به نام خدا

دانشگاه صنعتی امیرکبیر

دانشکده مهندسی کامپیوتر

اصول علم ربات

تمرین سری .....

نام و نام خانوادگی	امیررضا طربخواه
شماره دانشجویی	۹۸۳۱۰۴۱
تاریخ ارسال گزارش	۱۴۰۱/۳/۲۷

فهرست گزارش سوالات (لطفاً پس از تکمیل گزارش، این فهرست را به‌روز کنید.)

سوال ۱ تا ۴

امیرضا طرفخواه 9831041

۱) سنسورهای Compass و Gyroscopes، سنسورهای Passive و GPS و Reflective sensors ها Active هستند. دلیل Active بودن GPS هم این است که از خودش پرتو یا انرژی ساطع می کند و از اطلاعات بدست آمده از ماهواره ها (محیط) می فهمد که دقیقا در کجا قرار دارد.

۲) GPS، چون دقت سنجش GPS به اندازه ای نیست که بتواند حرکت ماهر خانه را به خوبی تشخیص دهد و دارای خطای زیاد است

۳) در ناوبری کور از قطب نما برای پیدا کردن Heading ربات حول z و برای بدست آوردن θz از شیب سنج استفاده می کنیم

۴) به این صورت که جهت فعلی دوچرخه را در شروع کار می خواند.

Scanned with CamScanner

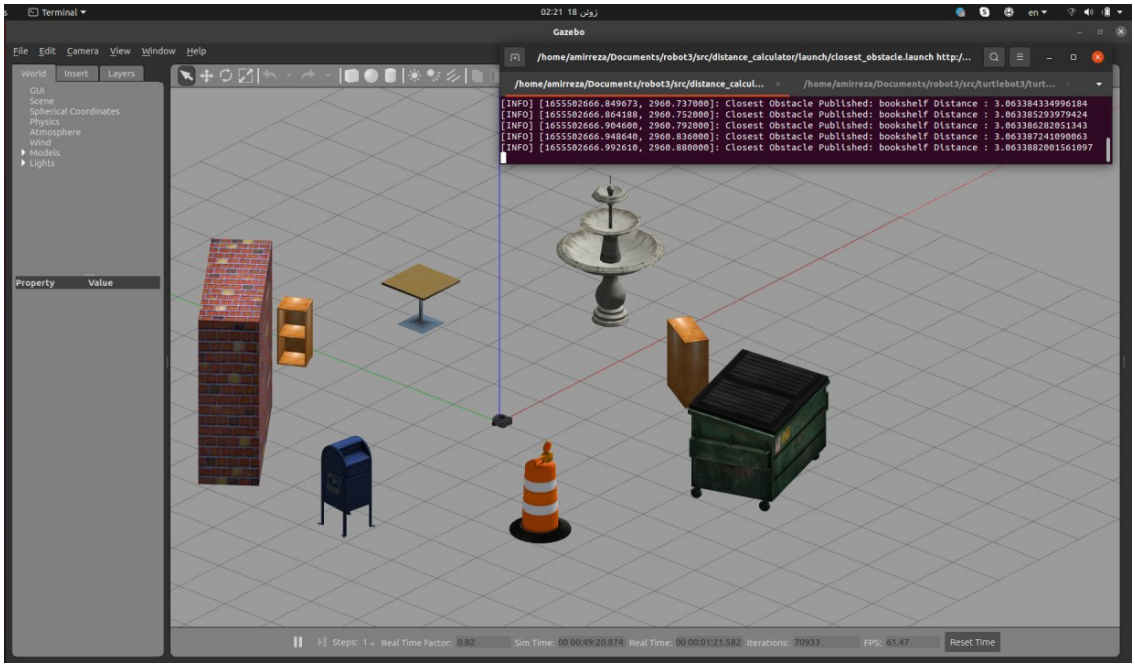
بخش عملی:

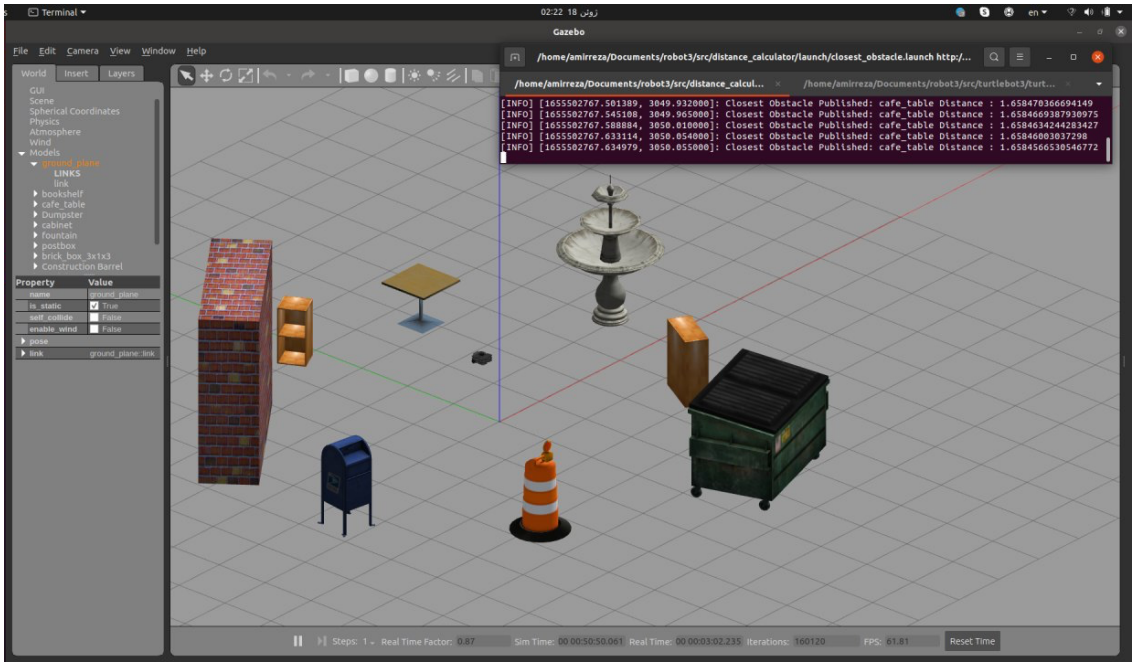
بخش اول:

کلیت این قسمت این است که ما فاصله نزدیک‌ترین جسم را از ربات را بیابیم. در قسمت الف این کار را با msg و در قسمت ب با استفاده از srv این کار را انجام میدهیم. در قسمت آخر یک کنترل کننده قرار میدهیم که وقتی فاصله ربات از جسم از ۱.۵ کمتر شد، شروع به چرخیدن کند. برای ذخیره محل موانع از یک دیکشنری که value آن یک تاپل است کمک میگیریم. نکته قابل توجه این است که در تمامی مراحل کنترل ربات را به صورت دستی داریم. داریم:

الف)

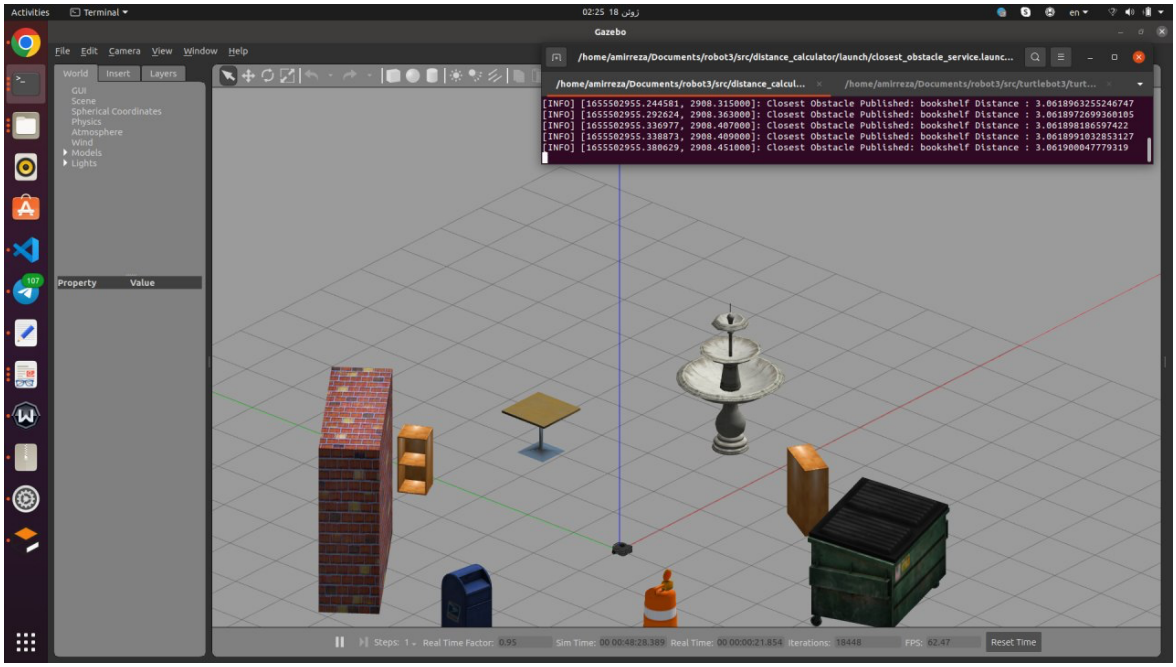
```python
#!/usr/bin/python3

import rospy
from nav_msgs.msg import Odometry
from distance_calculator.msg import Obstacle
import math

class ClosestObstacle:

    def __init__(self) -> None:
        rospy.init_node("minimum_distance_node" , anonymous=True)
        self.odom_subscriber = rospy.Subscriber("/odom" , Odometry , callback = self.calculate_distance)
        self.obstacle_publisher = rospy.Publisher('/ClosestObstacle' , Obstacle , queue_size=10)

        # obstacle details
        self.obstacles_dictionary = {
            "bookshelf" : (2.64, -1.55),
            "dumpster" :  (1.23, -4.57),
            "barrel" :    (-2.51, -3.08),
            "postbox" :   (-4.47, -0.57),
            "brick_box" : (-3.44, 2.75),
            "cabinet" :   (-0.45, 4.05),
            "cafe_table": (1.91, 3.37),
            "fountain" :  (4.08, 1.14)
        }

    def calculate_distance(self,odom):
        x_position = odom.pose.pose.position.x
        y_position = odom.pose.pose.position.y
        minimum_diststance = 10 ** 6
        for this_obstacle,position in self.obstacles_dictionary.items():
            distance = math.sqrt(((x_position - position[0]) ** 2) + ((y_position - position[1]) ** 2))
            if distance < minimum_diststance:
                closest = this_obstacle
                minimum_diststance = distance
        obstacle = Obstacle()
        obstacle.obstacle_name = closest
        obstacle.distance = minimum_diststance
        rospy.loginfo(f"Closest Obstacle Published: {closest} Distance : {minimum_diststance}")
        self.obstacle_publisher.publish(obstacle)

if __name__ == "__main__":
    start_node = ClosestObstacle()
    rospy.spin()
```


Terminal
Gazebo
File Edit Camera View Window Help
World Insert Layers
/home/amirreza/Documents/robot3/src/distance_calculator/launch/closest_obstacle.launch http:/...
[INFO] [1655502666.849673, 2960.737000]: Closest Obstacle Published: bookshelf Distance : 3.063384334996184
[INFO] [1655502666.864188, 2960.752000]: Closest Obstacle Published: bookshelf Distance : 3.063385293979424
[INFO] [1655502666.904600, 2960.792000]: Closest Obstacle Published: bookshelf Distance : 3.063386282051343
[INFO] [1655502666.948640, 2960.836000]: Closest Obstacle Published: bookshelf Distance : 3.063387241090063
[INFO] [1655502666.992610, 2960.880000]: Closest Obstacle Published: bookshelf Distance : 3.063388200156109
Property Value
Reset Time


Terminal
Gazebo
File Edit Camera View Window Help
World Insert Layers
/home/amirreza/Documents/robot3/src/distance_calculator/launch/closest_obstacle.launch http:/...
[INFO] [1655502767.501389, 3049.932000]: Closest Obstacle Published: cafe_table Distance : 1.658478366694149
[INFO] [1655502767.545108, 3049.965000]: Closest Obstacle Published: cafe_table Distance : 1.658466938793097 5
[INFO] [1655502767.588884, 3050.010000]: Closest Obstacle Published: cafe_table Distance : 1.658463424428342 7
[INFO] [1655502767.633114, 3050.054000]: Closest Obstacle Published: cafe_table Distance : 1.658460030372 98
[INFO] [1655502767.634979, 3050.055000]: Closest Obstacle Published: cafe_table Distance : 1.658456653054677 2
Property Value
Reset Time

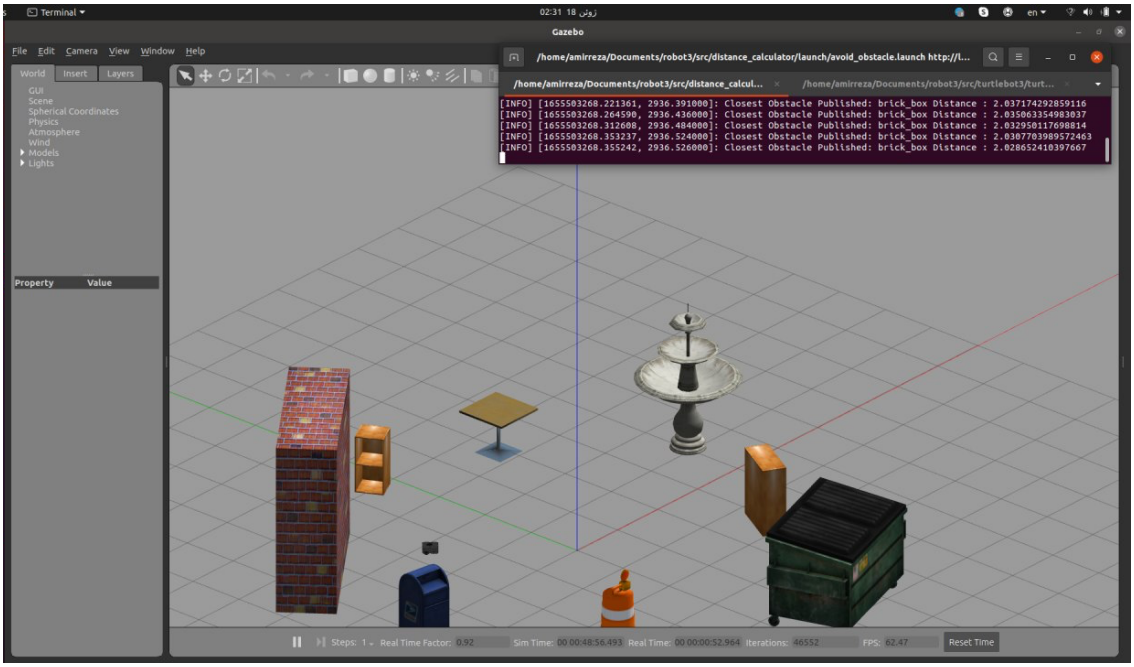
ب)

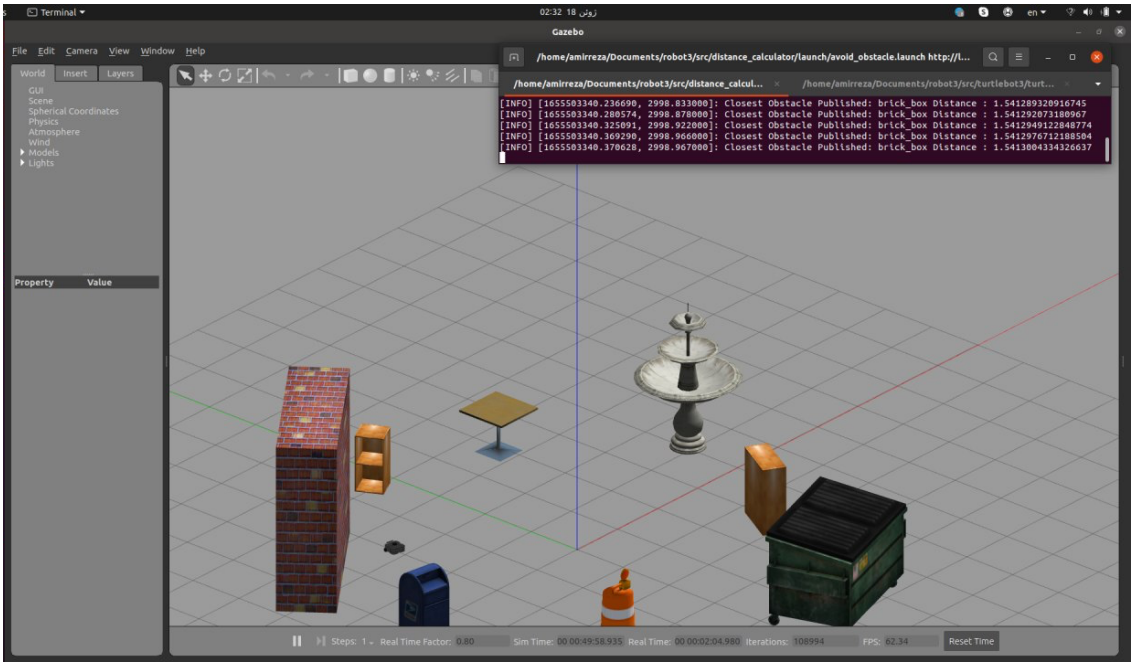
```python
#!/usr/bin/python3

import rospy
from distance_calculator.msg import Obstacle
from distance_calculator.srv import GetDistance

class ClosestObstacle:

    def __init__(self) -> None:
        rospy.init_node("minimum_distance_node" , anonymous=True)
        # prepare service
        rospy.wait_for_service('get_distance')
        self.get_distance_service = rospy.ServiceProxy('get_distance', GetDistance)
        self.obstacle_publisher = rospy.Publisher('/ClosestObstacle' , Obstacle , queue_size=10)

        # obstacle details
        self.obstacles_dictionary = {
            "bookshelf" : (2.64, -1.55),
            "dumpster" :  (1.23, -4.57),
            "barrel" :    (-2.51, -3.08),
            "postbox" :   (-4.47, -0.57),
            "brick_box" : (-3.44, 2.75),
            "cabinet" :   (-0.45, 4.05),
            "cafe_table": (1.91, 3.37),
            "fountain" :  (4.08, 1.14)
        }

    def calculate_distance(self):
        while True :
            minimum_diststance = 10 ** 6
            for this_obstacle in self.obstacles_dictionary.keys():
                res = self.get_distance_service(this_obstacle)
                current_distance = res.distance
                if current_distance < minimum_diststance:
                    closest = this_obstacle
                    minimum_diststance = current_distance
            obstacle = Obstacle()
            obstacle.obstacle_name = closest
            obstacle.distance = minimum_diststance

            rospy.loginfo(f"Closest Obstacle Published: {closest} Distance : {minimum_diststance}")
            self.obstacle_publisher.publish(obstacle)

if __name__ == "__main__":
    startNode = ClosestObstacle()
    startNode.calculate_distance()
```


Activities
Terminal
Gazebo
File Edit Camera View Window Help
World Insert Layers
GUI
Scene
Spherical Coordinates
Physics
Atmosphere
Wind
Models
Lights
Property Value
/home/amirreza/Documents/robot3/src/distance_calculator/launch/closest_obstacle_service.launc...
/home/amirreza/Documents/robot3/src/distance_calcul...
/home/amirreza/Documents/robot3/src/turtlebot3/turt...
[INFO] [1655502955.244581, 2908.315000]: Closest Obstacle Published: bookshelf Distance : 3.0618963255246747
[INFO] [1655502955.292624, 2908.363000]: Closest Obstacle Published: bookshelf Distance : 3.0618972699360185
[INFO] [1655502955.336977, 2908.407000]: Closest Obstacle Published: bookshelf Distance : 3.061898186597422
[INFO] [1655502955.338873, 2908.409000]: Closest Obstacle Published: bookshelf Distance : 3.0618991032853127
[INFO] [1655502955.380629, 2908.451000]: Closest Obstacle Published: bookshelf Distance : 3.061900047779319
Steps: 1
Real Time Factor: 0.95
Sim Time: 00 00:48:28.389
Real Time: 00 00:00:21.854
Iterations: 18448
FPS: 62.47
Reset Time

ج)

```python
#!/usr/bin/python3
import rospy
from nav_msgs.msg import Odometry
from distance_calculator.msg import Obstacle
from math import radians,pi
from sensor_msgs.msg import LaserScan
import numpy as np
import tf
from geometry_msgs.msg import Twist

class AvoidObstacleNode:

    def __init__(self) -> None:
        rospy.init_node("turn_obstacle_node" , anonymous=True)
        self.cmd_publisher = rospy.Publisher('/cmd_vel' , Twist , queue_size=10)
        self.angular_speed=0.1

    def angular_error(self, diff):
        if diff < -pi:
            return diff+ 2 * pi
        if diff > pi:
            return diff - 2 * pi
        return diff

    def get_heading(self):
        msg = rospy.wait_for_message("/odom" , Odometry)
        orientation = msg.pose.pose.orientation
        self.pose_x= msg.pose.pose.position.x
        self.pose_y= msg.pose.pose.position.y
        roll, pitch, yaw = tf.transformations.euler_from_quaternion((
            orientation.x ,orientation.y ,orientation.z ,orientation.w
        ))
        self.yaw = yaw
        return yaw

    def rotate(self,remaining):
        prev_angle = self.get_heading()
        twist = Twist()
        twist.angular.z = self.angular_speed
        self.cmd_publisher.publish(twist)

        # rotation loop
        while remaining >= 0.001:
            current_angle = self.get_heading()
            delta = abs(prev_angle - current_angle)
            delta=self.angular_error(delta)
            remaining -= delta
            prev_angle = current_angle
        self.cmd_publisher.publish(Twist())
        rospy.sleep(1)

    def check_distance(self):
        while True:
            obs= rospy.wait_for_message('/ClosestObstacle' , Obstacle )
            if obs.distance <= 1.5:

                twist = Twist()
                twist.linear.x = 0
                self.cmd_publisher.publish(twist)
                rospy.sleep(1)

                msg = rospy.wait_for_message("/scan" , LaserScan, timeout=1)
                ranges=np.array(msg.ranges)
                min_ind=np.argmin(ranges)

                remaining = self.angular_error(radians(180-min_ind))
                self.rotate(remaining)

if __name__ == "__main__":
    startNode = AvoidObstacleNode()
    startNode.check_distance()
```





بخش دوم:

با استفاده از فایل launch و include کردن محیط را لود میکنیم. برای قرار دادن ربات در نقطه مشخص و اعمال پارامترها از یک دیکشنری در پایتون استفاده کرده و با استفاده از پارامتر mode که از طریق فایل launch به کدمان پاس میدهیم، تشخیص میدهیم که چه مقادیری را اعمال کنیم.

```python
#!/usr/bin/python3

import math
# from turtle import position
import rospy
import numpy as np
import tf
from geometry_msgs.msg import Twist
from sensor_msgs.msg import LaserScan
from nav_msgs.msg import Odometry
import matplotlib.pyplot as plt

class PIDController():


    def __init__(self):

        rospy.loginfo("Start making")

        self.cmd_vel = rospy.Publisher('/cmd_vel', Twist, queue_size=5)

        rospy.init_node('square', anonymous=False)
        def shutdown():
            rospy.loginfo("Stopping the robot...")
            self.cmd_vel.publish(Twist())
            plt.plot(list(range(len(self.errs))), self.errs, label='errs')
            plt.axhline(y=0,color='R')
            plt.draw()
            plt.legend(loc="upper left", frameon=False)
            plt.savefig(f"errs_{self.k_p}_{self.k_d}_{self.k_i}.png")
            plt.show()

            rospy.sleep(1)
        rospy.on_shutdown(shutdown)

        self.mode = rospy.get_param("/square/mode")
        mode = self.mode

        modes = {}
        modes["square"] = [0, 0.8, 10, 0.4, 1]
        modes["maze"] = [0, 1.5, 20, 0.2, 0.5]
        modes["path_to_goal"] = [0, 1.5, 15, 0.3, 0.5]

        self.k_i = modes[mode][0]
        self.k_p = modes[mode][1]
        self.k_d = modes[mode][2]
        self.v = modes[mode][3]
        self.D = modes[mode][4]

        self.goalx = 3
        self.goaly = -1

        self.dt = 0.005
        rate = 1/self.dt

        self.r = rospy.Rate(rate)
        self.errs = []
        rospy.loginfo("Finished")

    def distance_to_goal(self, odometry):
        position = odometry.pose.pose.position
        return math.sqrt((self.goalx - position.x)**2 + (self.goaly - position.y)**2)

    def current_heading(self, msg: Odometry, mode):
        if mode == "current":
            orientation = msg.pose.pose.orientation
            roll, pitch, yaw = tf.transformations.euler_from_quaternion((
                orientation.x ,orientation.y ,orientation.z ,orientation.w
            ))
            return float("{:.2f}".format(yaw))
        else:
            position = msg.pose.pose.position
            return float("{:.2f}".format(np.arctan2((self.goaly - position.y), (self.goalx - position.x))))

    def angle_difference(self, odometry):
        goal_heading = self.current_heading(odometry, "goal")
        current_heading = self.current_heading(odometry, "current")

        if current_heading > 0:
            sign = -1 if (current_heading - pi < goal_heading < current_heading) else +1
        else:
            sign = +1 if (current_heading + pi > goal_heading > current_heading) else -1
        return sign * (pi - abs(abs(current_heading - goal_heading) - pi))

    def window_interval(self, diff, r):
        angle = int(diff * 180 / pi)
        left_w = angle - r
        if left_w < -180:
            left_w += 360
        right_w = angle + r
        if right_w > 180:
            right_w -= 360
        return left_w, right_w

    def is_window_clear(self, laser_scan, odometry):
        laser_range = laser_scan.ranges
        distance = self.distance_to_goal(odometry)

        diff = self.angle_difference(odometry)
        left_i, right_i = self.window_interval(diff, 20)

        if left_i * right_i > 0:
            return True if min(laser_range[left_i:right_i]) > distance else False
        else:
            return True if min(laser_range[left_i:] + laser_range[:right_i]) > distance else False
```

```python
    def window_interval(self, diff, r):
        angle = int(diff * 180 / pi)
        left_w = angle - r
        if left_w < -180:
            left_w += 360
        right_w = angle + r
        if right_w > 180:
            right_w -= 360
        return left_w, right_w

    def is_window_clear(self, laser_scan, odometry):
        laser_range = laser_scan.ranges
        distance = self.distance_to_goal(odometry)

        diff = self.angle_difference(odometry)
        left_i, right_i = self.window_interval(diff, 20)

        if left_i * right_i > 0:
            return True if min(laser_range[left_i:right_i]) > distance else False
        else:
            return True if min(laser_range[left_i:] + laser_range[:right_i]) > distance else False

    def follow_wall(self):

        sum_i_theta = 0
        previous_error = 0

        move = Twist()
        move.angular.z = 0
        move.linear.x = self.v

        while not rospy.is_shutdown():
            self.cmd_vel.publish(move)

            laser_scan = rospy.wait_for_message("/scan", LaserScan)
            odometry = rospy.wait_for_message("/odom", Odometry)

            if self.mode == "path_to_goal" and self.distance_to_goal(odometry) < 0.2:
                rospy.signal_shutdown("goal is reached")

            if self.mode == "path_to_goal":
                reachable = self.is_window_clear(laser_scan, odometry)
            else:
                reachable = False

            if reachable:
                stop = Twist()
                self.cmd_vel.publish(stop)

                while abs(diff) > 0.1:
                    rotation = Twist()
                    rotation.angular.z = 0.1 if diff > 0 else -0.1
                    self.cmd_vel.publish(rotation)
                    odometry = rospy.wait_for_message("/odom", Odometry)
                    diff = self.angle_difference(odometry)

                self.cmd_vel.publish(stop)
                go = Twist()
                go.linear.x = 0.05
                self.cmd_vel.publish(go)
                continue

            ## ELSE
            if self.mode == "test" or self.mode == "a":
                d = min(laser_scan.ranges[:180]) # return robot distance to wall
                err = d - self.D
            else:
                front_distance = min(laser_scan.ranges[:35])
                err = min([front_distance, min(laser_scan.ranges[35:180])]) - self.D   # return distance from side wall

            if err == float('inf'):
                err = 10
            else:
                self.errs.append(err)

            sum_i_theta += err * self.dt

            P = self.k_p * err
            I = self.k_i * sum_i_theta
            D = self.k_d * (err - previous_error)

            move.angular.z = P + I + D
            previous_error = err

            if self.mode == "test" or self.mode == "a":
                move.linear.x = self.v
            else:
                if front_distance < self.D:
                    move.linear.x = 0
                else:
                    move.linear.x = self.v

            self.r.sleep()


if __name__ == '__main__':
    try:
        pi = math.pi
        pidc = PIDController()
        pidc.follow_wall()
    except rospy.ROSInterruptException:
        rospy.loginfo("Navigation terminated.")
```

طبق کد بالا میتوان گفت منطق کد برای همه بخش‌ها یکسان است و فقط برای بخش آخر که نقطه مقصد داریم، کمی منطق اضافه‌تر داریم تا مسیر حرکت به سمت نقطه پایانی سوق داده شود. نتایج حاصل به صورت زیر است: