrent_po[1]+1][4] = 1

            cerrent_po = [cerrent_po[0],cerrent_po[1]+1]

        if status_logic == 'turn back':
            grid[cerrent_po[0]][cerrent_po[1]][0] = 2 #grid[cerrent_po[0]][cerrent_po[1]] have front
wall
            grid[cerrent_po[0]][cerrent_po[1]][1] = 2 #grid[cerrent_po[0]][cerrent_po[1]] have left
wall
            grid[cerrent_po[0]][cerrent_po[1]][2] = 2 #grid[cerrent_po[0]][cerrent_po[1]] have right
wall



            grid[cerrent_po[0]+1][cerrent_po[1]][4] = 1
            # grid[cerrent_po[0]+1][cerrent_po[1]][3] = 1


            cerrent_po = [cerrent_po[0]+1,cerrent_po[1]]
```

```
        if status_logic == 'turn left':
            grid[cerrent_po[0]][cerrent_po[1]][0] = 2 #grid[cerrent_po[0]][cerrent_po[1]] have front
wall
            grid[cerrent_po[0]][cerrent_po[1]][2] = 2 #grid[cerrent_po[0]][cerrent_po[1]] have right
wall
            grid[cerrent_po[0]][cerrent_po[1]-1][4] = 1
            # grid[cerrent_po[0]+1][cerrent_po[1]][3] = 1
            cerrent_po = [cerrent_po[0],cerrent_po[1]-1]


    if robo_status_now =='E':
        if status_logic == 'move_forward':
            grid[cerrent_po[0]][cerrent_po[1]][0] = 2 #grid[cerrent_po[0]][cerrent_po[1]] have front
wall
            grid[cerrent_po[0]][cerrent_po[1]+1][4] = 1
            # grid[cerrent_po[0]+1][cerrent_po[1]][3] = 1




            cerrent_po = [cerrent_po[0],cerrent_po[1]+1]

        if status_logic == 'turn right': #turn right and move forward
            # grid[cerrent_po[0]][cerrent_po[1]][2] = 2 #grid[cerrent_po[0]][cerrent_po[1]] have right
wall
            grid[cerrent_po[0]+1][cerrent_po[1]][4] = 1

            cerrent_po = [cerrent_po[0]+1,cerrent_po[1]]

        if status_logic == 'turn back':
            grid[cerrent_po[0]][cerrent_po[1]][0] = 2 #grid[cerrent_po[0]][cerrent_po[1]] have front
wall
            grid[cerrent_po[0]][cerrent_po[1]][2] = 2 #grid[cerrent_po[0]][cerrent_po[1]] have right
wall
            grid[cerrent_po[0]][cerrent_po[1]][3] = 2 #grid[cerrent_po[0]][cerrent_po[1]] have back
wall



            grid[cerrent_po[0]][cerrent_po[1]-1][4] = 1
            # grid[cerrent_po[0]+1][cerrent_po[1]][3] = 1


            cerrent_po = [cerrent_po[0],cerrent_po[1]-1]
```

```
        if status_logic == 'turn left':
            grid[cerrent_po[0]][cerrent_po[1]][2] = 2 #grid[cerrent_po[0]][cerrent_po[1]] have right
wall
            grid[cerrent_po[0]][cerrent_po[1]][3] = 2 #grid[cerrent_po[0]][cerrent_po[1]] have back
wall
            grid[cerrent_po[0]-1][cerrent_po[1]][4] = 1
            # grid[cerrent_po[0]+1][cerrent_po[1]][3] = 1
            cerrent_po = [cerrent_po[0]-1,cerrent_po[1]]


    if robo_status_now =='S':
        if status_logic == 'move_forward':
            grid[cerrent_po[0]][cerrent_po[1]][1] = 2 #grid[cerrent_po[0]][cerrent_po[1]] have left
wall
            grid[cerrent_po[0]+1][cerrent_po[1]][4] = 1
            # grid[cerrent_po[0]+1][cerrent_po[1]][3] = 1




            cerrent_po = [cerrent_po[0]+1,cerrent_po[1]]

        if status_logic == 'turn right': #turn right and move forward
            # grid[cerrent_po[0]][cerrent_po[1]][2] = 2 #grid[cerrent_po[0]][cerrent_po[1]] have right
wall
            grid[cerrent_po[0]][cerrent_po[1]-1][4] = 1

            cerrent_po = [cerrent_po[0],cerrent_po[1]-1]

        if status_logic == 'turn back':

            grid[cerrent_po[0]][cerrent_po[1]][1] = 2 #grid[cerrent_po[0]][cerrent_po[1]] have left
wall
            grid[cerrent_po[0]][cerrent_po[1]][2] = 2 #grid[cerrent_po[0]][cerrent_po[1]] have right
wall
            grid[cerrent_po[0]][cerrent_po[1]][3] = 2 #grid[cerrent_po[0]][cerrent_po[1]] have back
wall



            grid[cerrent_po[0]+1][cerrent_po[1]][4] = 1
            # grid[cerrent_po[0]+1][cerrent_po[1]][3] = 1
```

```
            cerrent_po = [cerrent_po[0]+1,cerrent_po[1]]


        if status_logic == 'turn left':
            grid[cerrent_po[0]][cerrent_po[1]][1] = 2 #grid[cerrent_po[0]][cerrent_po[1]] have left
wall
            grid[cerrent_po[0]][cerrent_po[1]][3] = 2 #grid[cerrent_po[0]][cerrent_po[1]] have back
wall
            grid[cerrent_po[0]][cerrent_po[1]+1][4] = 1
            # grid[cerrent_po[0]+1][cerrent_po[1]][3] = 1
            cerrent_po = [cerrent_po[0],cerrent_po[1]+1]


    if robo_status_now =='W':
        if status_logic == 'move_forward':
            grid[cerrent_po[0]][cerrent_po[1]][0] = 2 #grid[cerrent_po[0]][cerrent_po[1]] have back
wall
            grid[cerrent_po[0]][cerrent_po[1]-1][4] = 1
            # grid[cerrent_po[0]+1][cerrent_po[1]][3] = 1




            cerrent_po = [cerrent_po[0],cerrent_po[1]-1]

        if status_logic == 'turn right': #turn right and move forward
            # grid[cerrent_po[0]][cerrent_po[1]][2] = 2 #grid[cerrent_po[0]][cerrent_po[1]] have right
wall
            grid[cerrent_po[0]-1][cerrent_po[1]][4] = 1

            cerrent_po = [cerrent_po[0]-1,cerrent_po[1]]

        if status_logic == 'turn back':
            grid[cerrent_po[0]][cerrent_po[1]][0] = 2 #grid[cerrent_po[0]][cerrent_po[1]] have front
wall
            grid[cerrent_po[0]][cerrent_po[1]][1] = 2 #grid[cerrent_po[0]][cerrent_po[1]] have left
wall
            grid[cerrent_po[0]][cerrent_po[1]][3] = 2 #grid[cerrent_po[0]][cerrent_po[1]] have back
wall
```

```
            grid[cerrent_po[0]][cerrent_po[1]+1][4] = 1
            # grid[cerrent_po[0]+1][cerrent_po[1]][3] = 1



            cerrent_po = [cerrent_po[0],cerrent_po[1]+1]



        if status_logic == 'turn left':
            grid[cerrent_po[0]][cerrent_po[1]][1] = 2 #grid[cerrent_po[0]][cerrent_po[1]] have left
wall
            grid[cerrent_po[0]][cerrent_po[1]][0] = 2 #grid[cerrent_po[0]][cerrent_po[1]] have front
wall
            grid[cerrent_po[0]+1][cerrent_po[1]][4] = 1
            # grid[cerrent_po[0]+1][cerrent_po[1]][3] = 1
            cerrent_po = [cerrent_po[0]+1,cerrent_po[1]]
```

การทำงาน:

- **เรียกใช้** getDirectionFacing() **และ** logic()
- update_position(): **ฟังก์ชันนี้จะทำหน้าที่อัปเดตตำแหน่งของหุ่นยนต์ในกริด โดยพิจารณาทิศทางที่หุ่นยนต์หันหน้า (robo_status_now) และอัปเดตสถานะของผนังในกริดตามข้อมูลเซ็นเซอร์**
- **ในแต่ละทิศทาง หุ่นยนต์จะอัปเดตข้อมูลกริดที่เกี่ยวข้อง เช่น ถ้าหุ่นยนต์หันไปทางเหนือ (N) และตรวจพบผนังด้านหน้า จะทำการอัปเดตผนังในกริดว่าเป็นสิ่งกีดขวาง**
- **ปรับปรุงตำแหน่งปัจจุบันของหุ่นยนต์ในแผนที่**

17. Check และ Update กำแพงจาก TOF Sensor

```
 status_logic = None
def check_tof_wall(tof_now):
    global cerrent_po,status_logic
    if tof_now ==True:


        if robo_status_now =='N':
            grid[cerrent_po[0]][cerrent_po[1]][0] = 2 #grid[cerrent_po[0]][cerrent_po[1]] have west wall


        if robo_status_now =='E':
            grid[cerrent_po[0]][cerrent_po[1]][2] = 2 #grid[cerrent_po[0]][cerrent_po[1]] have north wall
```

```
        if robo_status_now =='S':
            grid[cerrent_po[0]][cerrent_po[1]][3] = 2 #grid[cerrent_po[0]][cerrent_po[1]] have east wall


        if robo_status_now =='W':
            grid[cerrent_po[0]][cerrent_po[1]][1] = 2 #grid[cerrent_po[0]][cerrent_po[1]] have south wall


def check_left_wall(sharp_l_now):
    global cerrent_po,status_logic
    if sharp_l_now ==True:


        if robo_status_now =='N':
            grid[cerrent_po[0]][cerrent_po[1]][1] = 2 #grid[cerrent_po[0]][cerrent_po[1]] have west wall


        if robo_status_now =='E':
            grid[cerrent_po[0]][cerrent_po[1]][0] = 2 #grid[cerrent_po[0]][cerrent_po[1]] have north wall


        if robo_status_now =='S':
            grid[cerrent_po[0]][cerrent_po[1]][2] = 2 #grid[cerrent_po[0]][cerrent_po[1]] have east wall


        if robo_status_now =='W':
            grid[cerrent_po[0]][cerrent_po[1]][3] = 2 #grid[cerrent_po[0]][cerrent_po[1]] have south wall


def check_right_wall(sharp_r_now):
    global cerrent_po,status_logic
    if sharp_r_now == True:
        if robo_status_now =='N':
            grid[cerrent_po[0]][cerrent_po[1]][2] = 2 #grid[cerrent_po[0]][cerrent_po[1]] have east wall


        if robo_status_now =='E':
            grid[cerrent_po[0]][cerrent_po[1]][3] = 2 #grid[cerrent_po[0]][cerrent_po[1]] have back wall


        if robo_status_now =='S':
            grid[cerrent_po[0]][cerrent_po[1]][1] = 2 #grid[cerrent_po[0]][cerrent_po[1]] have west wall


        if robo_status_now =='W':
            grid[cerrent_po[0]][cerrent_po[1]][0] = 2 #grid[cerrent_po[0]][cerrent_po[1]] have front wall
```

วัตถุประสงค์:

- **check_tof_wall(), check_left_wall(), และ check_right_wall(): ฟังก์ชันเหล่านี้จะทำการตรวจสอบสิ่งกีดขวางทางด้านหน้า ด้านซ้าย และด้านขวาของหุ่นยนต์ตามข้อมูลจากเซ็นเซอร์ที่กำหนด (TOF) และทำการอัปเดตกริดว่าเจอผนังที่จุดใดบ้าง**

การทำงาน:

- **อัพเดตค่าในโครงสร้างข้อมูล grid ตามข้อมูลจากเซ็นเซอร์ และ อัพเดตกำแพงบางกำแพง**

18. ตรวจสอบสถานะของกำแพงรอบๆ ตำแหน่งปัจจุบัน

```
f_wall = None
l_wall = None
r_wall = None
def check_2_wall():
    global f_wall,l_wall,r_wall
    if robo_status_now =='N':
        if grid[cerrent_po[0]][cerrent_po[1]][0] == 2:
            f_wall = True
        else:
            f_wall = False

        if grid[cerrent_po[0]][cerrent_po[1]][1] == 2:
            l_wall = True
        else:
            l_wall = False


        if grid[cerrent_po[0]][cerrent_po[1]][2] == 2:
            r_wall = True
        else:
            r_wall = False


        if grid[cerrent_po[0]][cerrent_po[1]][2] == 2:
            b_wall = True
        else:
            b_wall = False


    if robo_status_now =='E':
        if grid[cerrent_po[0]][cerrent_po[1]][0] == 2:
            l_wall = True

        else:
```

```
        l_wall = False


    if grid[cerrent_po[0]][cerrent_po[1]][2] == 2:
        f_wall = True

    else:
        f_wall = False

    if grid[cerrent_po[0]][cerrent_po[1]][3] == 2:
        r_wall = True

    else:
        r_wall = False




if robo_status_now =='S':

    if grid[cerrent_po[0]][cerrent_po[1]][2] == 2:
        r_wall = True

    else:
        r_wall = False


    if grid[cerrent_po[0]][cerrent_po[1]][1] == 2:
        l_wall = True

    else:
        l_wall = False


    if grid[cerrent_po[0]][cerrent_po[1]][3] == 2:
        f_wall = True

    else:
        f_wall = False


if robo_status_now =='W':
    if grid[cerrent_po[0]][cerrent_po[1]][0] == 2:
```

```
        r_wall = True

    else:
        r_wall = False

    if grid[cerrent_po[0]][cerrent_po[1]][1] == 2:
        f_wall = True

    else:
        f_wall = False


    if grid[cerrent_po[0]][cerrent_po[1]][3] == 2:
        l_wall = True

    else:
        l_wall = False
```

การทำงาน:

- **อัพเดตตัวแปรสถานะกำแพงด้านหน้า ซ้าย ขวา ตามทิศทางปัจจุบันของหุ่นยนต์ เพื่อตรวจสอบว่าในตำแหน่งนี้มีกำแพงตรงไหนบ้าง**

19.ฟังก์ชันตรรกะการตัดสินใจในการเดิน หรือการเลือกเส้นทาง

```
''' ----- Logic ----- '''
def logic():
    global status_logic, check
    ep_gimbal.moveto(pitch=0, yaw=-90, pitch_speed=100,
yaw_speed=100).wait_for_completed()
    sl = tof_distance
    ep_gimbal.moveto(pitch=0, yaw= 0, pitch_speed=100,
yaw_speed=100).wait_for_completed()
    tof = tof_distance
    ep_gimbal.moveto(pitch=0, yaw= 90, pitch_speed=100,
yaw_speed=100).wait_for_completed()
    sr = tof_distance
    ep_gimbal.recenter(pitch_speed=400, yaw_speed=400).wait_for_completed()
```

```
if sl <400:
    status_sl = True
else:
    status_sl = False

if sr <400:
    status_sr = True

else:
    status_sr = False

if tof <400:
    status_of_tof = True

else:
    status_of_tof = False



check_all_wall(status_of_tof,status_sl,status_sr)


check_2_wall()
if f_wall == False and r_wall == True:
    ep_chassis.drive_wheels(w1=0, w2=0, w3=0, w4=0)
    time.sleep(0.2)
    adjust_wall()
    detect()
    if check:
        move_forward()
    status_logic = 'move_forward'

elif r_wall == False:
    ep_chassis.drive_wheels(w1=0, w2=0, w3=0, w4=0)
    time.sleep(0.2)
    adjust_wall()
    detect()
    if check:
        turn_right()
    detect()
    if check:
        move_forward()
    status_logic = 'turn right'
```

```
elif f_wall == True and r_wall == True and l_wall == True:
    ep_chassis.drive_wheels(w1=0, w2=0, w3=0, w4=0)
    time.sleep(0.2)
    adjust_wall()
    detect()
    if check:
        turn_back()
    detect()
    if check:
        move_forward()
        status_logic = 'turn back'



elif f_wall == True and r_wall == True:
    ep_chassis.drive_wheels(w1=0, w2=0, w3=0, w4=0)
    time.sleep(0.2)
    adjust_wall()


    detect()
    if check:
        turn_left()
    detect()
    if check:
        move_forward()
    status_logic = 'turn left'
```

วัตถุประสงค์: ตัดสินใจเส้นทางการเคลื่อนที่ของหุ่นยนต์ การทำงาน:

- **ตรวจสอบสภาพแวดล้อมด้วยการหมุน gimbal และอ่านค่าจากเซ็นเซอร์**
- **เรียกใช้** check_all_wall() **และ** check_2_wall()
- **ตัดสินใจเคลื่อนที่ตามลำดับความสำคัญ: เดินหน้า, เลี้ยวขวา, ถอยหลัง, เลี้ยวซ้าย**
- **เรียกใช้ฟังก์ชัน** detect() **เพื่อตรวจจับวัตถุ**
- **ปรับตำแหน่งด้วย** adjust_wall() **ก่อนการเคลื่อนที่**

20.ฟังก์ชันแสดงผล

```
''' ----- Display map ----- '''
def print_pretty_grid(grid):

    for row in range(6):
        # Print top walls
        top_line = "|"
        mid_line = "|"
        bot_line = "|"
        for col in range(6):
            cell = grid[row][col]
            north_wall = cell[0]
            west_wall = cell[1]
            east_wall = cell[2]
            south_wall = cell[3]

            # Top wall (N)
            top_line += f" {'_' if north_wall == 2 else ' ' }   "

            # Middle line with walls (W to E)
            middle = ' V ' if cell[4] == 1 else ' ? '  # Use V for visited cells, ? for unvisited
            mid_line += f"{'|' if west_wall == 2 else ' '}{middle}{'|' if east_wall == 2 else ' '}"

            # Bottom wall (S)
            bot_line += f" {'_' if south_wall == 2 else ' ' }   "

        print(top_line + "|")
        print(mid_line + "|")
        print(bot_line + "|")
        print("|\t\t\t\t\t|")  # spacer line
```

วัตถุประสงค์: แสดงแผนที่เสมือนในรูปแบบที่อ่านง่าย การทำงาน:

- วนลูปผ่านทุกช่องในแผนที่
- สร้างการแสดงผลแผนที่โดยใช้สัญลักษณ์ต่างๆ แทนกำแพงและพื้นที่ที่สำรวจแล้ว

21.ตรวจสอบว่าครบทุกช่องหรือยัง

```
def check_all_cells_visited(grid):
    for row in range(6):
        for col in range(6):
            if grid[row][col][4] == 0:  # If any cell is not visited
                return False
    return True
```

วัตถุประสงค์: ตรวจสอบว่าได้สำรวจครบทุกช่องในแผนที่หรือยัง
การทำงาน: วนลูปตรวจสอบทุกช่องในแผนที่ ถ้าพบช่องที่ยังไม่ได้สำรวจจะคืนค่า False

22.ส่วนหลักในการทำงาน

```
if __name__ == "__main__":
    ep_robot = robot.Robot()
    print("Initializing robot...")
    ep_robot.initialize(conn_type="ap")


    ep_sensor = ep_robot.sensor
    ep_chassis = ep_robot.chassis
    ep_gimbal = ep_robot.gimbal
    ep_blaster = ep_robot.blaster
    ep_camera = ep_robot.camera
    ep_sensor_adaptor = ep_robot.sensor_adaptor
    time.sleep(2)


    center_x = 1280 / 2
    center_y = 720 / 2


    p = 0.36
    i = p / (0.7 / 2)
    d = p * (0.7 / 8)
    # p = 0.4705
    # i = 1.1192
    # d = 0.0494


    accumulate_err_x = 0
    accumulate_err_y = 0
    data_pitch_yaw = []
    prev_time = time.time()


    check = True
    count = 0
```

```
ep_chassis.sub_position(freq=10, callback=sub_position_handler)
ep_sensor.sub_distance(freq=10, callback=tof_data_handler)
ep_chassis.sub_attitude(freq=10, callback=sub_attitude_info_handler)
ep_camera.start_video_stream(display=False, resolution=camera.STREAM_720P)



# ปรับตำแหน่ง Gimbal ให้ตรงศูนย์
ep_gimbal.recenter(pitch_speed=400, yaw_speed=400).wait_for_completed()
adjust_wall()
# adjust_all_walls()


grid[cerrent_po[0]][cerrent_po[1]][4] = 1


try:
    while True:
        if tof_distance is None or adc_l is None or adc_r is None:
            print("Waiting for sensor data...")
            time.sleep(1)
            continue


        maze_complete = False
        while not maze_complete:
            update_position()
            # logic()
            print_pretty_grid(grid)
            # print(grid)

            # Check if all cells have been visited
            if check_all_cells_visited(grid):
                print("Maze exploration complete! All cells have been visited.")
                maze_complete = True
                # Stop the robot
                ep_chassis.drive_wheels(w1=0, w2=0, w3=0, w4=0)
                break

except KeyboardInterrupt:
    print("Program stopped by user")
except Exception as e:
    print(f"An error occurred: {e}")
finally:
    print("Cleaning up...")
```

```
    ep_blaster.set_led(brightness=255, effect=blaster.LED_OFF)
    ep_chassis.unsub_position()
    ep_chassis.unsub_attitude()
    ep_sensor.unsub_distance()
    ep_chassis.drive_speed(x=0, y=0, z=0)
    ep_robot.close()
    print("Program ended...")
```

โปรแกรมหลักทำงานในลูป while ที่:

- รอให้ข้อมูลจากเซ็นเซอร์พร้อม
- เรียกใช้ update_position() เพื่ออัพเดตแผนที่และตำแหน่ง
- แสดงแผนที่ด้วย print_pretty_grid()
- ตรวจสอบว่าสำรวจครบทุกช่องหรือยังด้วย check_all_cells_visited()
- หยุดการทำงานเมื่อสำรวจครบทุกช่อง
- มีการใช้ try-except เพื่อจัดการข้อผิดพลาดที่อาจเกิดขึ้น
- ในส่วน finally มีการทำความสะอาด เช่น ปิดไฟ LED, ยกเลิกการ subscribe ข้อมูลต่างๆ, และปิดการเชื่อมต่อกับหุ่นยนต์

ปัญหาที่พบ
1.Sharp_senser ไม่ไกลพอในระยะ 4 กระเบื้องทำให้ต้องใช้ TOF เข็กกำแพงแทน
2.เกิดปัญหา time out ทำให้ไม่สามารถ run หุ่นได้
3.logic ไม่สามารถแก้ไขปัญหา deadlockได้
4.เวลา detect จะค้างบ้าง และ ประมวลผลไม่ทันทำให้ เกิด time out