



Department of  
Computer Engineering

به نام خدا



Amirkabir University of Technology  
(Tehran Polytechnic)

دانشگاه صنعتی امیرکبیر  
دانشکده مهندسی کامپیوتر  
**اصول علم ربات**

تمرین سری سوم

سارا تاجرنيا	نام و نام خانوادگی
۹۸۳۱۰۱۶	شماره دانشجویی
۱۴۰۱/۳/۱۰	تاریخ ارسال گزارش

## فهرست گزارش سوالات

3.....	بخش تئوری
3 .....	سوال 1 – active/ passive
4.....	سوال 2 – GPS
5 .....	سوال 3 – dead reckoning heading
6 .....	سوال 4 – قطب نما در ناوبری کور heading
6.....	بخش عملی
7 .....	سناریو اول
7 .....	الف
10.....	ب
13.....	ج
15.....	سناریو دوم
15.....	الف
19.....	ب
21.....	ج

## بخش تئوری

### سوال ۱ – active/ passive

۱. دو نمونه از سنسور های active و دو نمونه از سنسور های passive نام ببرید. آیا GPS یک سنسور active است یا passive؟ جواب خود را تشریح کنید. (۵ امتیاز)

سنسور های active ← فاصله یاب لیزری (laser rangefinder) - سنسور بازتاب (Optical encoders) - رمزگذار های نوری (reflectivity sensor)

سنسور های passive ← قطب نما (compass) - ژیروسکوپ ها (gyroscopes) دوربین های (CCD/CMOS camera) CCD/ CMOS

یک سنسور active GPS است زیرا کارایی در این سنسور به این صورت است که انرژی را ساطع کرده و واکنش را اندازه گیری میکند (حاصل از محیط نیست). البته باید در نظر گرفت که برای این کار ویژگی های محیط تاثیرگذار است.

## سوال ۲ – GPS

۲. برای مکان یابی در داخل ساختمان کدام یک از سنسورهای زیر کارایی مناسبی ندارد؟ گزینه صحیح را انتخاب و دلیل را شرح دهید. (۵ امتیاز)

GPS

IMU

Odometry

Wireless Beacon

زیرا سنسور GPS در مکان یابی های با مساحت بالا استفاده میشود و اگر برای داخل یک ساختمان استفاده شود چون مساحت نسبتا پایین دارد خطای بسیار بالا رفته و دقت کمی داریم.

به طوری که دقت اسمی GPS در حدود 10 متر است. ( البته با فیلتر کردن می توان آن را به 2-3 متر رساند) که این دقت برای مکان یابی داخل ساختمان خطای بالایی دارد.

### سوال ۳ – heading در dead reckoning

۳. در ناوبری کور (dead reckoning) برای تشخیص heading چه سنسوری و چگونه مورد استفاده قرار می‌گیرد؟ (۵ امتیاز)

در ناوبری کور برای تشخیص heading یا جهت‌گیری از سنسور قطب نما (Compass) و شیب سنج (Inclinometer) استفاده می‌شود.

دلیل استفاده از Compass این است که از اندازه گیری مطلق برای جهت‌گیری استفاده می‌کند همچنین از طیف وسیعی راه حل‌ها برای اندازه گیری در نظر می‌گیرد.

همچنین دلیل استفاده از Inclinometer فناوری اندازه گیری شیب‌ها است که معمولاً بر اساس سیالات است.

## سوال ۴ – قطب نما در ناوبری کور heading

۴. قطب نما در ناوبری کور چگونه می تواند به تشخیص heading کمک کند؟ (۵ امتیاز)

دلیل استفاده از قطب نما در ناوبری کور این است که از اندازه گیری مطلق برای جهت گیری استفاده میکند همچنین طیف وسیعی از راه حل ها برای اندازه گیری در نظر میگیرد.

همچنین ضعف هایی که این قطب نما دارد از جمله ضعف میدان زمین، اختلال توسط منابع مغناطیسی دیگر، محدودیت های پهنه ای باند، جهت گیری غیر مطلق است.

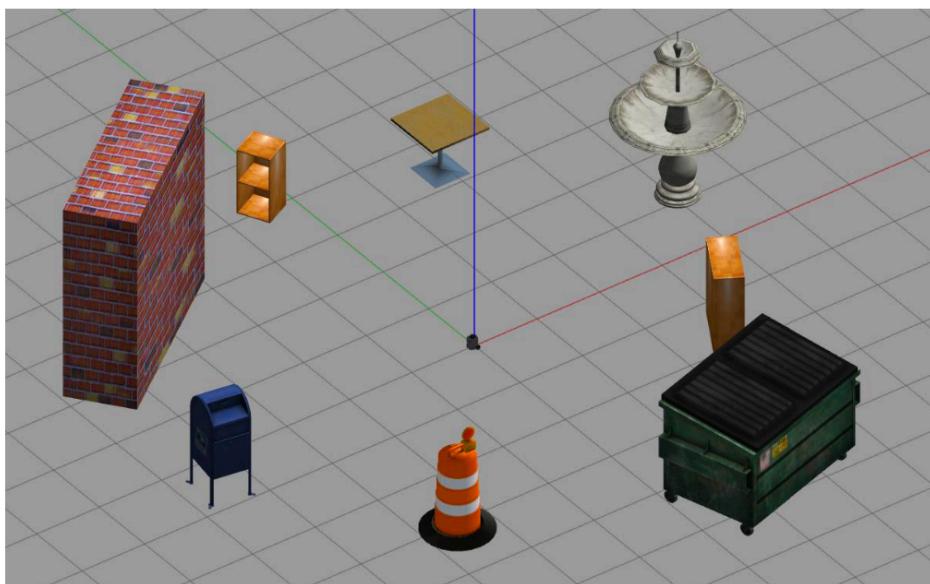
## بخش عملی

### سناریو اول (الف)

#### سناریوی ۱. نزدیک ترین مانع

ابتدا ربات را در دنیای داده شده که فایل آن ضمیمه شده است (detect\_obstacles.world)، قرار دهید. در این دنیا در اطراف ربات تعدادی مانع موجود است. موقعیت مرکز و نام هر کدام از موانع در اختیار شما داده می‌شود (obstacle\_positions.txt). با استفاده از پکیج turtlebot3\_teleop و دستور اجرایی زیر شما می‌توانید ربات را با کلید های کیبورد در محیط حرکت دهید.

`roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch`



(الف) **امتیاز ۱۰** در این بخش می‌بایست یک نود تشکیل دهید که بتواند فاصله ربات تا مرکز هر کدام از موانع را بیابد و سپس نزدیکترین مانع و فاصله ربات تا آن مانع را روی تاپیک‌ای منتشر کند. بدین منظور ربات موقعیت فعلی خود را از طریق تاپیک "odom" دریافت (subscribe) می‌نماید. سپس با داشتن موقعیت فعلی و مرکز هر کدام از موانع (در فایل obstacle\_positions.txt) اقدام به محاسبه فاصله‌ی خود تا این مانع می‌نماید. در انتها کمترین مقدار فاصله و نام مانع مربوطه (نزدیکترین مانع به ربات) را یافته و به صورتی پیامی بر روی تاپیک "ClosestObstacle" منتشر می‌نماید. فرمت پیام به صورت زیر باشد: (نام مانع، فاصله تا مرکز نزدیکترین مانع)

```
string obstacle_name  
float64 distance
```

توجه: اطلاعات مربوط به مانع را در هر node که نیاز بود می‌توانید بصورت دستی در خود کد وارد کنید.

در ابتدا قبل از پرداختن به قسمت الف باید فایل را تحت عنوان s1 با turtlebot3 تولید کرد.

قطعه کد launch این قسمت به شکل زیر است:

```
<launch>
  <node pkg="s1" type="dis.py" name="dis" output="screen">
    </node>
    <arg name="model" default="$(env TURTLEBOT3_MODEL)" doc="model type [burger, waffle, waffle_pi]"/>
    <arg name="x_pos" default="0.0"/>
    <arg name="y_pos" default="0.0"/>
    <arg name="z_pos" default="0.0"/>
    <arg name="yaw" default="0.0"/>
    <include file="$(find gazebo_ros)/launch/empty_world.launch">
      <arg name="world_name" value="$(find s1)/detect_obstacles.world"/>
      <arg name="paused" value="false"/>
      <arg name="use_sim_time" value="true"/>
      <arg name="gui" value="true"/>
      <arg name="headless" value="false"/>
      <arg name="debug" value="false"/>
    </include>

    <param name="robot_description" command="$(find xacro)/xacro --inorder $(find \
      turtlebot3_description)/urdf/turtlebot3_$(arg model).urdf.xacro" />

    <node pkg="gazebo_ros" type="spawn_model" name="spawn_urdf" args="-urdf -model turtlebot3_$(arg model) \
      -x $(arg x_pos) -y $(arg y_pos) -z $(arg z_pos) -param robot_description" />
  </launch>
```

کد به صورت زیر است:

```
#!/usr/bin/python3

from cmath import rect
import rospy
from nav_msgs.msg import Odometry
from math import sqrt
from s1.msg import co

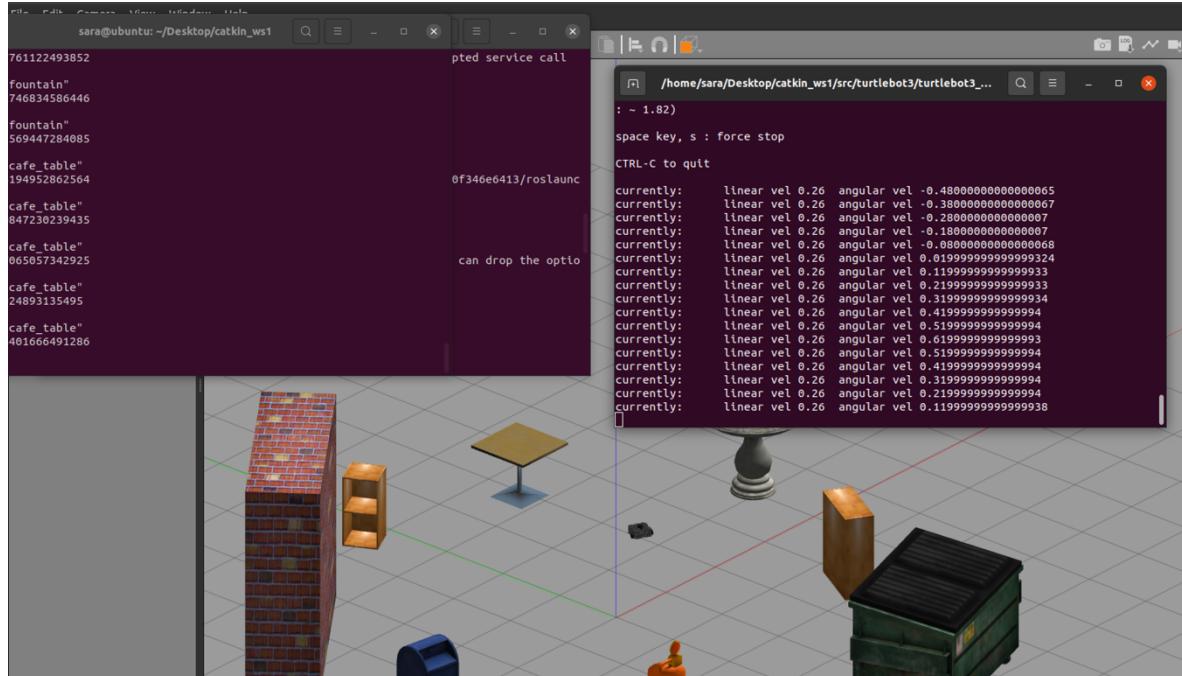
def Barriers(name, distance):
    barriers = calculate()
    barriers.obstacle_name = name
    barriers.distance = distance
    return barriers

class distances1:
    def __init__(self) -> None:
        self.default, self.book_shelf, self.dumpster, self.barrel, self.postbox, self.cabinet, self.cafe_table, self.fountain =
        -1, (2.64, -1.55), (1.23, -4.57), (-2.51, -3.08), (-4.47, -0.57), (-3.44, 2.75), (-0.45, 4.05), (1.91, 3.37), (4.08, 1.14)
        self.obstacles = [("book_shelf", self.book_shelf), ("dumpster", self.dumpster), ("barrel", self.barrel), ("postbox", self.postbox),
                          ("brick_box", self.brick_box), ("cabinet", self.cabinet), ("cafe_table", self.cafe_table), ("fountain", self.fountain)]
        rospy.init_node("controller", anonymous=False)
        self.cal = rospy.Publisher('Barriers', calculate, queue_size=10)

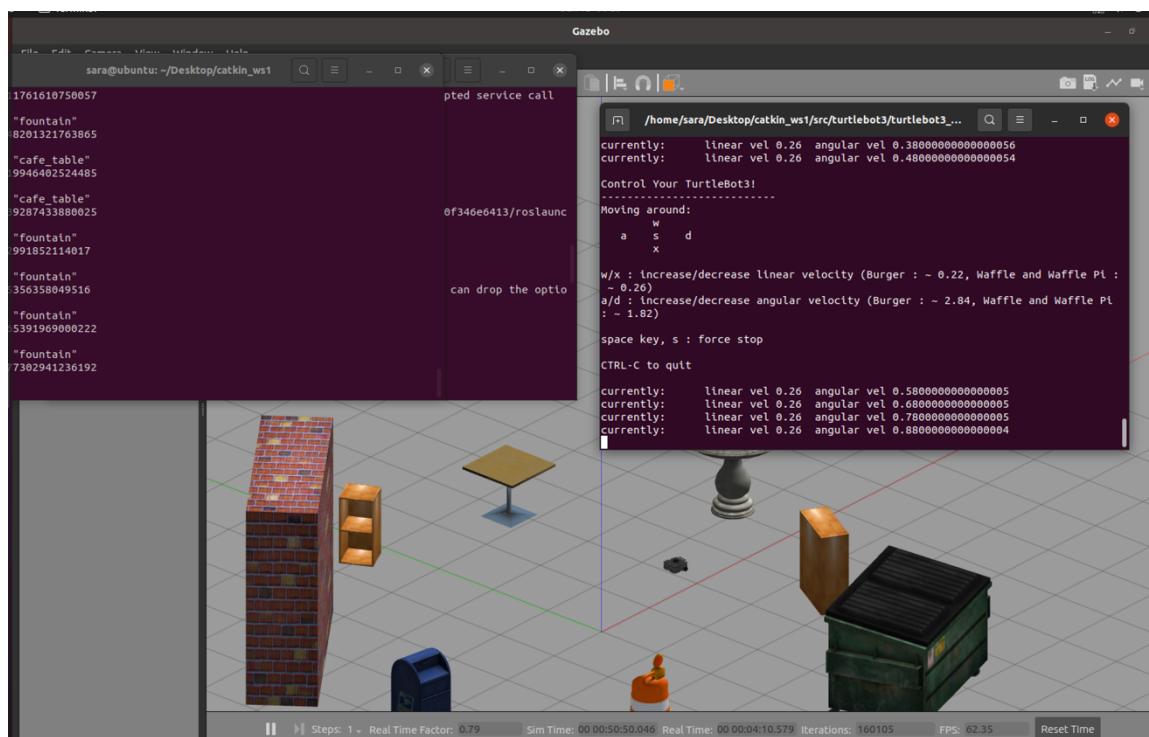
    def calculate_distance(self):
        while True:
            if not rospy.is_shutdown():
                rate = rospy.Rate(1)
                msg = rospy.wait_for_message("odom", Odometry)
                curr_x, curr_y = msg.pose.pose.position.x, msg.pose.pose.position.y
                x_dis, y_dis = obstacle[1][0] - curr_x, obstacle[1][1] - curr_y
                distances = [(obstacle[0], sqrt((x_dis*x_dis) + (y_dis*y_dis))) for obstacle in self.obstacles]
                minimum = min(distances, key=lambda x: x[1])
                calculate = Barriers(minimum[0], minimum[1])
                self.cal.publish(calculate)
                rate.sleep()

if __name__ == "__main__":
    distance = distances1()
    distance.calculate_distance()
```

در ادامه برای این قسمت یک فایل را طوری طراحی میکنیم تا در ابتدا بتوانند فاصله بین موقعیت مکانی فعلی ربات را تا مرکز اشیا که موقعیت دقیق آن در کد آمده مشخص کند سپس از بین این فاصله ها کمترین آن را مشخص کرده و آن را برمیگرداند برای مثال در شکل زیر همانطور که پیداست کمترین فاصله با cofe table است:



اگر موقعیت ربات را تغییر دهیم همان طور که در شکل پیدا است کمترین فاصله با ربات برابر شی است:



## سناریو اول (ب)

ب) **۱۵ (امتیاز)** هدف از این بخش بازنویسی بخش الف با استفاده از سرویس‌هاست (به هندزان ۴ مراجعه کنید). ابتدا یک node دیگر ایجاد کنید. در این node یک سرویس به اسم GetDistance بسازید که به عنوان ورودی نام مانع را به صورت یک رشته دریافت کرده و در خروجی فاصله تا مرکز آن مانع را بصورت float بر می‌گرداند. حال node قسمت الف را با استفاده از این سرویس بازنویسی کنید.

ورودی سرویس:

string obstacle\_name

خروجی سرویس:

float64 distance

در این قسمت میتوان فاصله هر یک از اشیا مورد نظر را با موقعیت فعلی ربات پیدا کرد. قطعه کد launch این قسمت به شکل زیر است:

```
<launch>
  <node pkg="s1" type="distance_node.py" name="distance_node" output="screen">
    </node>
    <arg name="model" default="$(env TURTLEBOT3_MODEL)" doc="model type [burger, waffle, waffle_pi]"/>
    <arg name="x_pos" default="0.0"/>
    <arg name="y_pos" default="0.0"/>
    <arg name="z_pos" default="0.0"/>
    <arg name="yaw" default="0.0"/>

    <include file="$(find gazebo_ros)/launch/empty_world.launch">
      <arg name="world_name" value="$(find s1)/detect_obstacles.world"/>
      <arg name="paused" value="false"/>
      <arg name="use_sim_time" value="true"/>
      <arg name="gui" value="true"/>
      <arg name="headless" value="false"/>
      <arg name="debug" value="false"/>
    </include>

    <param name="robot_description" command="$(find xacro)/xacro --inorder $(find
      | turtlebot3_description)/urdf/turtlebot3_${arg model}.urdf.xacro" />

    <node pkg="gazebo_ros" type="spawn_model" name="spawn_urdf" args="-urdf -model turtlebot3_${arg model}
      | -x ${arg x_pos} -y ${arg y_pos} -z ${arg z_pos} -param robot_description" />
  </launch>
```

کد زیر به شکل زیر است:

```
#!/usr/bin/python3

from cmath import rect
import rospy
from nav_messages.message import Odometry
from math import sqrt
from s1.srv import GetDistance, distance1

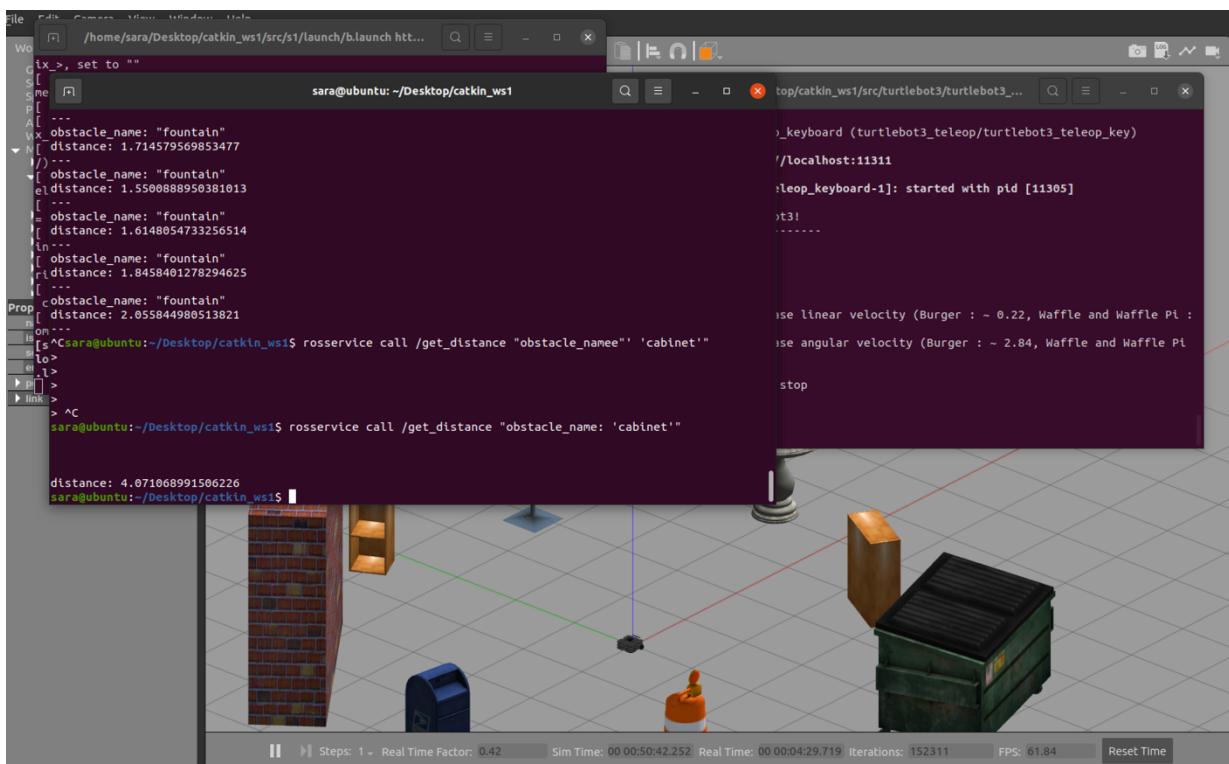
def find_distance():
    rospy.init_node('distance_node', anonymous=True)
    distance2 = distance()
    distance3 = rospy.Service('get_distance', GetDistance, get_distance)
    rospy.spin()

class Distance():
    def __init__(self) -> None:
        self.default, self.book_shelf, self.dumpster, self.barrel, self.postbox, self.cabinet, self.cafe_table, self.fountain = \
            -1, (2.64, -1.55), (1.23, -4.57), (-2.51, -3.08), (-4.47, -0.57), (-3.44, 2.75), (-0.45, 4.05), (1.91, 3.37), (4.08, 1.14)
        self.obstacles = {
            "book_shelf": self.book_shelf,
            "dumpster": self.dumpster,
            "barrel": self.barrel,
            "postbox": self.postbox,
            "brick_box": self.brick_box,
            "cabinet": self.cabinet,
            "cafe_table": self.cafe_table,
            "fountain": self.fountain
        }

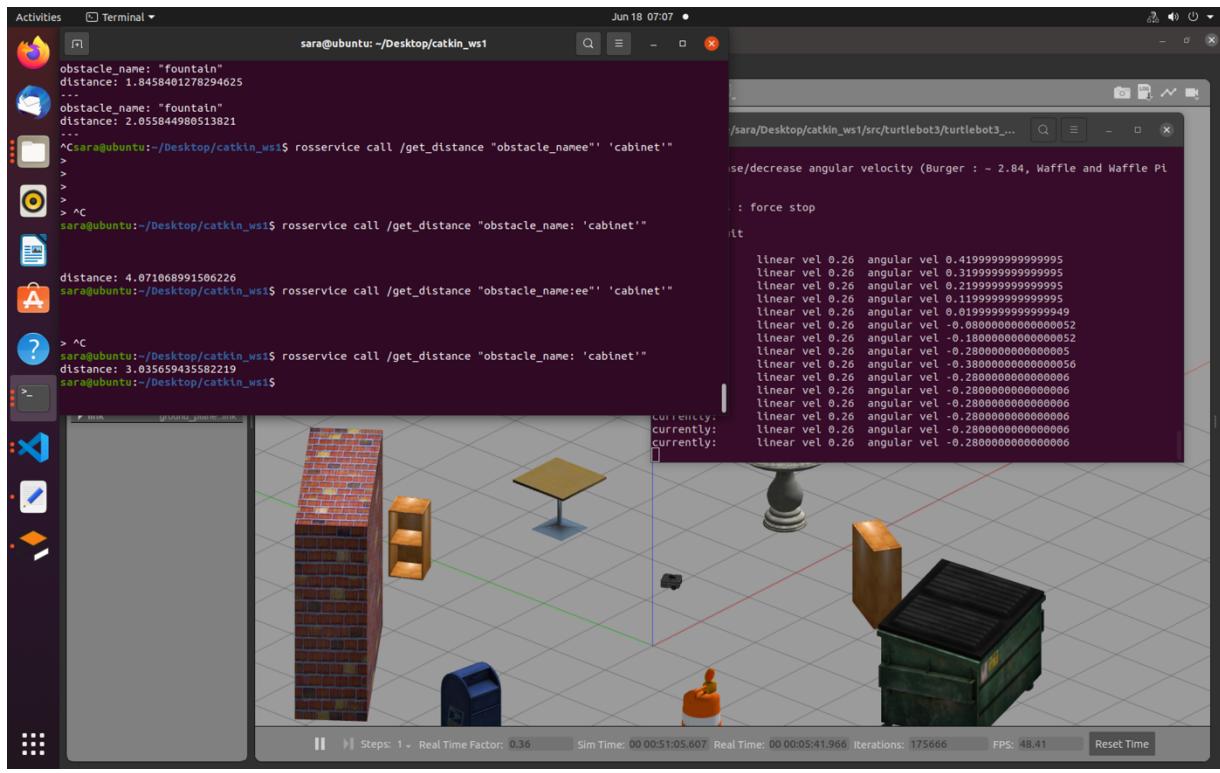
    def get_distance(self, req):
        message = rospy.wait_for_message("odom", Odometry)
        x_pos = message.pose.pose.position.x
        y_pos = message.pose.pose.position.y
        x_dis = self.obstacles[req.obstacle_name][0] - x_pos
        y_dis = self.obstacles[req.obstacle_name][1] - y_pos
        Distance = sqrt((x_dis*x_dis) + (y_dis*y_dis))
        distance1 = distance1()
        distance1.distance = Distance
        return distance1

if __name__ == '__main__':
    find_distance()
```

به این صورت که برای مقدار فاصله ربات از cabinet درخواست زیر داده میشود و مقدار آن به این صورت برگردانده میشود:



پس از تغییر موقعیت ربات دوباره میتوان فاصله آپدیت شده را بدست آورد:



## سناریو اول (ج)

ج) **(۱۰ امتیاز)** هدف از این بخش آشنایی با سنسور LiDAR و استفاده از تاپیک LaserScan است. ابتدا نودی جدید تشکیل داده و ربات را مشابه با قسمت‌های قبل از طریق کیبورد حرکت دهید. در این نود، به صورت مدام فاصله تا نزدیکترین مانع را از طریق تاپیک "ClosestObstacle" دریافت (subscribe) کنید. حال می‌خواهیم در صورتیکه فاصله نزدیکترین مانع تا ربات کمتر از ۱/۵ متر بود، ربات بایستد و سپس زاویه‌ای مناسب بچرخد تا مسیر حرکتش خلاف موقعیت مانع شود. بدین منظور از سنسور LiDAR استفاده می‌نماییم. این سنسور فاصله اطراف ربات را با رزولوشن یک درجه در اختیار شما قرار می‌دهد. با استفاده از این سنسور می‌خواهیم زاویه چرخش مناسب ربات برای فاصله گرفتن از نزدیکترین مانع را محاسبه کنیم. (راهنمایی: به عنوان مثال اگر کمترین مقدار فاصله متعلق به ایندکس ۳۰ آرایه‌ی ranges (خوانده شده از LiDAR) باشد، ربات باید ۱۵۰ درجه به سمت راست خود بچرخد)

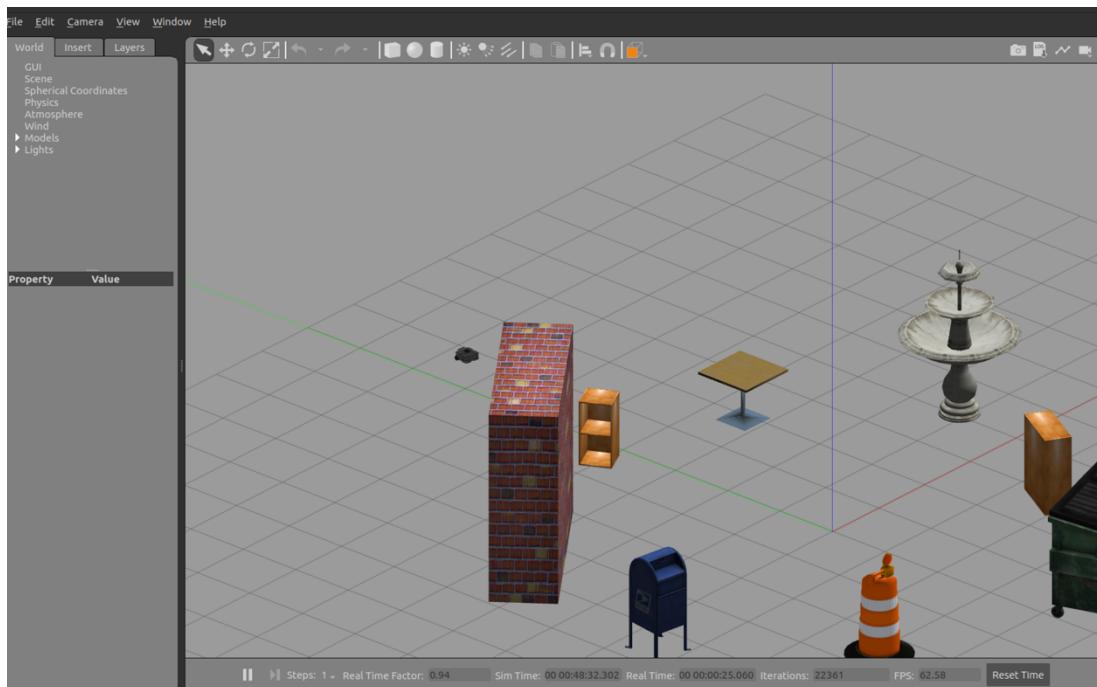
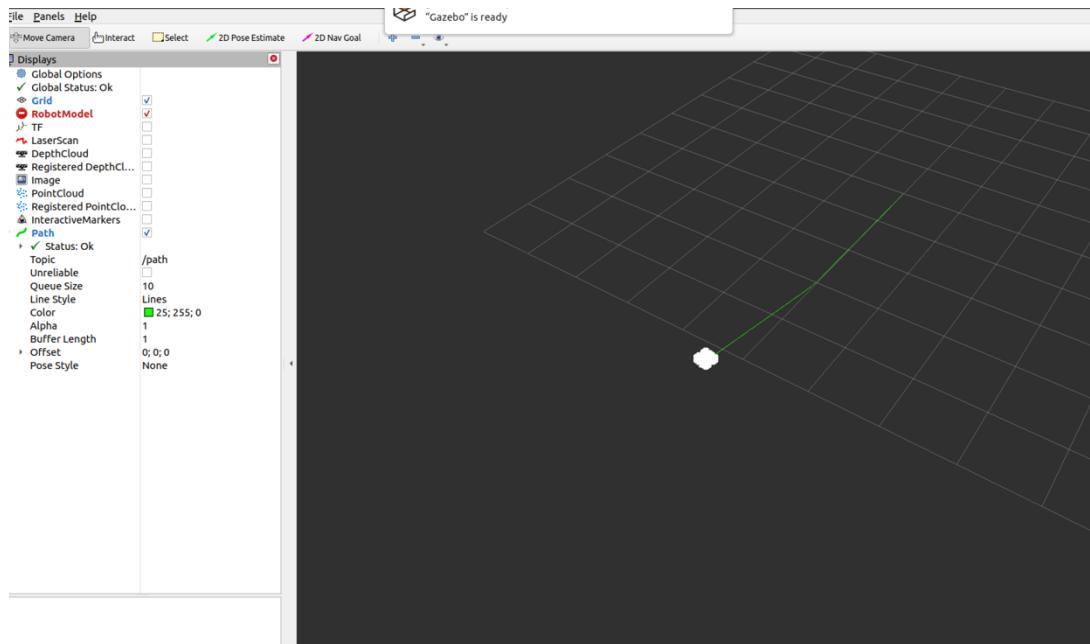
قطعه کد launch این قسمت به شکل زیر است:

```
<launch>
  <node pkg="s1" type="dis.py" name="dis" output="screen">
  </node>
  <node pkg="s1" type="auto_return.py" name="auto_return" output="screen">
  </node>
  <arg name="model" default="$(env TURTLEBOT3_MODEL)" doc="model type [burger, waffle, waffle_pi]"/>
  <arg name="x_pos" default="0.0"/>
  <arg name="y_pos" default="0.0"/>
  <arg name="z_pos" default="0.0"/>
  <arg name="yaw" default="0.25"/>

  <node pkg="s1" type="monitor.py" name="monitor"></node>
  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="world_name" value="$(find s1)/detect_obstacles.world"/>
    <arg name="paused" value="false"/>
    <arg name="use_sim_time" value="true"/>
    <arg name="gui" value="true"/>
    <arg name="headless" value="false"/>
    <arg name="debug" value="false"/>
  </include>

  <param name="robot_description" command="$(find xacro)/xacro --inorder $(find
    | | turtlebot3_description)/urdf/turtlebot3_${arg model}.urdf.xacro" />
  <node pkg="gazebo_ros" type="spawn_model" name="spawn_urdf" args="-urdf -model turtlebot3_${arg model}
    | | -x ${arg x_pos} -y ${arg y_pos} -z ${arg z_pos} -Y 1.57 -param robot_description" />
  <include file="$(find turtlebot_rviz_launchers)/launch/view_robot.launch"/>
</launch>
```

## خروجی برای قسمت ج:



## سناریو دوم (الف)

### سناریوی ۲. دنبال کردن دیوار

در این سوال باید یک الگوریتم دنبال کردن دیوار (wall following) را با کمک سنسور LaserScan پیاده سازی کنید. این الگوریتم ساده برای حرکت ابتدایی در یک محیط ناشناخته مناسب می باشد. **بدین منظور هندزان سوم را حتما مطالعه کنید.**

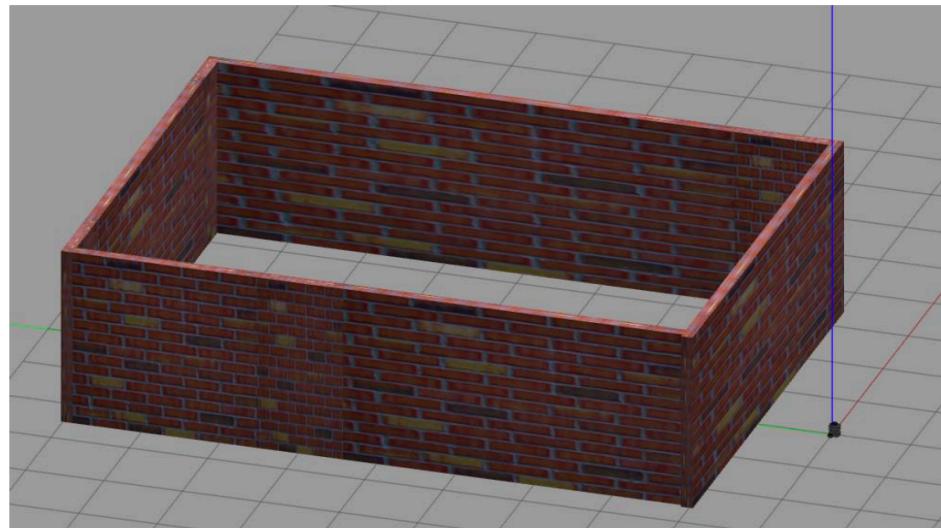
بر اساس این الگوریتم، ربات همواره در کنار دیوار حرکت می کند. ربات می تواند دنبالگر دیوار راست یا چپ باشد. دنبالگر دیوار راست، در هر لحظه دیوار را در سمت راست خود می بیند و حرکت می کند. به این صورت که هنگامی که ربات مستقیم حرکت کرده و دیوار را پیدا نمود، به سمت چپ می پیچد تا دیوار در سمت راست آن قرار بگیرد. اگر در ادامه مسیر ربات در مقابلش به دیوار برسد، و همچنان در راست خود دیوار باشد، به چپ می پیچد. اما هنگامی که دیوار سمت راست پایان می یابد، به راست گردش می کند تا دوباره دیوار در سمت راست خود قرار گیرد و سپس حرکت می کند. ربات دنبالگر چپ نیز بر خلاف این دنبالگر همواره دیوار سمت چپ خودش را دنبال می کند.

در این قسمت شما می توانید دنبالگر راست، و یا چپ را پیاده سازی نمایید. ربات همواره یک فاصله مشخصی را با دیوار حفظ کرده و آن را دنبال می کند. این فاصله را خودتان تعیین کنید. ربات می تواند با سرعت خطی حرکت داشته باشد و زمانی که نیاز به چرخش باشد، بایستد، و بچرخد. دقیقاً چرخش ربات لزوماً ۹۰ درجه نمی باشد و وابسته به فاصله ای که ربات با دیوار دارد می باشد. یعنی چرخش تا زمانی که با دیوار فاصله مشخص داشته باشد، ادامه می یابد.

الف) **(۱۰ امتیاز)** ابتدا ربات را در دنیای square.world به سمت دیوار قرار دهید. ربات باید یک مسیر مستقیم را طی کند تا به یک دیوار برسد. سپس دیوار را دنبال نماید و کل مسیر مربعی را طی کند. مسیر حرکت ربات را بر روی rviz رسم کرده و در گزارش خود، تصویری از آنرا قرار دهید.

نقطه اولیه: (صفر و صفر)

زاویه اولیه: ۹۰ درجه یعنی  $1.57 \text{ rad}$



این کلاس direction را به صورت زیر پیاده سازی میکنیم:

```
#!/usr/bin/python3
from statistics import mean
import rospy, tf
from sensor_messages.message import LaserScan
from nav_messages.message import Odometry
from s1.message import co
from geometry_messages.message import Twist
import numpy as np

class direction():
    def __init__(self) -> None:
        rospy.init_node("auto_return", anonymous=False)
        self.cmd_publisher = rospy.Publisher('/cmd_vel', Twist, queue_size=10)

    def cmd():
        return self.cmd_publisher

    def read_angle(self):
        data = rospy.wait_for_message("/scan", LaserScan)
        return data.ranges.index(min(data.ranges))//4

    def spin(cmd, angle):
        ban = Twist()
        ban.angular.z, ban.linear.x = 0.0, 0.0
        cmd.publish(ban)
        message = rospy.wait_for_message("/odom", Odometry)
        dirc = message.pose.pose.dirc
        _, _, angle1 = tf.transformations.euler_from_quaternion((dirc.x, dirc.y, dirc.z, dirc.w))
        target_angle, twist.angular.z = angle1 + angle, 0.4
        twist = Twist()
        cmd.publish(twist)
        angle2 = find_angle2(target_angle, angle1, Odometry, angle3, angle1)
        cmd.publish(ban)
        rospy.sleep(1)
```

```

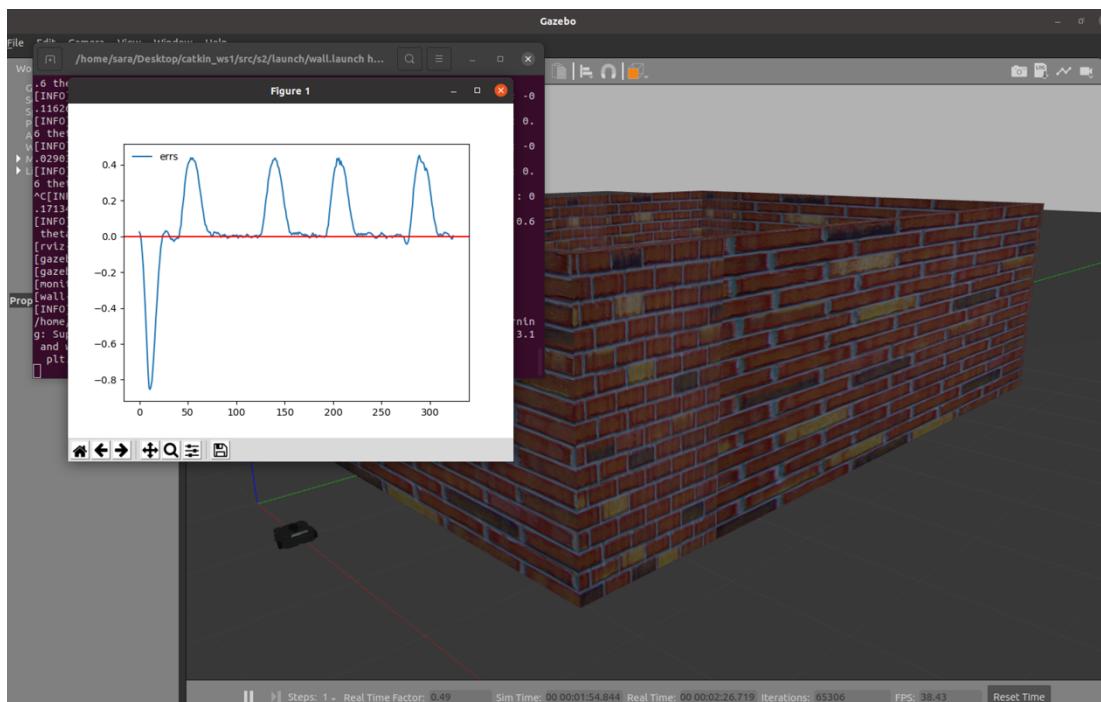
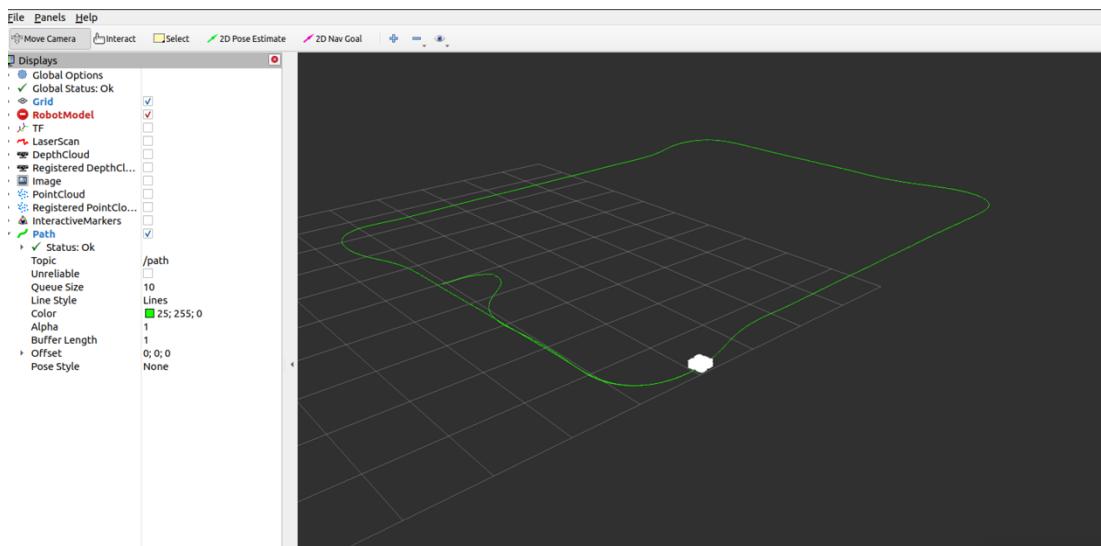
def find_angle2(target_angle, angle1, Odometry, angle3, angle2):
    while target_angle >= abs(angle1 - angle2):
        message = rospy.wait_for_message("/odom" , Odometry)
        dirc = message.pose.pose.dirc
        _, _, angle3 = tf.transformations.euler_from_quaternion((dirc.x ,dirc.y ,dirc.z ,dirc.w))
        if angle3 > 0:
            if angle3 > 0:
                if angle2 > 0:
                    angle2 -= abs(angle3 - angle2)
                else:
                    angle2 -= abs(angle3 + angle2)
            else:
                if angle2 > 0:
                    angle2 -= abs(-angle3 - angle2)
                else:
                    angle2 -= abs(-angle3 + angle2)
        else :
            angle2 = angle3
    return angle2

def find_dir(cmd):
    while True:
        if rospy.is_shutdown():
            return
        closestObstacle = rospy.wait_for_message("/ClosestObstacle", co)
        if closestObstacle.distance < 1.5 :
            spin(cmd, np.deg2rad(read_angle())-10)
            while closestObstacle.distance < 1.5
                twist = Twist()
                twist.angular.z, ban.linear.x = 0.0, 0.4
                cmd.publish(twist)
        else :
            twist = Twist()
            twist.angular.z, ban.linear.x= 0.0, 0.4
            cmd.publish(twist)

if __name__ == '__main__':
    direction = direction()
    cmd = direction.cmd
    direction.find_dir(cmd)

```

در این قسمت هم ربات با در نظر گرفتن فاصله تا دیوار یک بار دور آن را بزند البته در این قسمت باید توجه شود که مقادیر به صورت کنترل شده به ربات داده شوند تا نه با آن برخورد کند نه از مسیر خارج شود(اعداد متناسب داده شده)

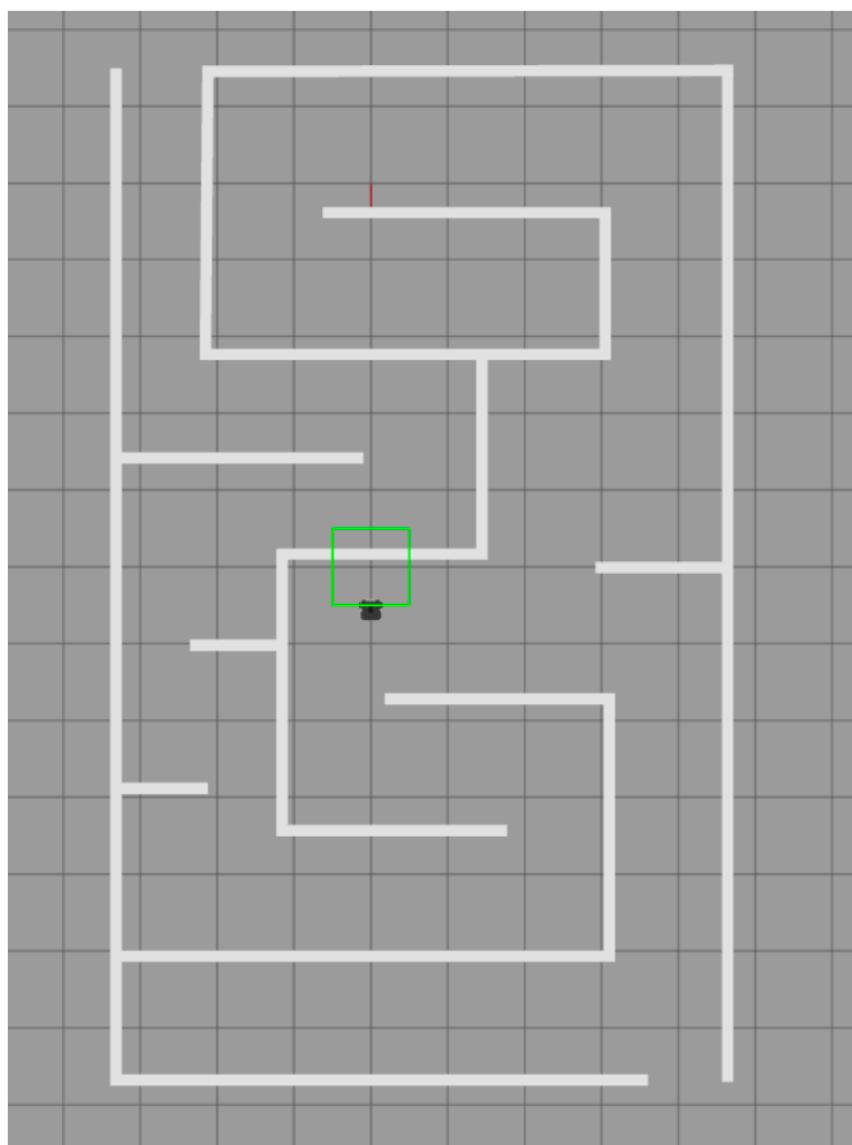


## سناریو دوم (ب)

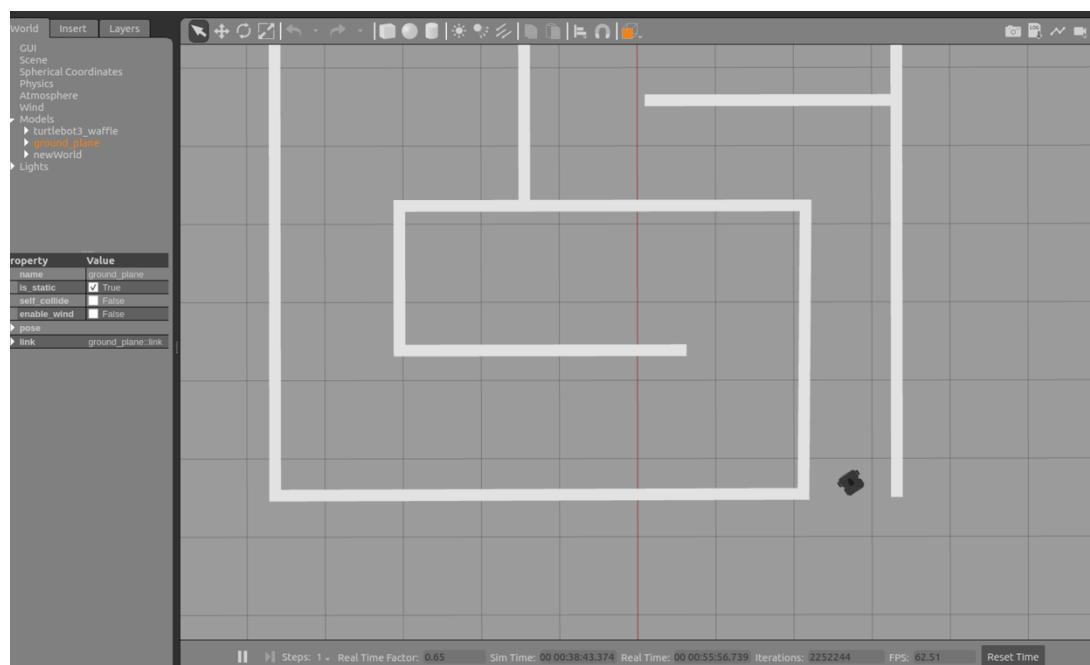
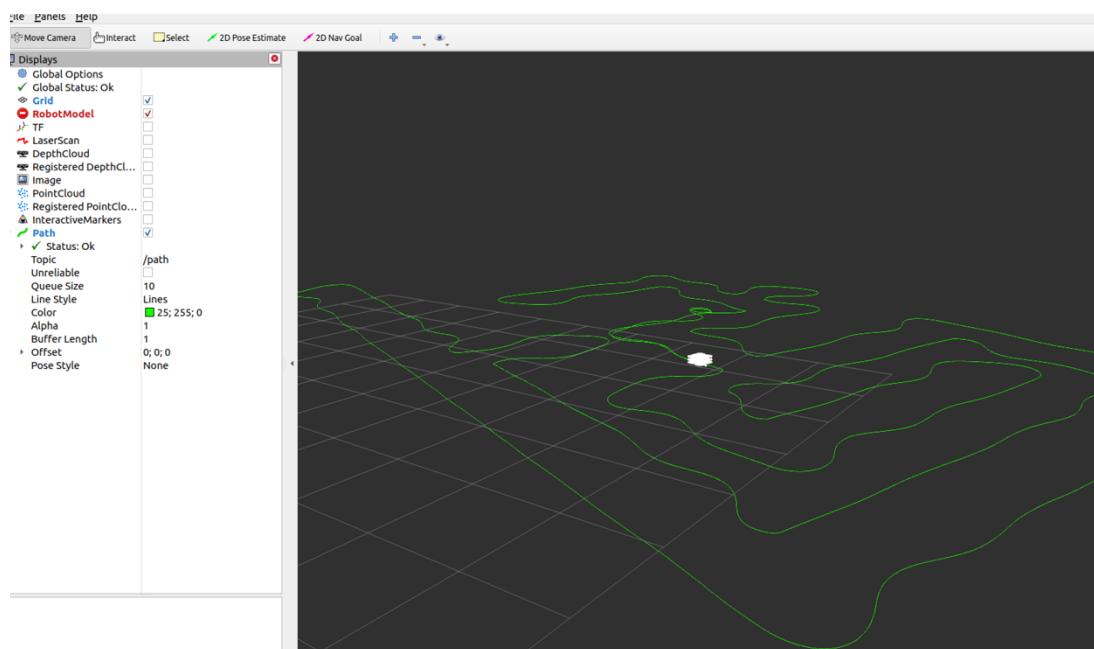
ب) (۱۵ امتیاز) حال دنیای maze.world را باز کنید. در این قسمت ربات شما باید با استفاده از دنبال کردن دیوار، خروجی هزار تو را پیدا کند و از آن خارج شود. ربات داخل هزار تو ظاهر می‌شود و سپس مسیر مستقیمی را طی می‌کند تا دیوار را پیدا کند. سپس همان دیوار را مانند بخش قبلی دنبال می‌کند تا هزار تو را پیمایش کند و در نهایت از یکی از خروجی‌ها خارج شود.

نقطه اولیه: (0, -0.5)

مسیر حرکت ربات را بر روی rviz رسم کرده و در گزارش خود، تصویری از آنرا قرار دهید.



خروجی در قسمت ب به شکل زیر است:



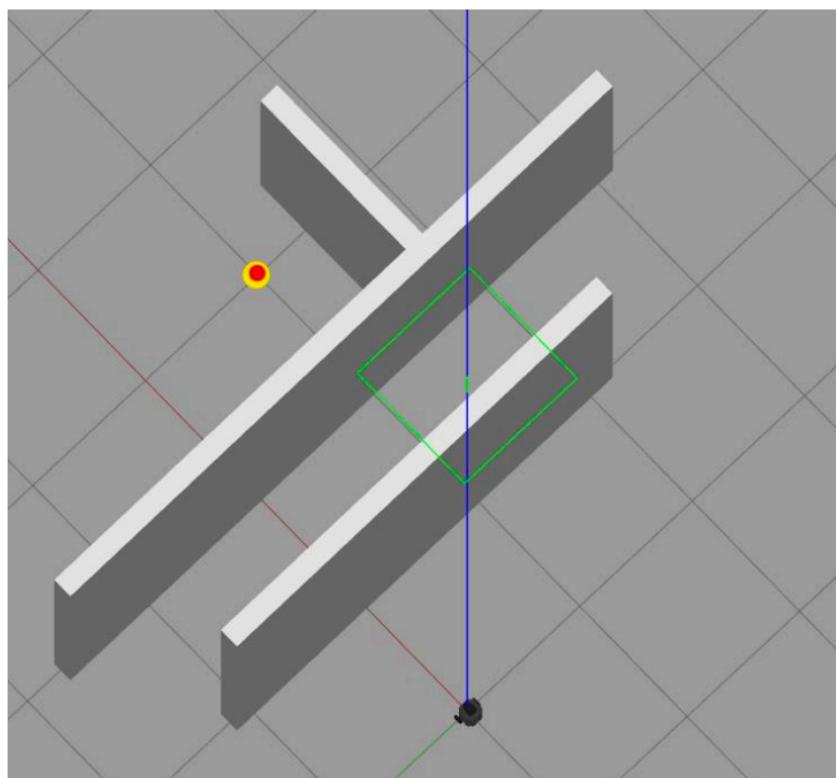
## سناریو دوم (ج)

ج) (۲۰ امتیاز) در دنیای path\_to\_goal.world هدف این است که به نقطه هدف مشخص شده برسید. در مسیر حرکت موانعی موجود می‌باشد. ربات شما در ابتدا و هر زمان که ممکن بود باید به صورت مستقیم به سمت نقطه هدف حرکت کند. هنگامی که در جلوی خود مانع م شاهده نمود، ربات باید از الگوریتم دنبال کردن دیوار استفاده کند و تا زمانی که حرکت مستقیم به سمت هدف میسر نمی‌باشد، دیوار مانع را دنبال کند. این قسمت یک تفاوت اصلی با دو قسمت قبل دارد. در دو قسمت قبل، پس از یافتن دیوار توسط ربات همواره آن را دنبال می‌کند و هر زمان که دیوار تمام بشود، آنقدر به سمت دیوار می‌چرخد تا دوباره دیوار را در همان طرف خود پیدا کند. اما در این بخش، ربات تنها هنگامی که نمی‌تواند مستقیماً به سمت هدف حرکت کند، دیوار را دنبال می‌کند. در غیر این صورت، یعنی در صورتی که مانع در مسیر جرکت ربات نباشد، ربات دیوار را رها کرده و مسیر مستقیم تا هدف را با سرعت خطی پیمایش می‌کند.

مسیر حرکت ربات را بر روی rviz رسم کرده و در گزارش خود، تصویری از آنرا قرار دهید.

نقطه اولیه: (صفر و صفر)

نقطه هدف : (3, -1)



خروجی در این قسمت به شکل زیر است:

