باسمه تعالی

دانشگاه صنعتی امیرکبیر

(پلی تکنیک تهران)

دانشکده مهندسی کامپیوتر

اصول علم ربات

استاد جوانمردی

تمرین سری سوم

تمرین‌های تئوری و گزارش‌کار تمرین‌های عملی

محمد جواد رجبی

9831025

بهار 1401

بخش تئوری)

سوال اول :

دو نمونه سنسور active : sonar و LiDAR و GPS

دو نمونه سنسور passive : Thermostat sensors(برای اندازه گیری دما)

و Spectrometer (طیف سنج) و Accelerometer (شتاب سنج)

بله GPS یک سنسور فعال یا به عبارتی active است چرا که با انتشار و فرستادن امواج به سمت ماهواره‌ها و اندازه گیری آن تا لحظه دریافت پاسخ برای حساب کردن فاصله و از طریق آن پیدا کردن موقعیت ، یک سنسور فعال محسوب می‌شود.

سوال دوم : GPS

دو دلیل عمده وجود دارد که چرا GPS را نمی توان در ساختمان استفاده کرد: قدرت سیگنال کم و دقت پایین.

قدرت سیگنال‌ایی که از ماهواره فرستاده می‌شود با طی مسافت طولانی کم می‌شود و وجود موانعی مثل ساختمان‌ها و دیوارها باعث می‌شود که این سیگنال‌ها حتی از این موانع نتوانند رد بشوند و یا در صورت عبور بسیار ضعیف شده باشند و عملا کارایی خود را از دست داده باشند.

مشکل دوم نیز همانطور که گفته شد دقت پایین است که یک سنسور GPS معمولی تقریبا دقتی با شعاع ۱۰ را دارد که این دقت برای ساختمان که نیاز دقت سانتی‌متری و یا حتی کمتر دارد پاسخ‌گو نیست.

سوال سوم و چهارم :

در ناوبری کور برای پیدا کردن heading می توانیم از قطب‌نما یا شیب سنج یا Gyroscopes استفاده کنیم. استفاده از قطب‌نما که مشخص است و می‌توانیم یک حالت اولیه تعریف کنیم و هر بار که heading را می خواستیم مقدار قطب‌نما را می خوانیم و با آن حالت اولیه مقایسه می‌کنیم و با توجه به تغییرات heading ربات را تععین میکنیم. شیب‌سنج هم جهت حرکت ربات که معمولا heading است را به ما می‌دهد چرا که معمولا یک جسم برای حرکت شیبی در همان راستا می‌گیرد. Gyroscopes نیز جهت را خروجی می‌دهد ولی حتی اگر خروجی نداد می‌توان بوسیله داشتن سرعت زاویه و پوزیشن اولیه ربات heading آن را بدست اورد

بخش عملی)

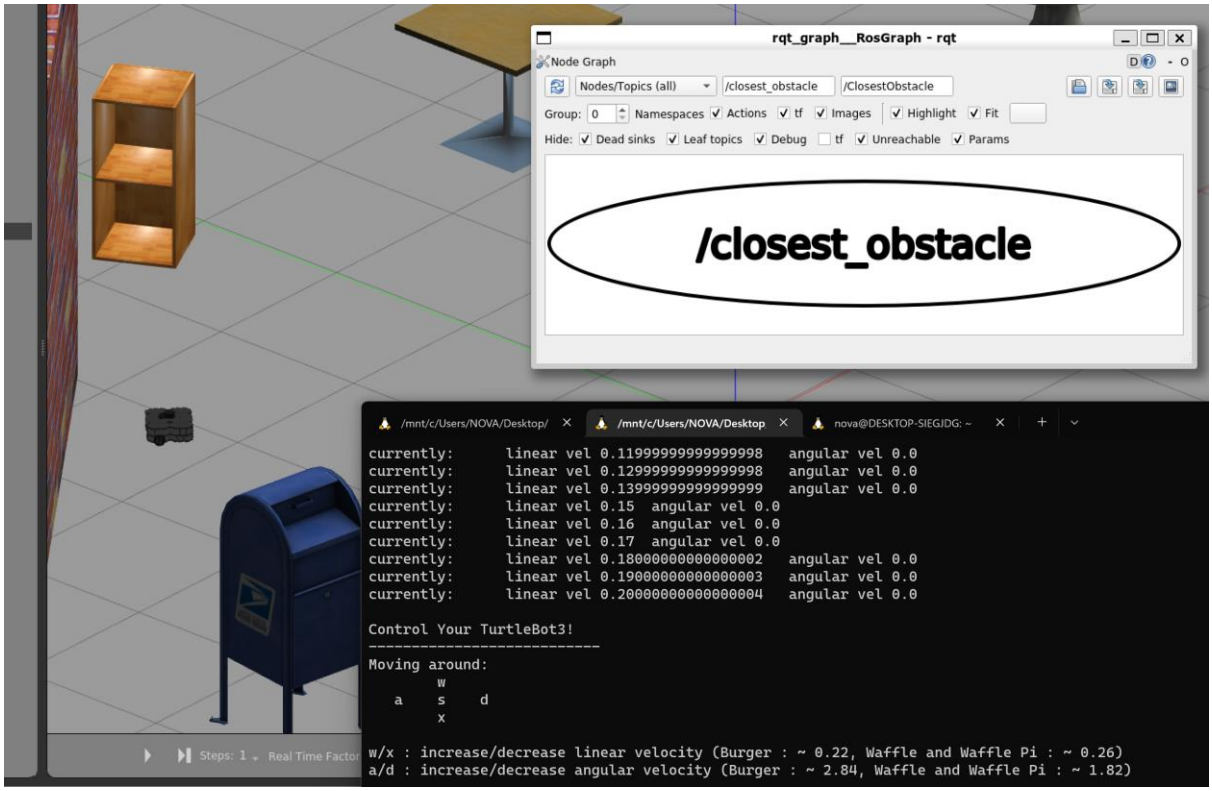
سناریو اول :

قسمت الف)

در این بخش سعی شد تا با پیاده‌سازی تاپیک‌ایی بتوان نزدیک ترین مانع به ربات را در دنیای detect_obstacles را بوسیله این تاپیک به نود ساب‌اسکرایپ کرده برگرداند. همچنین در این سناریو برای کنترل ربات (تعیین سرعت خطی و زاویه‌ای برای حرکت دادن ربات) از نود teleop_key که برای ربات مورد نظر ما پیاده سازی شده است، استفاده می‌کنیم.

```
def find_closest_obstacle(self):

    min_distance = 9999
    closeset_obsetacle = "nothing"

    for i in self.obstacles:

        obstacle_position = self.obstacles[i]
        distance = self.Euclidean_distance(self.myposition ,obstacle_position)

        if distance <= min_distance:
            min_distance = distance
            closeset_obsetacle = i

    return closeset_obsetacle , min_distance

def run(self):

    while not rospy.is_shutdown():

        closeset_obsetacle , min_distance = self.find_closest_obstacle()
        # print(f"obstacle:{closeset_obsetacle} and distance:{min_distance}")
        myobstacle = Obstacle()
        myobstacle.distance = min_distance
        myobstacle.obstacle_name = closeset_obsetacle

        self.publisher.publish(myobstacle)
        self.rate.sleep()
```

همانطور که مشاهده می‌کنید تصویر دو تا از تابع‌های اصلی این نود را نشان می‌دهند که تابع find_closest_obstacle() با داشتن مختصات موانع وظیفه حساب کردن فاصله بین ربات و موانع و برگرداندن نام و فاصله مانع مورد نظر که نزدیک ترین مانع تا ربات را دارد و همچنین تابع run() وظیفه صدا زدن این تابع و انتشار پیام مورد نظر را دارد



برای انتشار پیام مورد نظرمان که شامل نام و فاصله نزدیک ترین مانع به ربات تحت تاپیک Closest_obstacle نیاز تا ما فایل پیام مورد نظرمان را ایجاد کنیم و تنظیمات متناسب با آن را برای هماهنگی با ROS انجام دهیم

HW3_scenario1 > CMakeLists.txt

```
47  ##   * add every package in MSG_DEP_SET to generate_messages(DEPE
48
49  # Generate messages in the 'msg' folder
50  add_message_files(
51    FILES
52    Obstacle.msg
53  )
54
55  # Generate services in the 'srv' folder
56  add_service_files(
57    FILES
58    ObstacleService.srv
59  )
60
```

closest_obstacle.py 1 | Obstacle.msg

HW3_scenario1 > msg > Obstacle.msg

```
1  string obstacle_name
2  float64 distance
```

قسمت ب)

هدف بخش بازنویسی بخش الف با استفاده از سرویس‌هاست. یک سرویس به اسم GetDistance می‌سازیم 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. پس ابتدا نیاز است تا سرویس مورد نظرمان را در ROS تعریف و تنظیمات مورد نیاز برای آن را انجام بدهیم. و در ادامه نود جدید می‌سازیم و نود قسمت الف را بوسیله سرویس بازنویسی می‌کنیم

HW3_scenario1 > srv > ObstacleService.srv

```
1   string obstacle_name
2   ---
3   float64 distance
```

```python
class Detector:

    def __init__(self) -> None:

        rospy.init_node("get_distance" , anonymous=False)
        # rospy.on_shutdown(self.on_shutdown)

        self.odom_subscriber = rospy.Subscriber("/odom" , Odometry , callback=self.odom_call
        rospy.Service("GetDistance" , ObstacleService , self.get_distance)


        self.myposition = (0,0)

        # building the dictionary of obstacle positions
        self.obstacles = dict()
        self.obstacles["bookshelf"] = (2.64, -1.55)
        self.obstacles["dumpster"] = (1.23, -4.57)
        self.obstacles["barrel"]    = (-2.51, -3.08)
        self.obstacles["postbox"] = (-4.47, -0.57)
        self.obstacles["brick_box"] = (-3.44, 2.75)
        self.obstacles["cabinet"] = (-0.45, 4.05)
        self.obstacles["cafe_table"]    = (1.91, 3.37)
        self.obstacles["fountain"] = (4.08, 1.14)
```

```python
def get_distance(self , req):

    obstacle_position = self.obstacles[req.obstacle_name]
    distance = self.Euclidean_distance(self.myposition ,obstacle_position)

    # print(distance)
    return distance
```

قسمت ج)

هدف از این بخش آشنایی و استفاده از سنسور LiDAR و ایجاد تاپیک LaserScan برای تشخیص موانع و کنترل ربات می‌باشد. ابتدا یک نود جدید می‌سازیم و از تاپیک Closest_obstacle برای تشخیص موانع و از تاپیک LaserScan برای چرخیدن ربات و پیدا کردن زاویه مناسب استفاده می‌کنیم. در این قسمت نیز مثل قسمت‌های قبلی برای حرکت ربات از نود teleop_key استفاده می‌کنیم و هنگامی که فاصله‌ی ۱.۵ متری از مانع‌ایی رسیدیم متوقف می‌شویم و ربات می‌چرخد تا مانع پشت سر ربات قرار بگیرد و همچنین ربات بتواند به جلو حرکت کند.

```
def run(self):

    while not rospy.is_shutdown():

        if self.closest_distance < 1.5:

            # print(f"obstacle:{self.closest_obstacle} and distance:{self.closest_distance}"

            self.stop_teleop = True
            self.cmd_publisher.publish(Twist())

            self.turn()
            self.cmd_publisher.publish(Twist())
            self.stop_teleop = False
            sleep(2)
            # print(myangle)
```

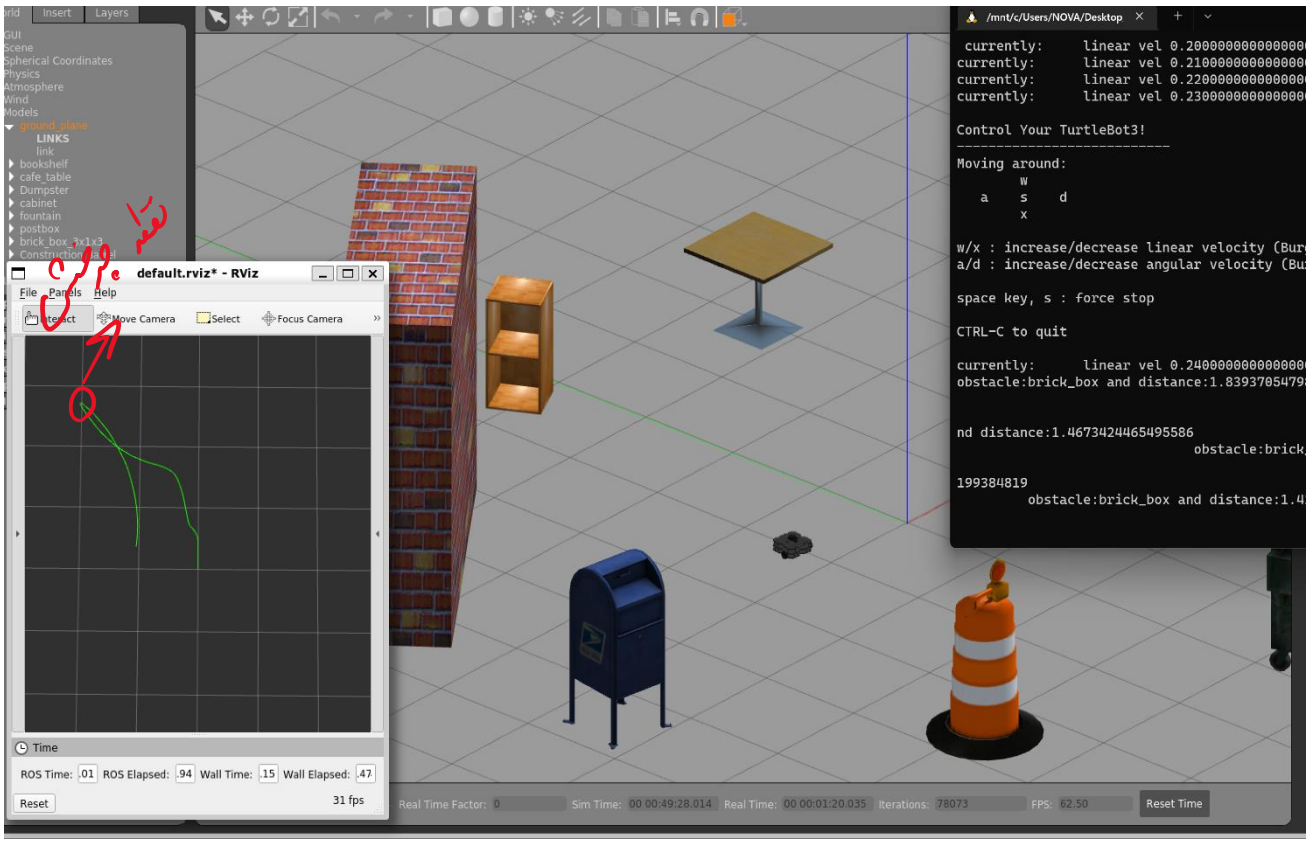
```
def turn(self):

    first_angle= self.closest_angle
    goal_angle = (150 + first_angle) % 360
    # print(first_angle)
    # print(goal_angle)

    twist = Twist()
    twist.angular.z = self.angular_speed
    self.cmd_publisher.publish(twist)

    while True :

        if (goal_angle - 5) <= self.closest_angle <= (goal_angle + 5):
            break
```

لازم به ذکر است چون برای چرخیدن ربات لازم است که تاپیک cmd_vel را ساب‌اسکرایپ کنیم و پیام حرکتی مناسب که شامل سرعت زاویه‌ایی می‌شود را منتشر کنیم و این پیام با پیام‌های منتشر شد از سمت teleop_key تداخل پیدا می‌کند لازم است تا کد teleop_key را متناسب با نیازمان تغییر بدهیم، برای همین تاپیک دیگری را ایجاد می‌کنیم به نام cmd_teleop و نود teleop_key پیام هایش را در این تاپیک منتشر می‌کند و ما در نود خود مان این پیام را در تاپیک cmd_vel منتشر می‌کنیم تا ربات حرکت کند مگر اینکه از قصد نخواهیم که نود teleop_key بتواند ربات را کنترل کند که این کار در هنگامی که ربات به فاصله‌ی ۱.۵ متری مانع می‌رسد و باید بایستد و سپس بچرخد کمک می‌کند و دیگر پیام‌های تاپیک cmd_vel تداخل پیدا نمی‌کنند .

```python
if __name__=="__main__":
    if os.name != 'nt':
        settings = termios.tcgetattr(sys.stdin)

    rospy.init_node('custom_teleop_key')
    pub = rospy.Publisher('cmd_teleop', Twist, queue_size=10)
```

```python
def teleop_callback(self ,msg):

    if self.stop_teleop == False:
        self.cmd_publisher.publish(msg)
```

```python
def __init__(self) -> None:

    rospy.init_node("LiDAR_sensor" , anonymous=False)

    self.obstacle_subscriber = rospy.Subscriber("/ClosestObstacle" , Obstacle , callback=self.obstacle_cal
    self.laser_subscriber = rospy.Subscriber("/scan" , LaserScan , callback=self.laser_callback)
    self.cmd_publisher = rospy.Publisher("/cmd_vel" , Twist , queue_size=10)
    self.teleopcmd_subscriber = rospy.Subscriber("/cmd_teleop" , Twist , callback=self.teleop_callback)
```



سناریو دوم :

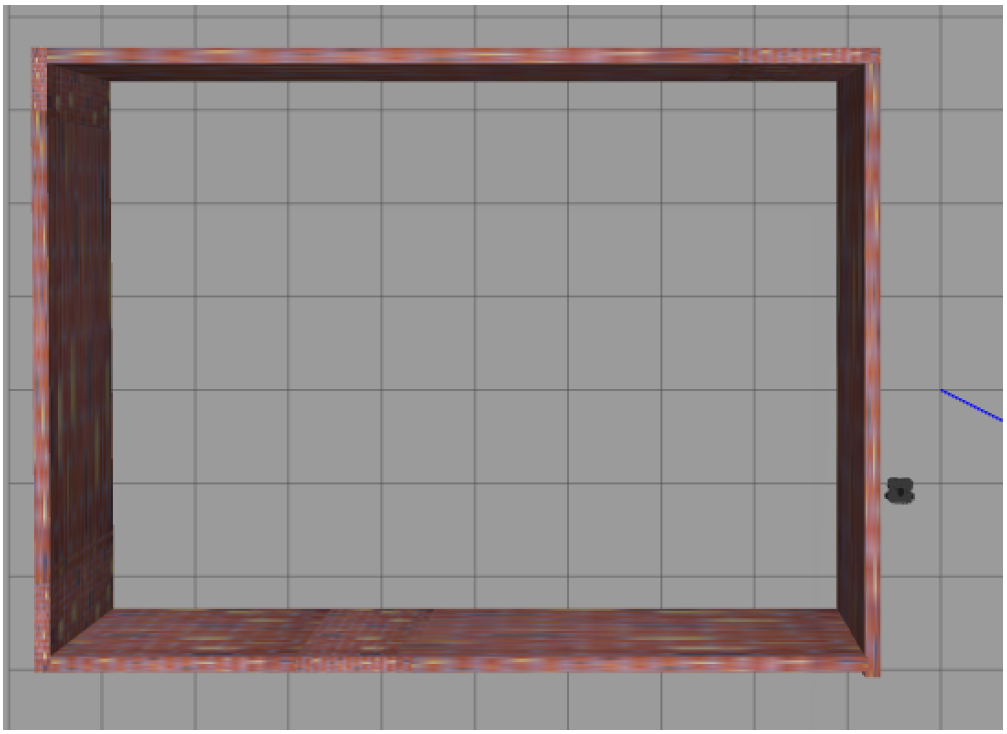
قسمت الف)

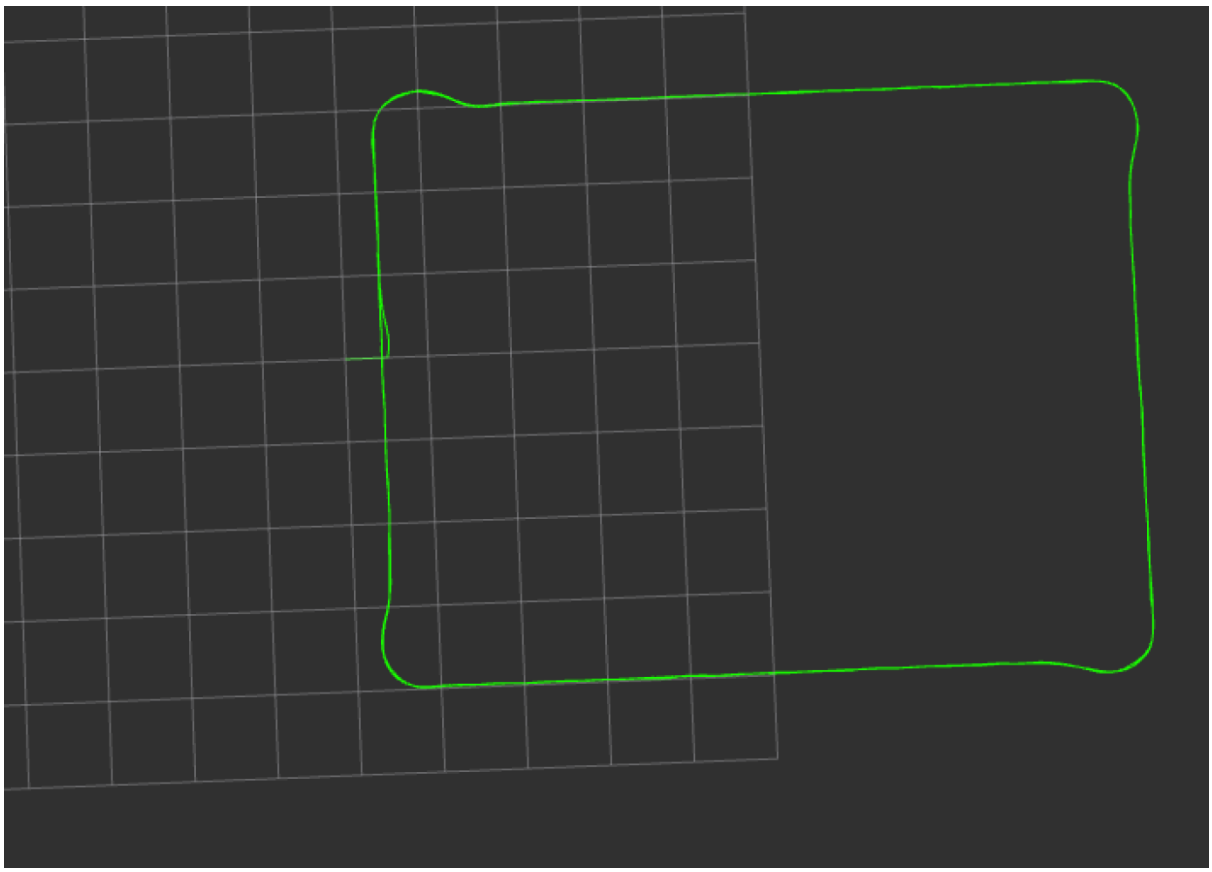
در این قسمت قرار است کدی را بزنیم که ربات فرآیند دنبال کردن دیوار را انجام بدهد ، برای این کار لازم است ابتدا به تاپیک LaserScan ساب‌اسکرایپ کنیم تا بتوانیم فاصله تا دیوار و زاویه ایی که آن نسبت به سنسور ربات دارد را بدست آوریم و بعد یک فاصله مشخص برای دنبال کردن دیوار ، خطای کنترلر را بدست می‌آوریم تا کنترلرمان که به نوعی یک PID کنترلر است با کنترل کردن ربات باعث شود که ربات دیوار را دنبال کند.

```python
def find_wall(self):

    min_distance = 999
    myangle = 0

    laser_msg = rospy.wait_for_message("/scan" , LaserScan)

    for i in range(360):

        if laser_msg.ranges[i] < min_distance:
            myangle = i
            min_distance = laser_msg.ranges[i]

    return myangle , min_distance
```

```python
def wall_following(self):

    print("wall_following has started")
    twist = Twist()
    sum_angular_error = 0
    prev_angular_error = 0

    while not rospy.is_shutdown():

        wall_angle ,wall_distance = self.find_wall()
        # angle_error = (270 - wall_angle) % 180
        distance_error = self.default_distance - wall_distance

        sum_angular_error += (distance_error * self.dt)

        # P = self.kp_a * distance_error + abs(self.kp_l * angle_error)
        P = self.kp_a * distance_error
        I = self.ki_a * sum_angular_error
        D = self.kd_a * (distance_error - prev_angular_error)

        twist.angular.z =  P + I + D
        # twist.linear.x = self.linear_speed - abs(self.kp_l * angle_error)
        twist.linear.x = self.linear_speed

        prev_angular_error = distance_error
        self.cmd_publisher.publish(twist)
        rospy.sleep(self.dt)
```

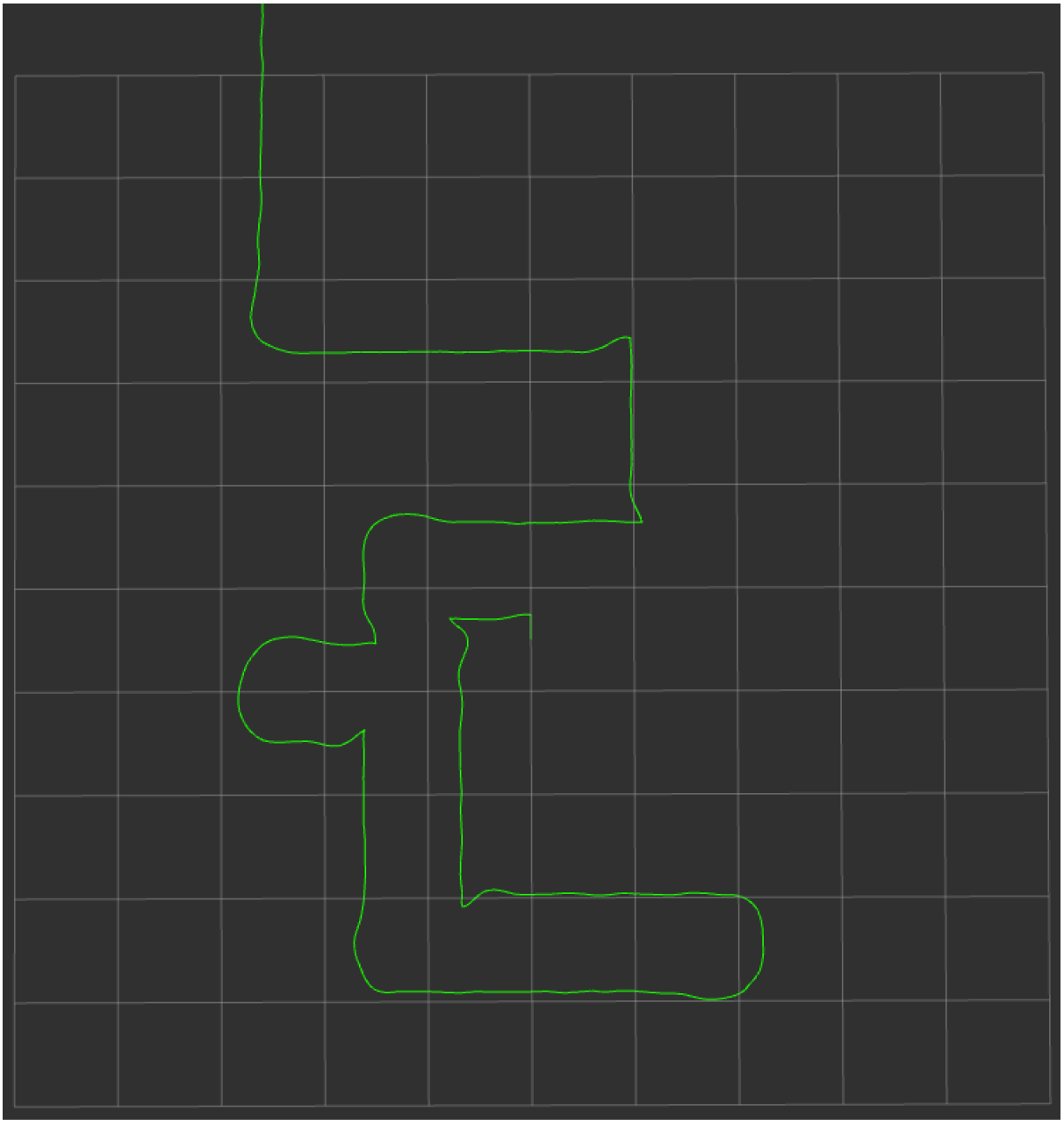
لازم به ذکر است که ربات راست‌گرا می‌باشد و ابتدا از نقطه (۰و۰) با زاویه ۹۰ درجه شروع به حرکت می‌کند تا به دیوار برسد و سپس به دنبال کردن دیوار می‌پردازد.

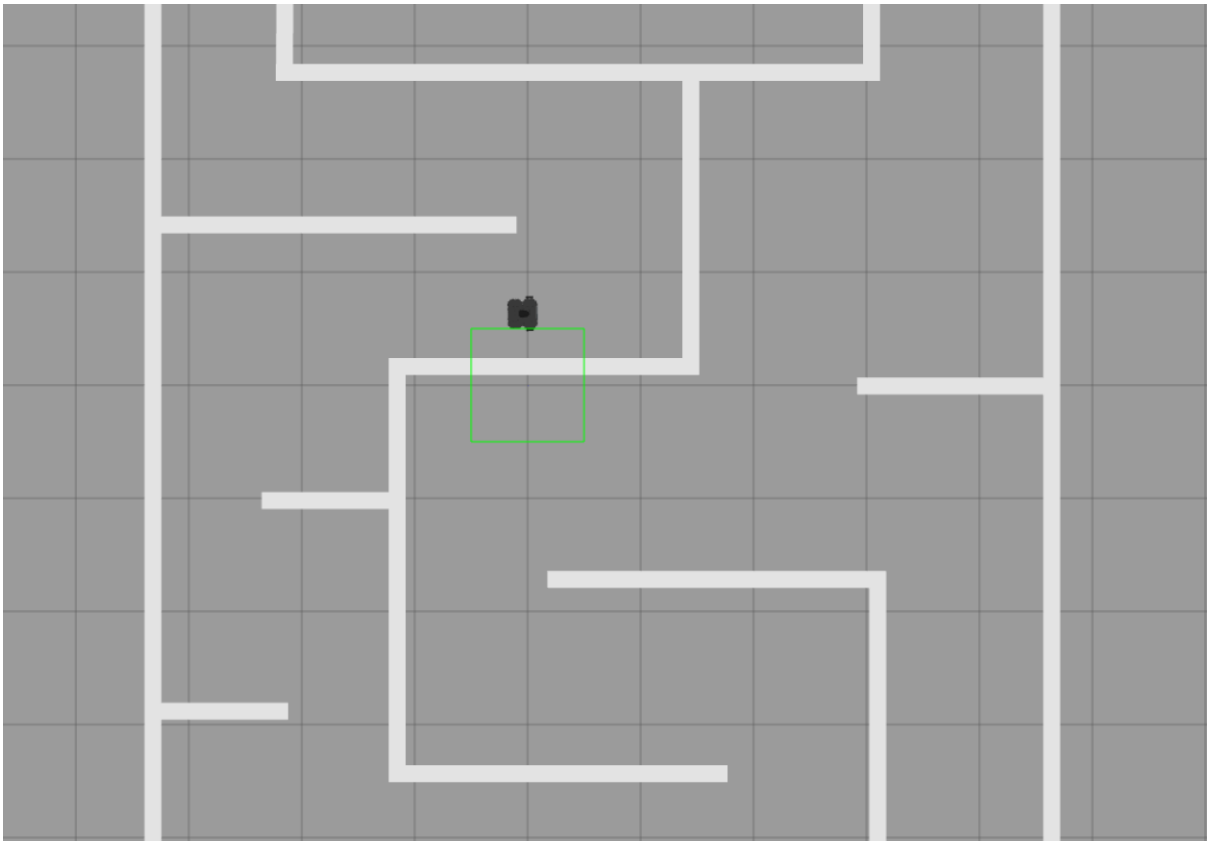




قسمت ب)

در این قسمت قصد دارم تا با الگوریتم دنبال کردن دیوار که در قسمت قبل پیاده‌سازی کردیم ربات را در دنیای maze کنترل کنیم و بتوانیم آن را از maze خارج کنیم. برای خارج شدن از یک maze فقط کافی است که شما نزدیک ترین دیوار را به صورت راست‌گرا دنبال کنید تا از maze خارج شوید، پس می توان گفت الگوریتم‌ایی که در قسمت قبل نوشته‌ایم کافی است و کافی است فقط آن را در صورت نیاز تغییر دهیم.





maze_follower.launch

HW3_scenario2 > launch > maze_follower.launch

```
<launch>


    <node pkg="HW3_scenario2" type="maze_follower.py" name="maze_follower" output="screen"></node>



    <include file="$(find turtlebot3_gazebo)/launch/custom_world.launch">
        <arg name="world_name_file" value="$(find turtlebot3_gazebo)/worlds/maze.world"/>
        <arg name="x_pos" value="-0.5"/>
        <arg name="y_pos" value="0.0"/>
        <arg name="z_pos" value="0.0"/>
        <arg name="yaw" value="0.0"/>
    </include>

    <node pkg="simple_controller" type="monitor.py" name="monitor"></node>
    <node name="rviz" pkg="rviz" type="rviz"/>


</launch>
```

قسمت ج)

در این قسمت قصد داریم ربات را به گونه‌ایی کنترل کنیم که به عبارتی دنبال کننده هدف باشد، یعنی ربات به سمت هدف حرکت می‌کند و هدف را دنبال می‌کند مگر این که به مانع‌ایی برسد مثل دیوار که در این صورت باید دیوار را دنبال کند تا جایی که دوباره بتوان به سمت هدف حرکت کند. برای پیاده‌سازی ما نیاز به دو کنترلر داریم، یکی برای کنترل ربات برای حرکت به سمت هدف که بسیار شبیه کنترلریی است که در تمرین شماره دو پیاده سازی کردیم و دیگری یک کنترلر برای دنبال کردن دیوار که مشابه کنترلر دنبال کننده دیوار قسمت‌های قبل است. پس در کل دو حالت داریم که یکی Goal_following است و دیگری Wall_following ، پس تغییر به موقع درست بین این دو حالت کافی است تا ربات را به هدف برساند.
برای تغییر Goal_following به Wall_following کافی است تا بوسیله سنسور لیزر ، فاصله از دیوار را بدست آوریم و برای مرز تغییر حالت، فاصله ایی را مشخص کنیم. همچنین برای تغییر Wall_following به Goal_following هم با توجه به این که هدف‌مان در این مسئله یک نقطه ثابت است و چیزی در دنیای مورد نظرمان تغییر نمی کند می توانیم از جهت ربات برای تغییر حالت گفته شده استفاده کنیم.
همچنین باید توجه داشت که ربات در دنیای path_to_goal است و نقطه هدف (۱- و۳) می‌باشد.

```
def run(self):

    while not rospy.is_shutdown():

        self.goal_following()
        self.turn_left()
        self.wall_following()
```