MSc Intelligent Systems

Mobile Robots

Coursework

Thawatchai Sangthep

P2681054

Abstract

This project deals with implementing a robot to do the missions. The missions are five missions. Firstly, the robot must find the center of the room and go to this location. Secondly, the robot goes out of the room and without crashing the wall. Thirdly, the robot explorers the room by following the wall. Fourthly, the robot must find the beacon and stop nearly the beacon. Lastly, the robot must leave the beacon and go to the center of the room. The learning outcomes that are assessed by this coursework are. We should understand the subject-specific issues relating to programming mobile robots, be able to program using the basic architectures to control robots, be able to program to execute navigation and sensor data analysis, and actuator control for mobile robots. The Python API and CoppeliaSim are tools to control the robot explorer under the conditions. The testing was given a good experience and result that has high values.

Contents

List of Figures

1. Introduction

This paper presents a mission. There are many tasks on the coursework. The robot must follow the mission and do it completely under the condition. The mission has a composition. Firstly, the mission wants the robot to search the center of the room. Secondly, the robot must run out of the small room and without crashing the wall. Thirdly, the robot runs around the room and keeps data on a map. Fourthly, the robot must run to find a beacon and stop. Lastly, after the robot meets the beacon, the robot goes back to the center of the room. The tools that we use CoppeliaSim to visualization, the Python is language to control the robot. So, we have to understand every flow and process to get the achievement.

1.1 Purpose

The purpose is written about when the missions appear how to solve them. We have known the basics of the control mechanism and using PID controller from the last work. We have to use all knowledge and estimate the process to get a possible solution. The mission can be split into five steps. Firstly, we developed a function that permits the Pioneer 3DX mobile robot to move about randomly inside the space under simulation till it can implement and calculate the middle of the room. Secondly, when the robot detected the middle of the room, how the robot does move out of the room and without crashing the wall. Thirdly, the robot explorer and follows the wall how to keep data on the map. Fourthly, how to control the robot to find a beacon and stop with it. At last, when the robot detected the beacon how to control the robot go back to the middle of the room that keeping the location. The missions challenged to design, analysis and implementation.

1.2 Environment

The map is created of the wall that is the basic structure of the wall but it has a little small room in the center, on the top right of the map has tower we call it is a beacon. The mapping developed are to be seen here as Figure 1:

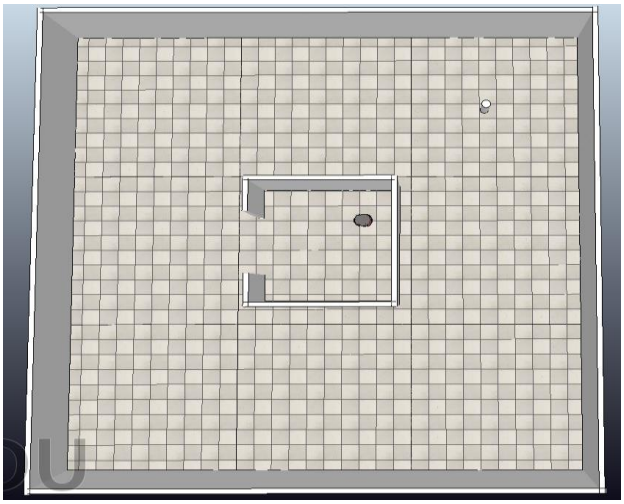


Figure 1: basic structure of the map

2. Architecture Design

The robot followed the missions while it thinks smart decisions based on the environment. The Python has been controlled by the Pioneer 3DX, we are using all sensors to control them.

To understand more on the process the below flow chart is created

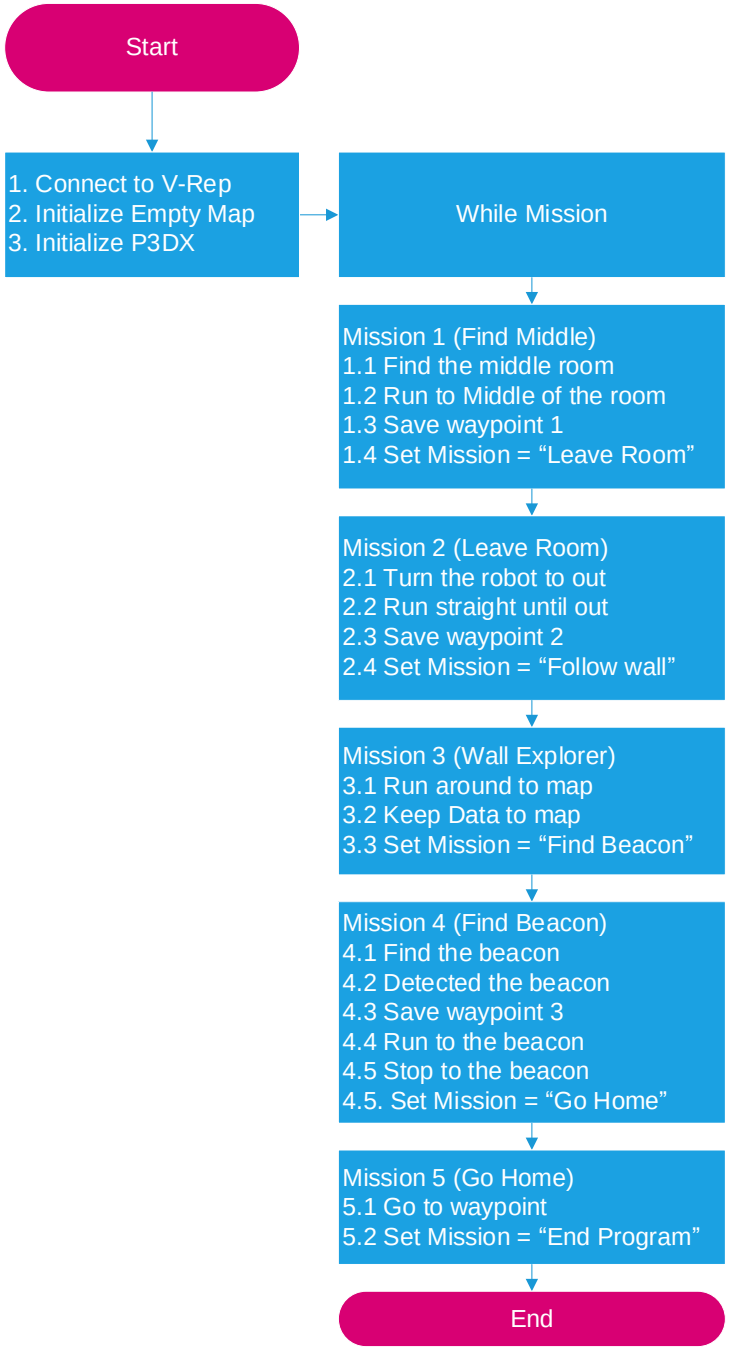


Figure 2: Flow chart of the coursework

2.1 System Description

2.1.1 The robot

Pioneer 3DX Mobile Robot is a small lightweight two-wheel two-motor differential drive robot ideal for indoor laboratory or classroom use. The robot is surrounded by SONAR sensors, on battery, wheel encoders, a microcontroller with ARCOS firmware and the Pioneer SDK advanced mobile robotics software development package. It can be used with a combination of various sensors such as video cameras, ultrasonic sensors, gyroscopes, etc. So, it is suitable for research and application involving mapping, teleoperation, localization, autonomous navigation, etc. [1]

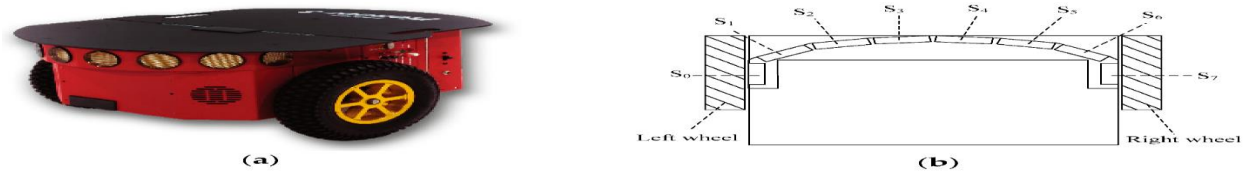


Figure 3: Robot 3DX and Sensors

2.1.2 The Program simulator

A robot simulator CoppeliaSim (V-REP). It is an integrated development environment based on a distributed control architecture, where each object/model can be individually controlled via an embedded script, a plugin, or remote API client. It can be controlled by using C/C++, Python, Java, Lua, MATLAB, or Octave. [2]

Figure 4: CoppeliaSim from the creators of V-REP

2.1.3 Python Script

We will be using Python language to continue the assignment. Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Python is simple, easy to learn syntax that emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. So, the debugger is written in Python itself, testifying to Python's introspective power. [3]

Figure 5: Python language

2.1.4 PID Controller

A PID Controller stands for proportional integral derivative and it is a feedback method based device used to control different process variables like pressure, flow, temperature, and speed in industrial applications. In this controller, a control loop feedback device is used to regulate all the process variables. Several types of control are used to drive a system in the direction of an objective location otherwise level. It is used to maintain the real output when comparing the robot to where it should be to where it is. The process of setting the optimal gains for PID to get an ideal response from a control system is called **tuning**. These are different methods of tuning of which the "guess and check". [4]

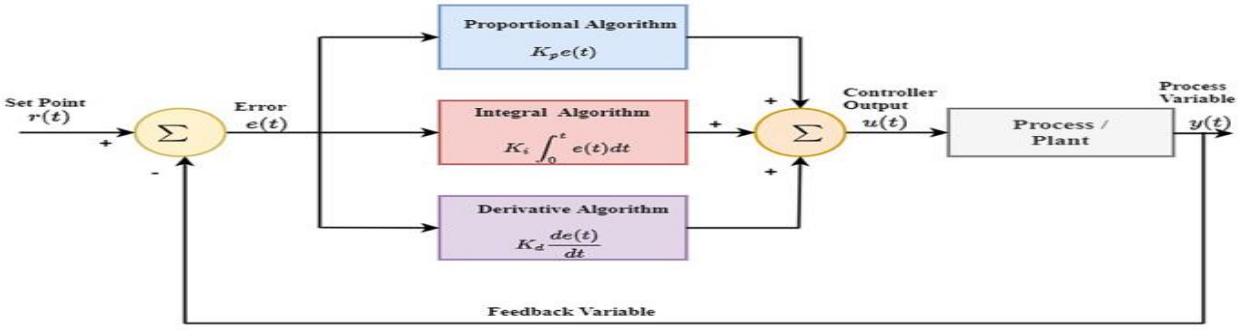


Figure 6: Working of PID controller

- Proportional part (P) = Kp * e(t), Kp is the gain of the proportional controller and e(t) is the error at time t.
- Integral part (I) = Ki * ∫ e(r) * d(t), Ki is the gain of the integral controller, e is the error and t is the integration variable (time interval)
- Derivative part (D) = Kd * de(t) / dt, Kd is the gain of the derivative controller, e is the error and t is the time.

3. Behavioral design

3.1 Controlling Pioneer 3DX

Pioneer P3DX is a form of a robot to controls the wheel of the robot. It has been affected by the code that we wrote. The condition of the wheels is simple. We can separate such as run, turn, using sensor and etc.

3.2 Design for the movement of the robot (run)

If Speed Motor Left = Right && >= 1 : The robot run “Straight Forward”

If Speed Motor Left = Right && <= -1 : The robot run “Straight Backward”

```
def move(self, velocity):
    # velocity < 0 = reverse
    # velocity > 0 = forward
    res = sim.simxSetJointTargetVelocity(clientID, self.leftMotor,
    velocity, sim.simx_opmode_blocking)
    res = sim.simxSetJointTargetVelocity(clientID, self.rightMotor,
    velocity, sim.simx_opmode_blocking)
```

3.3 Design for the movement of the robot (turn)

If Speed Motor Left > Right : The robot turn “Right”

If Speed Motor Left < Right : The robot turn “Left”

```
def turn(self, turnVelocity):
    # turnVelocity < 0 = trun left
    # turnVelocity > 0 = turn right
    res = sim.simxSetJointTargetVelocity(clientID, self.leftMotor,
    turnVelocity, sim.simx_opmode_blocking)
    res = sim.simxSetJointTargetVelocity(clientID, self.rightMotor,
    turnVelocity, sim.simx_opmode_blocking)
```

3.4 Design using all ultrasonic sensors

```
# Setup Sonars
self.sonarHandles = []
text = 'Pioneer_p3dx_ultrasonicSensor'

for i in range(1,17):
    sonarText = text + str(i)
    res, tempHandle = sim.simxGetObjectHandle(clientID, sonarText, sim.simx_opmode_blocking)
    self.sonarHandles.append(tempHandle)
```

4. Experimental design

4.1 Configurator sensor

The Pioneer 3DX has been a configurator to make it effective to detect the wall and open the vision when detected. So, all sensors can see in the simulator and wild range to 4[m]

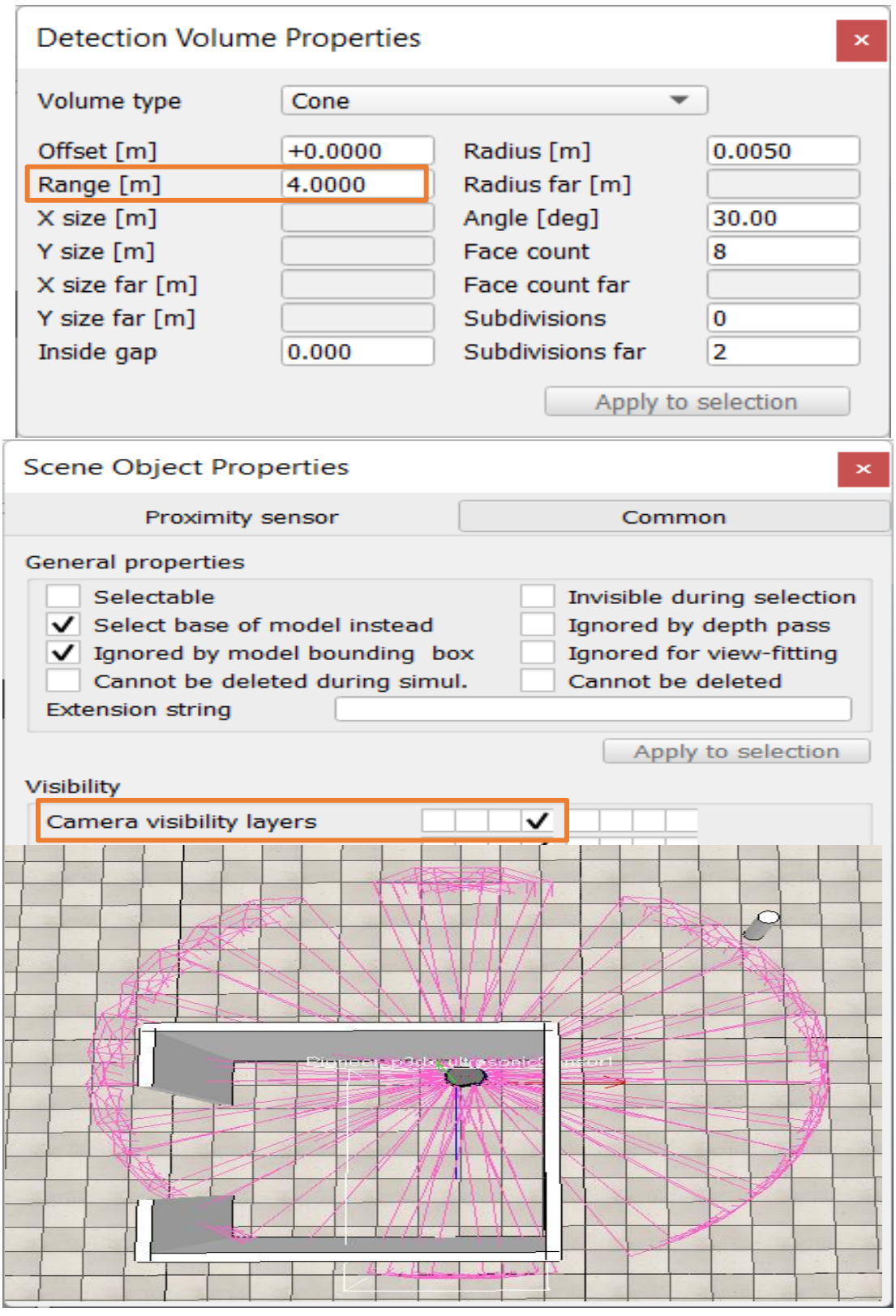


Figure 7 : Configuration Vision sensor and range

4.2 Set felt pen

When the robot run, we should know that it runs somewhere by using felt pen tool to record on the map

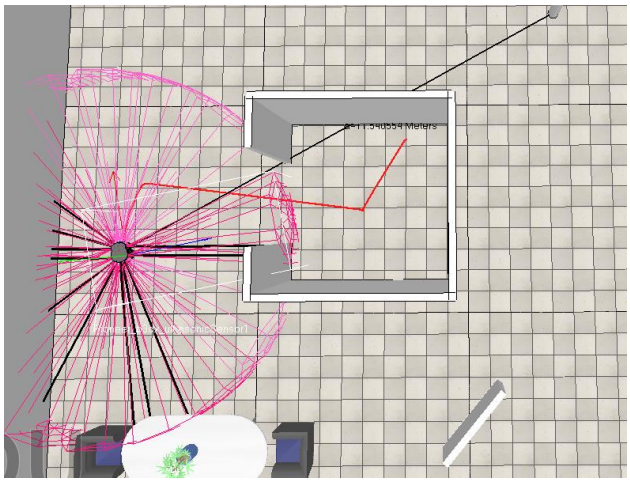


Figure 8 : Configuration felt pen

4.3 Set the object

When the robot run, it must avoid some obstructions into a map.

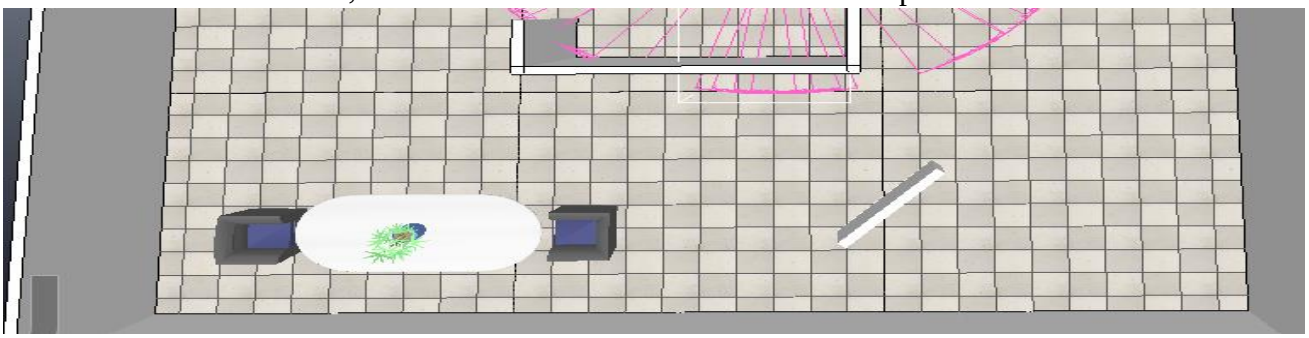


Figure 9 : Configuration obstruction

5. Result and discussion

Mission 1 : find the middle of the room. How to design the robot to find the middle of the room. I searched the solution to solve this mission (Solution 1). The way to achievement is used the robot uses the sensors to find x1,x2,y1,y2 and calculate the location of between (x1 + x2) / 2 , we will receive mean of X and (y1 + y2) /2, we will receive mean of Y, Then we will receive the center of the room of X and Y. But the problem is if the robot cannot calculate because of environment the door of small room. The value will be unlimited.

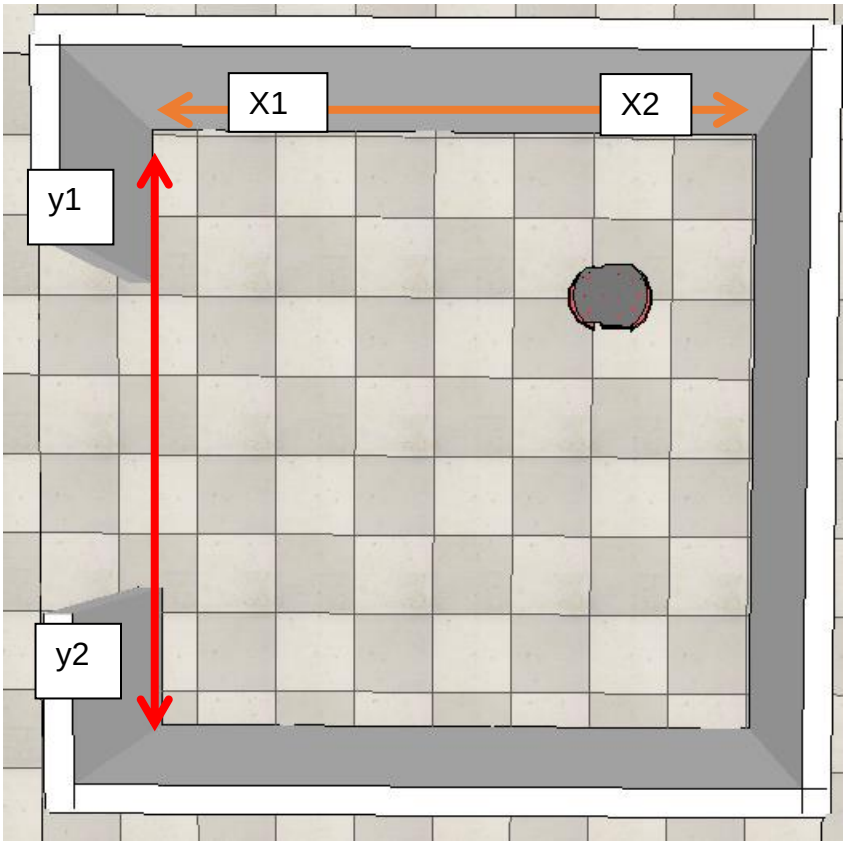


Figure 10 : How to find the middle of the room

However, we got a solution to implement but it is not successful. So, we changed another way by seeing in the simulator and setting the middle values and ordering the robot runs to location (Solution 2).

To understand more on the process the below flow chart is created

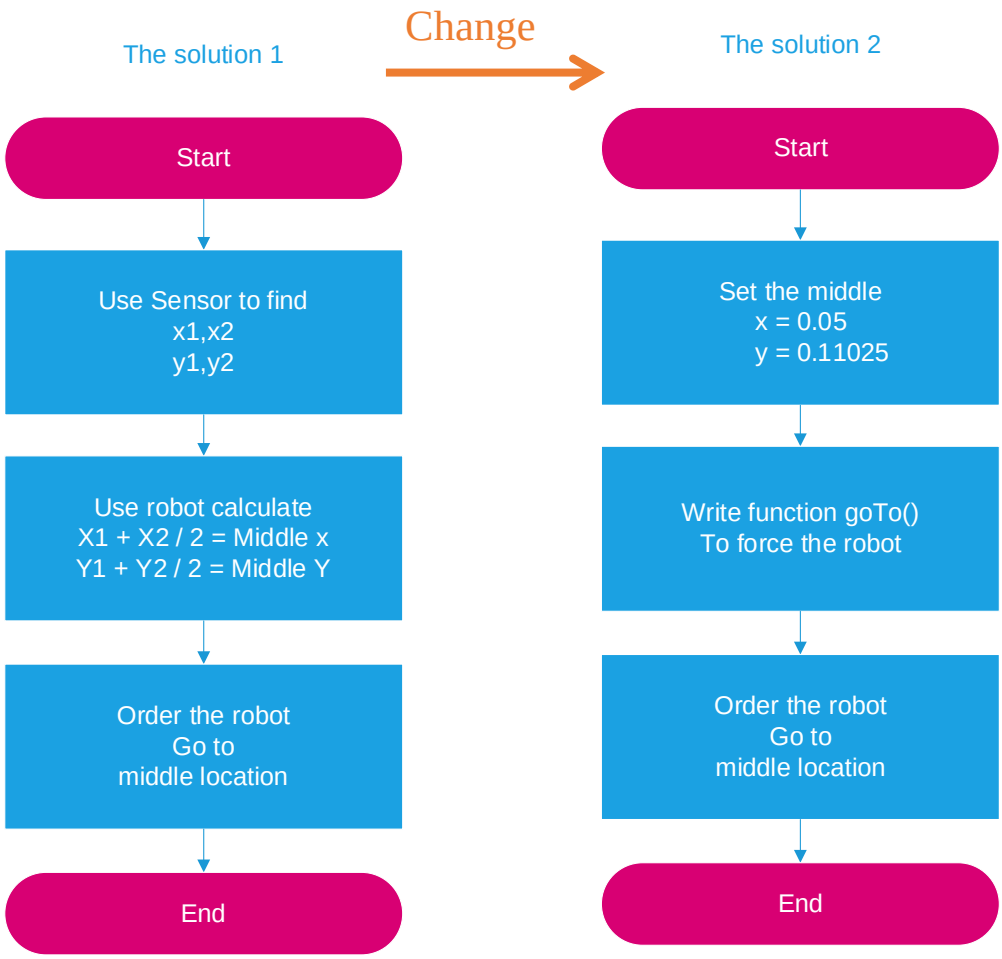
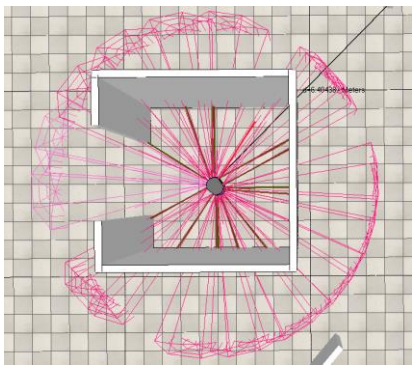


Figure 11 : Workflow of Mission 1 (Find the middle)



```
robot - Mission :  1 : Find Middle
robot - location :  [0.885018527507782, 1.6523551940917969, -3.114309549331665]
Searching the middle
Mission 1 Complete - Middle of the room Arrived
The middle :  [(0.05, 0.11025)]
```

Figure 12 : output of mission 1 to find the middle

Mission 2 : Leave the room, when the robot runs between the small room, the robot will detect and try to turn to avoid obstruction because following the condition that we set and the last the robot will not leave from a small room to crash the wall, the problem is how to set the robot to ignore the wall and run straight. So, the solution is set the robot turn the correct way and set to stop working of the sensor and order run.

To understand more on the process the below flow chart is created

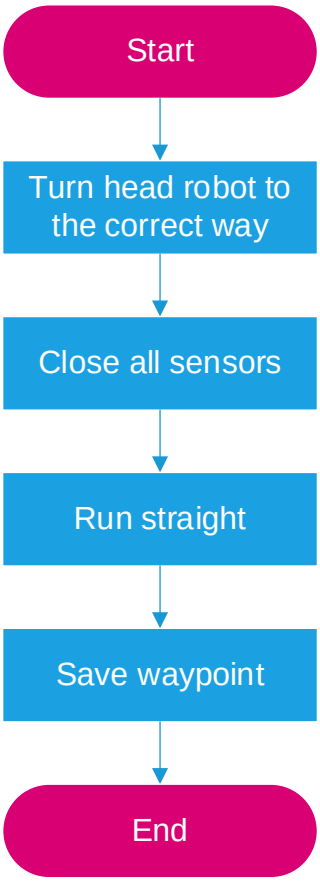
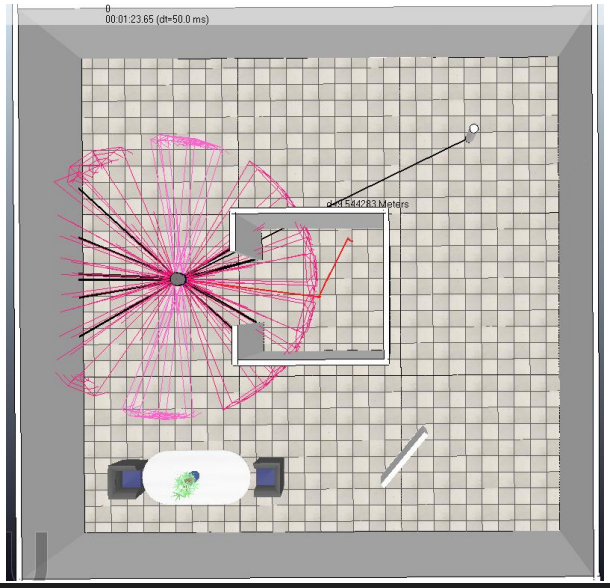


Figure 13 : Workflow of Mission 2 (Leave room)



```
robot - Mission :  2 : Leave Room
robot - location :  [0.022363994270563126, -0.08258186280727386, 2.765410900115967]
Current map :  [(0.05, 0.11025)]
Mission 2 Complete - robot leave of the room
```

Figure 14 : output of mission 2 Leave the room

Mission 3 : Wall Explorer, when the robot runs around the wall to explorer to keep data on the map, it must avoid obstruction. So, we have to configure functions and keep data.

To understand more on the process the below flow chart is created

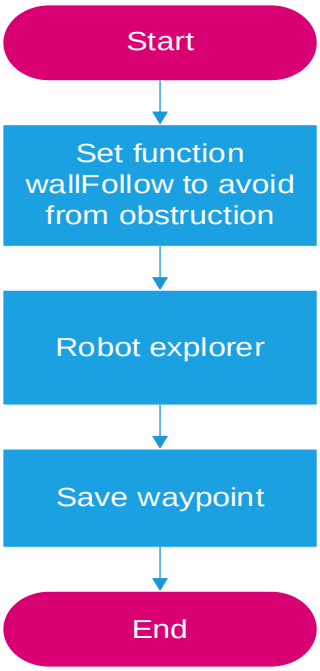
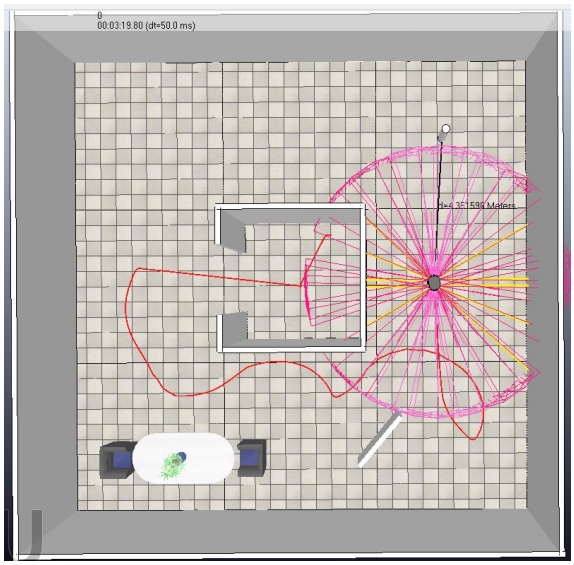


Figure 15 : Workflow of Mission 3 (Wall Explorer)



```
robot - Mission :  3 : Wall Explorer
robot - location :  [-3.9642581939697266, 0.5671107769012451, 2.982553243637085]
Current map :  [(0.05, 0.11025), [-3.6755619049072266, 0.520837664604187]]
Mission 3 Complete - Scan Room complete
```

Figure 16 : output of mission 3 Wall Explorer

Mission 4 : Find Beacon, when robot runs, it must detected all environment. So, the robot can meet the beacon and stop in front of beacon.

To understand more on the process the below flow chart is created

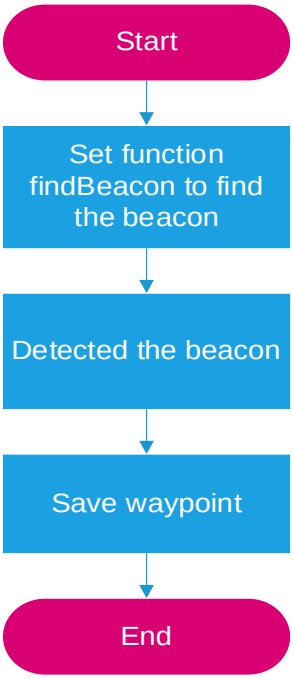
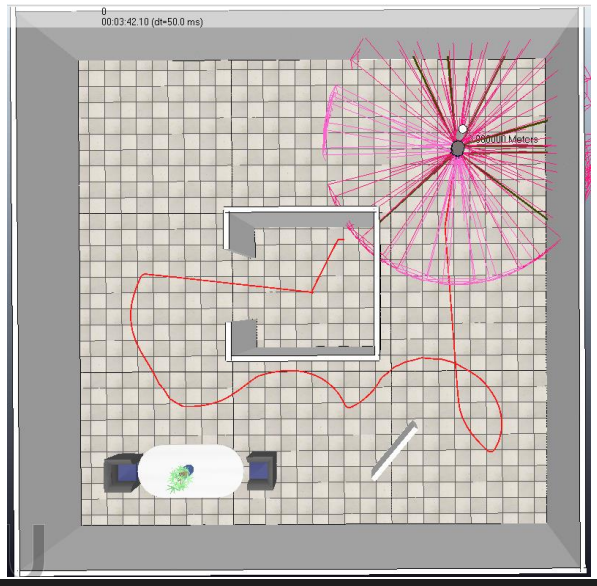


Figure 17 : Workflow of Mission 4 (Find the beacon)




```
robot - Mission :  4 : Find Beacon
robot - location :  [4.66829776763916, -3.7324366569519043, 1.761963129043579]
Current map :  [(0.05, 0.11025), [-3.8287100791931152, 0.43403321504592896]]
Beacon Detected
Current map :  [(0.05, 0.11025), [-3.8287100791931152, 0.43403321504592896], [4.232975006103516, 1.8335111141204834]]
Mission 4 Complete - found Beacon
```

Figure 18 : output of mission 4 Find the beacon

Mission 5 : Go home, when the robot finds and detects the beacon, it must go back to the center of the map by using the map location that the robot stores. So, it must use a location that has store to go back home.

To understand more on the process the below flow chart is created

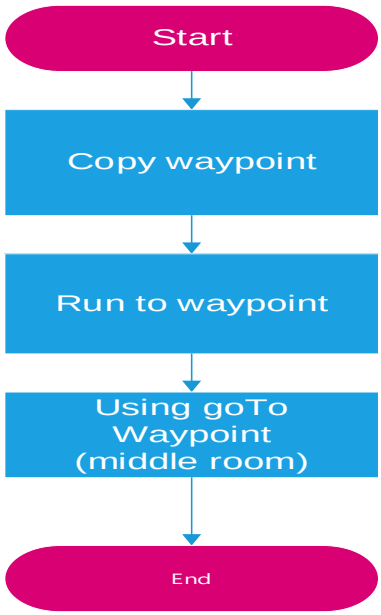
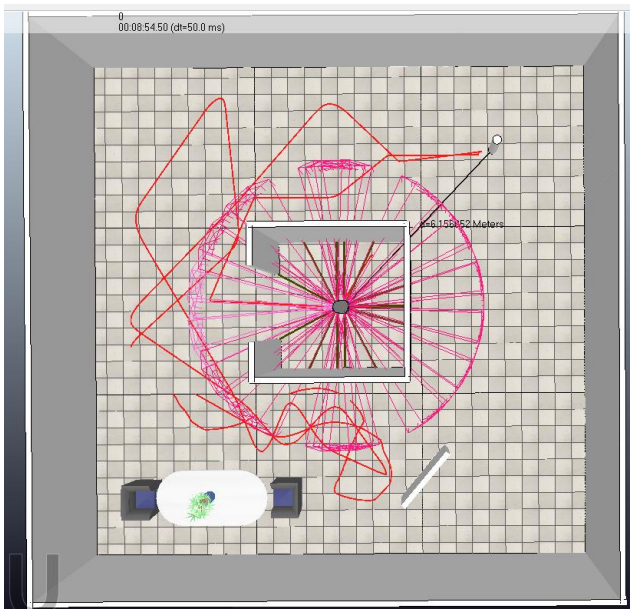


Figure 19 : Workflow of Mission 5 (Go home)




```
robot - Mission :  5 : Go Home
robot - location :  [4.317751884460449, 4.606032848358154, 0.5239296555519104]
Current map :  [(0.05, 0.11025), [-3.8376431465148926, 0.4359820485115051], [1.8661870956420898, 3.2804455757141113], [4.311267375946045, 4.600831031799316]]
Waypoint has arrived :  1.8661870956420898  ,  3.2804455757141113
Waypoint has arrived :  -3.8376431465148926  ,  0.4359820485115051
Waypoint has arrived :  0.05  ,  0.11025
Mission 5 Complete - robot go back to center of the small room
```

Figure 20 : output of mission 5 Go home

6. Conclusion

This project brings the fifth mission to do. we control the robot by using Python programming. The output we use CoppeliaSim to simulator tool. All missions have a step to process and individual. Firstly, the mission 1, the robot must run to the middle, we used the value of the real simulator to control the robot move. Secondly, the mission 2, the robot must leave of the room and without bumping the wall, we used function to control the robot turn correct way and run without open sonar sensors. Thirdly, the mission 3, the robot must follow the wall like a explorer, we used function follow the wall and keep data to store on map. Fourthly, the mission 4, the robot must find the beacon and stop in front of beacon, we used a function to keep a location of beacon and control to robot run. Lastly, the mission 5, after the robot detected the beacon, the robot will run go back to home by using go back waypoint.

7. Appendix

When we look at our assessment, we must design a strategy where the robot is run by using PID and stores the previous location (x,y) to follow the mission.

7.1 Design for the movement of the robot

If Speed Motor Left = Right && >= 1 : The robot run “Straight Forward”

If Speed Motor Left = Right && <= -1 : The robot run “Straight Backward”

```
def move(self, velocity):
    # velocity < 0 = reverse
    # velocity > 0 = forward
    res = sim.simxSetJointTargetVelocity(clientID, self.leftMotor,
    velocity, sim.simx_opmode_blocking)
    res = sim.simxSetJointTargetVelocity(clientID, self.rightMotor,
    velocity, sim.simx_opmode_blocking)
```

If Speed Motor Left > Right : The robot turn “Right”

If Speed Motor Left < Right : The robot turn “Left”

```
def turn(self, turnVelocity):
    # turnVelocity < 0 = trun left
    # turnVelocity > 0 = turn right
    res = sim.simxSetJointTargetVelocity(clientID, self.leftMotor,
    turnVelocity, sim.simx_opmode_blocking)
    res = sim.simxSetJointTargetVelocity(clientID, self.rightMotor,
    turnVelocity, sim.simx_opmode_blocking)
```

7.2 Using all sonars

```
# Setup Sonars
self.sonarHandles = []
text = 'Pioneer_p3dx_ultrasonicSensor'

for i in range(1,17):
    sonarText = text + str(i)
    res, tempHandle = sim.simxGetObjectHandle(clientID, sonarText, sim.simx_opmode_blocking)
    self.sonarHandles.append(tempHandle)
```

7.3 Configuration Parameters

```
33  #------------- Configulation Parameters -------------#
34  # Set status mission
35  MISSION = ["1 : Find Middle",
36             "2 : Leave Room",
37             "3 : Wall Follow",
38             "4 : Find Beacon",
39             "5 : Go Home","End Program"
40             ]          #To keep track of prime directive
41
42  THRESHOLD_BEACON = 3          #Max distance that the beacon can be detected, set standard to 7
43  MIN_DISTANCE_BEACON=0.5       #Min distance to stany away from the Beacon (to take robot dimension into account)
44  MIN_DISTANCE_ROOM=0.05        #Tolerance with regards to moving to the middle of the room
45  MIN_FRONT_DISTANCE=0.2        #Min distance accepted with regards to forward distance to the wall
46  MIN_SIDE_DISTANCE=1           #Min distance accepted with regards to side distance to the wall
47  ANTI_COLLISION_DISTANCE=1.5   #Distance at which the anti-collision becomes active
48  SETPOINT = 2;                 #The desired distance from the wall the robot must follow in the wall follow algorithm
49  RUNTIME = 50;               #Time the small room following logic should run
50  ENDTIME = 750;               #Time the wall following logic should run
51  SPEED_ROBOT=3                 #Speed at which the robot drives, note that PID tuning must be done again if changed!
52
53  ###Testing turning interval in inner room
54  ANGLES=[0, math.pi/8, math.pi/4, 3*math.pi/8, math.pi/2, 5*math.pi/8, 3*math.pi/4, 7*math.pi/8]
55
```

7.4 Set mission and PID

```
114             self.objective = MISSION[0] # Set mission to Array
115             self.antiCollisionDistance = ANTI_COLLISION_DISTANCE
116             self.minFrontDistance = MIN_DISTANCE_BEACON
117 
118             self.turnKp = 3.75 # Proportional Gain turn
119             self.turnKi = 0.006 # Integral Gain turn
120             self.turnKd = 6 #Derivative Gain turn
121 
122             self.Kp = 1 # Proportional Gain Follow Wall
123             self.Ki = 0 # Integral Gain Follow Wall
124             self.Kd = 35 # Derivative Gain Follow Wall
```

7.5 getPosition

```
148     def getPosition(self):
149         # Get the position form the robot && the world frame (-1 value sees )
150         res, self.fullPosition = sim.simxGetObjectPosition(clientID, self.robot, -1, sim.simx_opmode_blocking)
151 
152         # Get the alignment of the robot && the world axis (-1 value sees)
153         res, self.nextAngles = sim.simxGetObjectOrientation(clientID, self.robot, -1, sim.simx_opmode_blocking)
154         self.position[0] = self.fullPosition[0]
155         self.position[1] = self.fullPosition[1]
156         self.position[2] = self.nextAngles[2]
157         return self.position
```

7.6 Turn to Direction

```
179     def turnToDirection(self, targetAngle):
180         # Method to turn the robot to a specific direction/heading based on PID
181         # Input : direction / angle in rad to turn to PID
182         current_error = adjustAngle(self.getPosition()[2] - targetAngle) # Currect Error
183         previous_error = 0 # Previous Error
184         sum_error = 0 # Sum Error (Integral)
185 
186         # Corrent turn as long as the differecnt is larger than 1/2 degree
187         while abs(current_error) > math.pi/180:
188             sum_error = sum_error + current_error # Update the Sum error
189             derivative = current_error - previous_error # Calculate the derivative
190             previous_error = current_error # Update Previous error
191             # PID Process Calculate control function
192             cp = self.turnKp * current_error + self.turnKi * sum_error + self.turnKd * derivative
193 
194             self.curveCorner(cp, -cp)
195 
196             #calculate the current error
197             current_error = adjustAngle(self.getPosition()[2] - targetAngle)
198 
```

7.7 findDirection

```
def findDirection(self, x, y):
    # Find the angle between the current orientation of the robot and the target object
    # Input : target coordinates x and y || output : angle in rad

    # get the current position and oreientation
    xCurrent, yCurrent, phiCurrent = self.getPosition()

    # Calculate difference vector
    xDiff = x - xCurrent
    yDiff = y - yCurrent

    # Calculate target angle taking into account of tan -1
    if xDiff < 0 :
        phi = math.atan(yDiff / xDiff) - math.pi
    else:
        phi = math.atan(yDiff / xDiff)
    return phi
```

7.8 senseAll

```
def senseAll(self):
    # Pinging all sensor and returning the distances in a vector
    # Output : lisy of 16 distances
    sensor_readings = []

    for i in range(0, 16): sensor_readings.append(robot.getDistanceReading(i))
    return sensor_readings
```

7.9 findMiddleOfTheRoom

```
def findMiddleOfTheRoom(self):
    # finding the middle of the room
    # The middle is (x, y) maybe (0.05, 0.1) || output : (x,y)  coordinates of the middle of the room
    print("Searching the middle")
    x = 0.05
    y = 0.11025
    return (x, y)
```

7.10 goTo

```
def goTo(self, x, y, tolerance):
    # Control to the location that you want
    # Input : x,y coordinates of target || output: Distance to object
    self.getPosition()

    while distance(x, y, self.position[0], self.position[1]) > tolerance :
        # Determine the direction (x,y) lies and turn to it
        self.turnToDirection(self.findDirection(x, y))

        # ping the sonars
        sensor = self.senseAll()

        # if obstacle in front => avoid
        if (sensor[3] < ANTI_COLLISION_DISTANCE or
            sensor[4] < ANTI_COLLISION_DISTANCE) and (self.findBeacon()[0] > tolerance or self.findBeacon()[0] < 0):

            while (sensor[3] < ANTI_COLLISION_DISTANCE or
            sensor[4] < ANTI_COLLISION_DISTANCE or
            sensor[2] < ANTI_COLLISION_DISTANCE or
            sensor[5] < ANTI_COLLISION_DISTANCE) and (self.findBeacon()[0] > tolerance or self.findBeacon()[0] < 0):
                vL,vR = robot.drive(SPEED_ROBOT,SPEED_ROBOT)
                robot.curveCorner(vL,vR)
                sensor = self.senseAll()

            while (sensor[8] < ANTI_COLLISION_DISTANCE or
            sensor[15] < ANTI_COLLISION_DISTANCE or
            sensor[1] < ANTI_COLLISION_DISTANCE or
            sensor[6] < ANTI_COLLISION_DISTANCE) :
                self.move(SPEED_ROBOT)
                sensor = self.senseAll()
        # otherwise move forward
        self.move(2)
```

7.11 drive

```
def drive(self,vL,vR):
    # receives the proposed wheel SPEED_ROBOTs but adjects for
    listReturnCodes = []
    listDetectionStates = []
    listDistances = []
    detect = []

    braitenbergL = [-0.2,-0.4,-0.6,-0.8,-1,-1.2,-1.4,-1.6]
    braitenbergR = [-1.6,-1.4,-1.2,-1,-0.8,-0.6,-0.4,-0.2]

    for i in range(0, 8):
        [returnCode, detectionState, distance, d1, d2] = sim.simxReadProximitySensor(clientID, self.sonarHandles[i], sim.simx_opmode_buffer)
        listReturnCodes.append(returnCode)
        listDetectionStates.append(detectionState)
        listDistances.append(distance)
        if detectionState == 1 and np.linalg.norm(distance) < self.antiCollisionDistance :
            if np.linalg.norm(distance) < self.minFrontDistance:
                distance = self.minFrontDistance
            detect.append(1 - ((np.linalg.norm(distance) - self.minFrontDistance) / (self.antiCollisionDistance - self.minFrontDistance)))
        else :
            detect.append(0)

    vLeft = vL
    vRight = vR

    for i in range(0,8) :
        vLeft = vLeft + braitenbergL[i] * detect[i]
        vRight = vRight + braitenbergR[i] * detect[i]

    return vLeft,vRight
```

7.12 findBeacon

```
def findBeacon(self, thresholdbeacon = THRESHOLD_BEACON):
    # Input : MAX distance of beacon that can be detected
    # Output : distance and position of beacon (array of 3 values)

    # Maxinum Distance the robot can sense the beacon
    threshold = thresholdbeacon

    # Get the distance to the beacon
    res, distance = sim.simxReadDistance(clientID, self.distHandle, sim.simx_opmode_buffer)
    # GET beacon position in x,y,z cube relative to world
    res, beaconPosition = sim.simxGetObjectPosition(clientID, self.beaconHandle, -1, sim.simx_opmode_blocking)

    # return distance and position if beacon is in range
    if distance >= threshold:
        return -1, [], threshold
    else:
        return distance, beaconPosition, threshold
```

7.13 scan

```
def scan(self):
    # Keep of detected points in absolute / world reference
    xy = []

    for sonar in self.sonarHandles[0:16]:   # all sonar
        # ping sonar
        res, detectionState, detectedPoint,detectedObjectHandle, detectedSurfaceNormalVector = sim.simxReadProximitySensor(clientID, sonar, sim.simx_opmode_buffer)

        # If a point is detected
        if detectionState==1:
            # Get the sensor orientation in next Angle for the sensor
            res, nextAngles = sim.simxGetObjectOrientation(clientID, sonar, -1, sim.simx_opmode_blocking)

            # Get position for the sonar VS the world frame (-1 values = sees)
            res, position = sim.simxGetObjectOrientation(clientID, sonar, -1, sim.simx_opmode_blocking)

            # Rotate the coordinates of the found point to receive absolute coordinates (- the offset of the sensor location)
            v1 = rotate(detectedPoint[0], detectedPoint[1], detectedPoint[2], nextAngles[0], nextAngles[1], nextAngles[2])

            # Add x and y coordinates of the detected point plus the location of the sensor to get absolute coordinates
            xy.append((position[0] + v1[0], position[1] + v1[1]))

    return xy
```

7.14 scanAround

```
def scanAround(self):
        # return function list of coordinates detected around the robot when turning around its axis in pi/8 degree increase
        xy = []
        for angle in ANGLES:
            self.turnToDirection(angle)
            xy += self.scan()

        return xy
```

7.15 wallFollow

```
def wallFollow(self):
    current_error = 0
    previous_error = 0
    sum_error = 0

    for i in range(ENDTIME):
        # Scan around every 30 ticks
        if i % 30 == 0:
            # Stop the require from the last driving = operates too long
            self.stop()
            map.addPoints(self.scan())

        # Ping sonars
        distanceFront = self.getDistanceReading(4)
        distanceRight = self.getDistanceReading(8)

        # Setting the front flag
        if (distanceFront <= SETPOINT): # object in front => turn left
            frontFlag = True
        else :
            frontFlag = False  # Don't have in front => keep running

        # Seting the Right Flag
        if distanceRight == 9999 :
            rightFlag = False  # Don't have in right => turn right
        else :
            rightFlag = True  # When detected on the right => robot will follow the wall

        if frontFlag :
            if distanceFront < MIN_FRONT_DISTANCE :
                # Robot will stuck, back out random
                self.curveCorner(-random.random(), -random.random())
                time.sleep(0.1)
    else :
        # turn left inplace
        self.curveCorner(0,SPEED_ROBOT)

elif rightFlag :
    # PID calculate the current error
    current_error = distanceRight - SETPOINT
    # Update Sum error
    sum_error = sum_error + current_error
    # Calculate the derivative
    derivative = current_error - previous_error
    # Updaet the previous error
    previous_error = current_error
    # calculate control function
    cp = self.Kp * current_error + self.Ki * sum_error + self.Kp * derivative

    self.curveCorner(SPEED_ROBOT,SPEED_ROBOT-cp)

else:
    # turn right inplace because the wall sloped away
    self.curveCorner(SPEED_ROBOT, 0)
```

7.16 Class map

```
class Map1() :
    def __init__(self) :
        # Parameters of the map
        self.wayPoints = []  # list of waypoints in (x,y) coordinates

    def getWayPoint(self, number) :
        return self.wayPoints[number]

    def getWayPoints(self) :
        return self.wayPoints

    def addWayPoint(self, wayPoint) :
        self.wayPoints.append(wayPoint)
```

7.17 Class map2

```
class Map2(Map1) :
    def __init__(self):
        # Upgrade Map1 > continue from map
        super().__init__()
        self.points = []  # List detected all point
        self.map = np.zeros((2, 2))   # creating a dummy 2x2 matrix

    def addPoints(self, points) :
        self.points += points

    def getPoints(self) :
        return(self.points)

    def printMap(self) :
        # Print XY scatter map of detected Points
        coord = list(zip(* self.getPoints()))

        # Building the plot
        plt.clf()
        plt.ylabel('Y-Axis')
        plt.xlabel('X-Axis')
        plt.title("Map of the Room")
        plt.scatter(coord[0], coord[1])
        plt.show()

    def createMap(self, resolution):
        # Input : points is list of (x, y) coordinates
        # Input : resolution is the size of an individual square of the map in m

        # splitting the points in 2 list of x and y
        coord = list(zip(* self.getPoints()))
```

```
        # Calculating the range and number of grid squares
        Xmax = int((max(coord[0]) - min(coord[0])) // resolution + 1) # range X
        Ymax = int((max(coord[1]) - min(coord[1])) // resolution + 1) # range Y

        self.max = np.zeros((Ymax, Xmax)) # create empty grid

    def updateMap(self, points, resolution):
        # Input : points is list of (x, y) coordinates
        # Input : resolution is the size of an individual square of the map in m

        # splitting the points in 2 list of x and y
        coord = list(zip(* self.getPoints()))

        # Calculating the range and number of grid squares
        xmin = min(coord[0]) # lower x value
        ymin = min(coord[1]) # lower y value

        for point in points:
            x = int((point[0] - xmin) // resolution)
            y = int(-(point[1] - ymin) // resolution)
            self.map[y][x] = 1

    def printGrid(self):
        plt.clf()
        plt.imshow(self.map)
        plt.show
```

7.18 Extra function

```
#------------ Extra function -----------#

def adjustAngle(angle) :
    # return angles with circles added
    while angle > math.pi : angle -= math.pi*2
    while angle <= -math.pi : angle += math.pi*2
    return angle

def distance (x1, y1, x2, y2) :
    # calculate distance between 2 points in 2 dim
    # input : (x1, y1) coordinates point 1, (x2, y2) coordinates point 2
    return math.sqrt((x1 - x2) **2 + (y1 - y2) **2)

def rotationMatrix(a, b, g):
    # Return the rotation matrix using Tait-Bryan angles alpha, beta and gamma
    rot = np.array([(cos(b) * cos(g), 0, sin(b)),

                    (cos(g) * sin(a) * sin(b), 0, -cos(b) * sin(a)),

                    (0, cos(g) * sin(a), 0)], dtype = float)
    return rot

def rotate(x, y, z, a, b, g):
    # Input : x y z Into -> sensor frame of reference
    # output : vector of x y z that they are rotation x y z
    v1 = np.array([(x), (y), (z)])
    v2 = np.matmul(rotationMatrix(a, b, g), v1)
    return v2
```

7.19 Start main program

```
if clientID!=-1:
    print ('Connected to remote API server')
    res,objs=sim.simxGetObjects(clientID,sim.sim_handle_all,sim.simx_opmode_blocking)
    if res==sim.simx_return_ok:
        print ('Number of objects in the scene: ',len(objs))
    else:
        print ('Remote API function call returned with error code: ',res)

    print()
    map = Map2()
    print(" Create Map ")
    print()

    # robot = Robot()
    robot = Robot2()
    print("robot - Mission : ", robot.getObjective())
    print("robot - location : ",robot.getPosition())
    # # Now try to retrieve data in a blocking fashion (i.e. a service call):
    # for i in range(10):
    #     print(robot.getDistanceReading(robot.frontLeftSonar))
    #     print(robot.getDistanceReading(robot.Front_30_RightSonar))
    #     print(robot.getDistanceReading(robot.RightSonar))
    #     print(robot.getDistanceReading(robot.backRightSonar))

        # time.sleep(0.2)

    # Function used to return a random integer within the range of 0 and 100 for the random wander
    randomNumber = random.randint(0, 100)

    while robot.getObjective() != "End Program":
        # the robot will move untill it detects an object
        robot.move(1)
```

7.20 Mission 1

```
#------------------- Mission 1 Find the middle ------------------#

        if robot.getObjective() == MISSION[0]: # Mission 1 : Find the middle of the small room
            # robot.smallroomFollow()

            middle = robot.findMiddleOfTheRoom()

            robot.goTo(middle[0], middle[1], MIN_DISTANCE_ROOM)

            # Modift 1 - scanning the smallroom
            map.addPoints(robot.scanAround())

            robot.setObjective(MISSION[1])
            # Add waypoint to the map
            map.addWayPoint(middle)
            print("Mission 1 Complete - Middle of the room Arrived")
            print("The middle : ", map.getWayPoints())
            print()

            print("robot - Mission : ", robot.getObjective())
            print("robot - location : ",robot.getPosition())
            print("Current map : ", map.getWayPoints())
```

7.21 Mission 2

```
#------------------ Mission 2 Leave of the room ------------------#

        elif robot.getObjective() == MISSION[1]: # Mission 2 : Leave of the small room
            sensor = robot.senseAll() # Sensor contrains a list with readings from all sensors
            while sensor[3] != 9999 or sensor[4] != 9999: # turn toward the opening
                robot.turn(0.5)
                sensor = robot.senseAll()

            # leave the room
            while sensor[0] != 9999 or sensor[15] != 9999 or sensor[7] != 9999 or sensor[8] != 9999:
                robot.move(SPEED_ROBOT)
                sensor = robot.senseAll()

                # Modify 2 - scanning while driving
                map.addPoints(robot.scan())

            time.sleep(1)
            robot.stop

            # Adding second waypoint to map
            currentPosition = robot.getPosition()
            map.addWayPoint([currentPosition[0], currentPosition[1]])

            # Modify 3 - scanning the second waypoint
            map.addPoints(robot.scan())

            robot.setObjective(MISSION[2])
            print("Mission 2 Complete - robot leave of the room")
            print()
            print("robot - Mission : ", robot.getObjective())
            print("robot - location : ",robot.getPosition())
            print("Current map : ", map.getWayPoints())
```

7.22 Mission 3

```
#------------------ Mission 3 Wall Explorer and Scan ------------------#

        elif robot.getObjective() == MISSION[2]: # Mission 3 : Wall Explorer and scanning
            while robot.getDistanceReading(4) > SETPOINT:
                robot.move(SPEED_ROBOT)

            robot.wallFollow()
            robot.setObjective(MISSION[3])
            print("Mission 3 Complete - Scan Room complete")
            print()
            print("robot - Mission : ", robot.getObjective())
            print("robot - location : ",robot.getPosition())
            print("Current map : ", map.getWayPoints())
```

7.23 Mission 4

```
#------------------ Mission 4 Random around beacon ------------------#

        elif robot.getObjective() == MISSION[3]: # Mission 4 Random around beacon
            beaconSensing = robot.findBeacon()
            while beaconSensing[0] == -1: # while no beacon is detected
                vL, vR = robot.drive(SPEED_ROBOT,SPEED_ROBOT)
                robot.curveCorner(vL,vR)
                beaconSensing = robot.findBeacon()

                # Modify 4 - scanning while driving
                map.addPoints(robot.scan())

            print("Beacon Detected")
            # Add waypoint of Beacon to the map
            currentPosition = robot.getPosition()
            map.addWayPoint([currentPosition[0], currentPosition[1]])
            print("Current map : ", map.getWayPoints())
            # Tolerance is the distance from the center -> it is not edge of the robot to center beacon
            robot.goTo(beaconSensing[1][0], beaconSensing[1][1], MIN_DISTANCE_BEACON)
            robot.setObjective(MISSION[4])
            currentPosition = robot.getPosition()
            map.addWayPoint([currentPosition[0], currentPosition[1]])
            print("Mission 4 Complete - found Beacon")
            print()
            print("robot - Mission : ", robot.getObjective())
            print("robot - location : ",robot.getPosition())
            print("Current map : ", map.getWayPoints())
            time.sleep(5)
            robot.stop()
```

7.24 Mission 5

```
#------------- Mission 5 Go back to center of the small room -------------#

        elif robot.getObjective() == MISSION[4]: # Go back to center of the small room
            waypoints = map.wayPoints.copy() # Copt the waypoint
            waypoints.pop() # Get rib of last waypoint because the robot is done

            for waypoint in range(len(map.wayPoints) -1) :
                target = waypoints.pop()
                robot.goTo(target[0], target[1], MIN_DISTANCE_ROOM)
                print("Waypoint has arrived : ", target[0], " , ", target[1])

            # Show the result that the robot complete
            print("Mission 5 Complete - robot go back to center of the small room")
            print()

            robot.setObjective(MISSION[5])
```

8. References

[1] What is Pioneer 3DX. https://www.generationrobots.com/media/Pioneer3DX-P3DX-RevA.pdf (accessed Nov. 23, 2021).

[2] What is Python. https://www.coursera.org/articles/what-is-python-used-for-a-beginners-guide-to-using-python (accessed Nov. 23, 2021).

[3] The robot simulator CoppeliaSim. https://www.coppeliarobotics.com (accessed Nov. 23, 2021).

[4] What is a PID controller : Working & Its applications. https://www.elprocus.com/the-working-of-a-pid-controller/ (accessed Nov.23, 2021)