دانشگاه صنعتی امیرکبیر
(پلی‌تکنیک تهران)

درس رباتیک
گزارش تمرین دوم

نگارش
محمدرضا حیدری : **9926053**

استاد درس
دکتر جوانمردی

اردیبهشت ماه **1403**

فهرست

تعریف مسئله

در آغاز کار، ربات باید درخواستی به سرویس GetNextDestination بفرستد. پس از دریافت مقصد بعدی به اندازه ی مناسب دوران کرده تا هدف دقیقا روبروی ربات قرار گیرد، سپس مستقیم به سمت هدف حرکت کند. ربات باید همواره رو به جلو حرکت کند تا فاصله اش از مانع کمتر از 3 متر شود. سپس ایستاده و دوباره درخواستی به GetNextDestination بفرستد و این فرایند را تا رسیدن به مقصد نهایی ادامه دهد. برای این کار نود کنترل می بایست از روی تاپیک ClosestObstacle تولید شده در نود سنسور subscribe نماید و از مقدار متغیر distance به عنوان فاصله تا نزدیک ترین مانع استفاده نماید.

Mission Node

این نود می بایست پاسخگوی سرویس GetNextDestination باشد، به گونه ای که برای هر درخواست، مقصد بعدی را از یک فایل text بخواند.

توابع مورد استفاده

`handle_get_next_destination` : این تابع پاسخگوی اصلی سرویس است.

`read_destination_from_line` : این تابع هر خط از فایل متنی را می خواند.

`parse_coordinates` : این تابع `x , y` را از خط جدا شده بر میگرداند.

```
#!/usr/bin/python3

import rospy
from homework.srv import GetNextDestination,GetNextDestinationResponse
import re
import os


def parse_coordinates(line):
    match = re.match(r'x:\s*([-+]?\d*\.\d+|\d+),\s*y:\s*([-+]?\d*\.\d+|\d+)',
line)
    if match:
        x = float(match.group(1))
        y = float(match.group(2))
        return x, y
    else:
        rospy.logerr("Invalid line format: %s", line)
        return None, None


line_number = 1
def read_destination_from_line():
    script_dir = os.path.dirname(os.path.abspath(__file__))
```

```
    file_path = os.path.join(script_dir, "obstacles.txt")
    try:
        with open(file_path, "r") as file:
            for _ in range(line_number - 1):
                file.readline()
            line = file.readline().strip()
            next_x, next_y = parse_coordinates(line)
            return next_x, next_y
    except IOError as e:
        rospy.logerr("Failed to read file: %s", str(e))
        return None, None




def handle_get_next_destination(req):
    global line_number
    x, y = read_destination_from_line()
    line_number = line_number+1
    if line_number == 5 :
        line_number = 1
    return GetNextDestinationResponse(next_x = x,next_y = y)

def mission_node():
    rospy.init_node('mission_node')
    service = rospy.Service('get_next_destination', GetNextDestination,
handle_get_next_destination)
    rospy.spin()

if __name__ == "__main__":
    mission_node()
```

Sensor_Node

این نود باید فاصله ربات تا موانع را با استفاده از سنسور LiDAR و تاپیک LaserScanبخواند و همواره مشخصات نزدیک ترین مانع را شامل فاصله بر روی تاپیک ClosestObstacleبفرستد. این تاپیک نیز به یک message customبا متغیر distanceنیاز دارد.

```
#!/usr/bin/python3
import rospy
from sensor_msgs.msg import LaserScan
```

```python
from homework.msg import proximity

class SensorNode:
    def __init__(self):
        rospy.init_node('sensor_node')
        self.pub_custom = rospy.Publisher('/custom_scan', proximity,
queue_size=10)
        rospy.Subscriber("/scan", LaserScan, self.laser_callback)
        rospy.on_shutdown(self.shutdown)
        rospy.spin()

    def laser_callback(self, msg):
        min_front_distance = (msg.ranges[0:30])
        min_front_distance2 = (msg.ranges[330:359])
        min_front_distance = min(min_front_distance)
        min_front_distance2 = min(min_front_distance2)
        min_front = min(min_front_distance,min_front_distance2)
        custom_msg = proximity()
        custom_msg.distance = min_front
        self.pub_custom.publish(custom_msg)

    def shutdown(self):
        rospy.loginfo("Shutting down sensor_node.")

if __name__ == "__main__":
    SensorNode()
```

Controller_Node

این نود وظیفه ی کنترل ربات را بر عهده دارد. فرض کنید که ربات ما دو stateدارد :

- حرکت به سمت جلو با سرعت خطی ثابت
- دوران در حالت ایستاده با سرعت زاویه ای دلخواه

*** یک ترم آلفا به صورت تجربی برای اصلاح جهت گیری استفاده شده است.

توابع مورد استفاده

get_heading_and_pose : مکان و جهت گیری ربات را در هر لحظه بر میگرداند.

get_next_destination : این تابع درخواستی به سرویس ارسال می کند.

calculate_rotation : میزان دوران مورد نیاز برای رسیدن به مقصد را با توجه مکان فعلی و مکان مقصد را بر میگرداند.

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

import rospy
import tf
import math
from sensor_msgs.msg import LaserScan
from nav_msgs.msg import Odometry
from geometry_msgs.msg import Twist
from homework.srv import GetNextDestination
from homework.msg import proximity

from math import radians

class Controller:

    def __init__(self) -> None:

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

        rospy.Subscriber('/custom_scan', proximity, self.callback_custom)
        self.laser_subscriber = rospy.Subscriber("/scan" , LaserScan ,
callback=self.laser_callback)
        self.cmd_publisher = rospy.Publisher('/cmd_vel' , Twist , queue_size=10)
        self.linear_speed = rospy.get_param("/controller_node/linear_speed") #
m/s
        self.angular_speed = -.8
        self.goal_angle = radians(0) # rad
        self.stop_distance = 2 # m
        self.distance = 0
        self.GO, self.ROTATE = 0, 1
        self.state = self.ROTATE


    def laser_callback(self, msg: LaserScan):
        if msg.ranges[0] <= self.stop_distance:
            self.state = self.ROTATE

    def get_heading_and_pose(self):

        msg = rospy.wait_for_message("/odom" , Odometry)
        orientation = msg.pose.pose.orientation
        position_y = msg.pose.pose.position.y
        position_x = msg.pose.pose.position.x
```

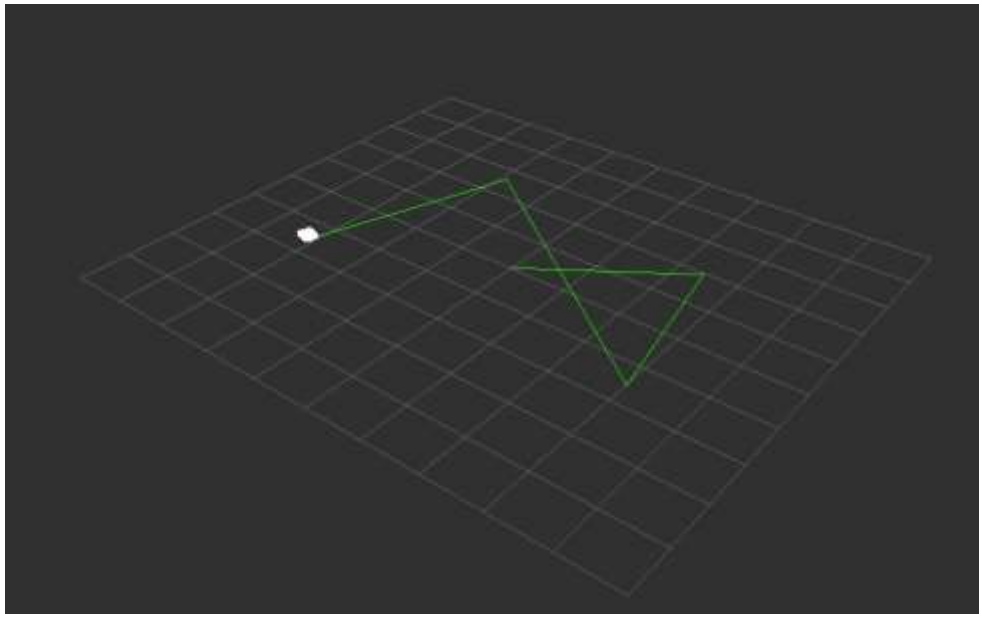
```
        roll, pitch, yaw = tf.transformations.euler_from_quaternion((
            orientation.x ,orientation.y ,orientation.z ,orientation.w
        ))
        return yaw ,position_x,position_y

    def get_next_destination(self):
     rospy.wait_for_service('get_next_destination')
     try:
        output = rospy.ServiceProxy('get_next_destination', GetNextDestination)
        response = output()
        return response.next_x, response.next_y
        pass
     except rospy.ServiceException as e:
        rospy.logerr("Service call failed: %s", e)
        return None, None

    def callback_custom(self,msg):
     self.distance = msg.distance

    def calculate_rotation(self,x, y, theta, next_x, next_y):
        delta_x = next_x - x
        delta_y = next_y - y
        target_angle = math.atan2(delta_y, delta_x)

        angle_diff = target_angle - theta


        rotate_degree = math.degrees(angle_diff)

        if rotate_degree > 180:
            rotate_degree -= 360
        elif rotate_degree < -180:
            rotate_degree += 360

        return rotate_degree


    def run(self):
        self.reamining = 0
        while not rospy.is_shutdown():
         next_x , next_y = self.get_next_destination();
         rospy.loginfo('next dest : x: %f , y :%f' , next_x,next_y)
         while (1):
            if self.linear_speed == .8 :
                if next_x == -4.6 and next_y == -4.3:
```

```
                    alpha  = 4
                elif next_x == -4.3 and next_y == 4.5:
                    alpha = -3
                elif next_x == 4.6 and next_y == -4.5 :
                    alpha = -12
                else :
                    alpha = -10
            elif self.linear_speed == 0.4 :
                if next_x == -4.6 and next_y == -4.3:
                    alpha  = 2
                elif next_x == -4.3 and next_y == 4.5:
                    alpha = -.5
                elif next_x == 4.6 and next_y == -4.5 :
                    alpha = -14
                else :
                    alpha = -2
            elif self.linear_speed == 0.2 :
                if next_x == -4.6 and next_y == -4.3:
                    alpha  = 1
                elif next_x == -4.3 and next_y == 4.5:
                    alpha = -.25
                elif next_x == 4.6 and next_y == -4.5 :
                    alpha = .5
                else :
                    alpha = 0
            prev_angle,position_x,position_y = self.get_heading_and_pose()
            rate = rospy.Rate(1)
            rate.sleep()
            if self.state == self.GO:
                twist = Twist()
                twist.linear.x = self.linear_speed
                self.cmd_publisher.publish(twist)
                rospy.loginfo('self.distance : %f',self.distance)
                rospy.loginfo('self.distancestop : %f',self.stop_distance)
                rospy.loginfo('out if')
                if (self.distance <= 3):
                  rospy.loginfo('in if')
                  prev_angle,position_x,position_y = self.get_heading_and_pose()
                  rospy.loginfo('now we are : x: %f , y :%f' ,
position_x,position_y)
                  error = (position_x-next_x)**2 + (position_y-next_y)**2
                  error = error**(0.5)
                  rospy.loginfo('error : %f',error)
                  next_x , next_y = self.get_next_destination();
                  rospy.loginfo('next dest : x: %f , y :%f' , next_x,next_y)
```

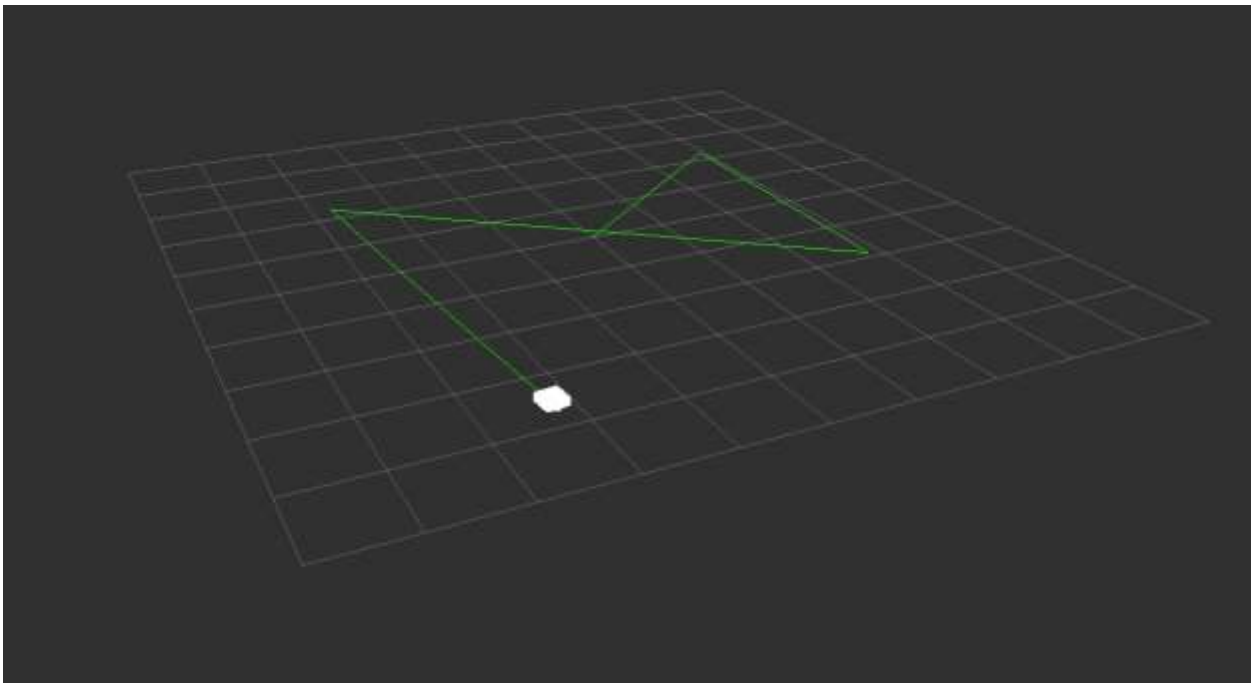
```
                    self.state = self.ROTATE
                continue

            self.cmd_publisher.publish(Twist())
            remaining =
self.calculate_rotation(position_x,position_y,prev_angle,next_x,next_y)-alpha
            if remaining<0:
                twist = Twist()
                twist.angular.z = self.angular_speed
                self.cmd_publisher.publish(twist)
            else:
               twist = Twist()
               twist.angular.z = self.angular_speed*(-1)
               self.cmd_publisher.publish(twist)

            while abs(remaining) >= 1:
                prev_angle,position_x,position_y = self.get_heading_and_pose()
                remaining =
self.calculate_rotation(position_x,position_y,prev_angle,next_x,next_y)-alpha

            self.cmd_publisher.publish(Twist())

            rospy.sleep(1)

            self.state = self.GO


if __name__ == "__main__":
    controller = Controller()
    controller.run()
```

نما های rviz به ازای سرعت های خطی مختلف

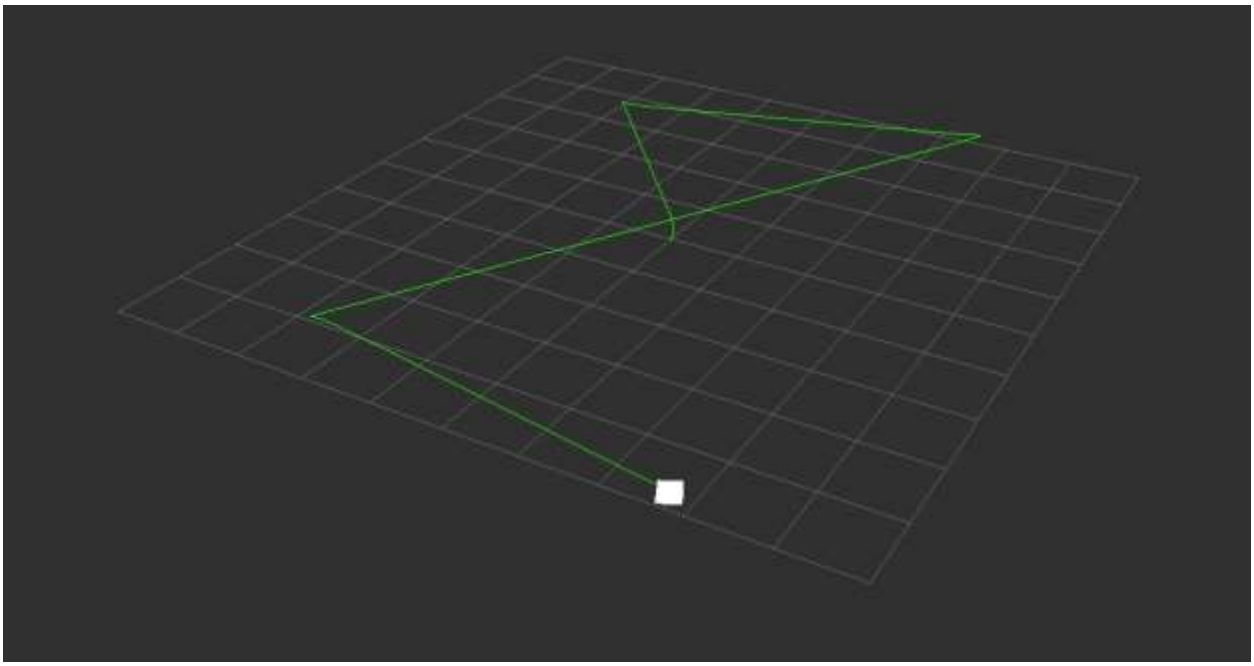
نمای rviz با سرعت خطی 0.2



نمای rviz با سرعت خطی 0.4



نمای rviz با سرعت خطی 0.8



خطای انحراف از مقصد به ازای هر سرعت خطی

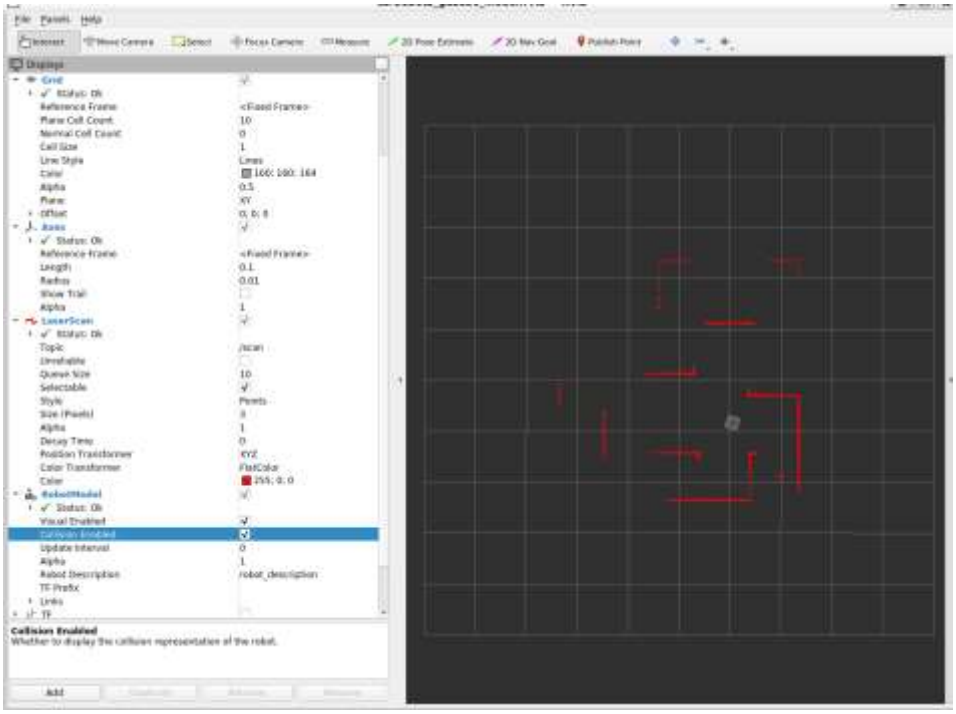
به ازای فاصله توقف 3 از مانع

مقصد / سرعت	اول	دوم	سوم	چهارم	میانگین
0.2	3.46	3.50	3.54	3.43	3.48
0.4	3.20	3.49	3.53	3.29	3.38
0.8	2.87	3.40	2.93	2.93	3.03

جدول 1) خطای فاصله از مقصد

گام دوم

بخش اول

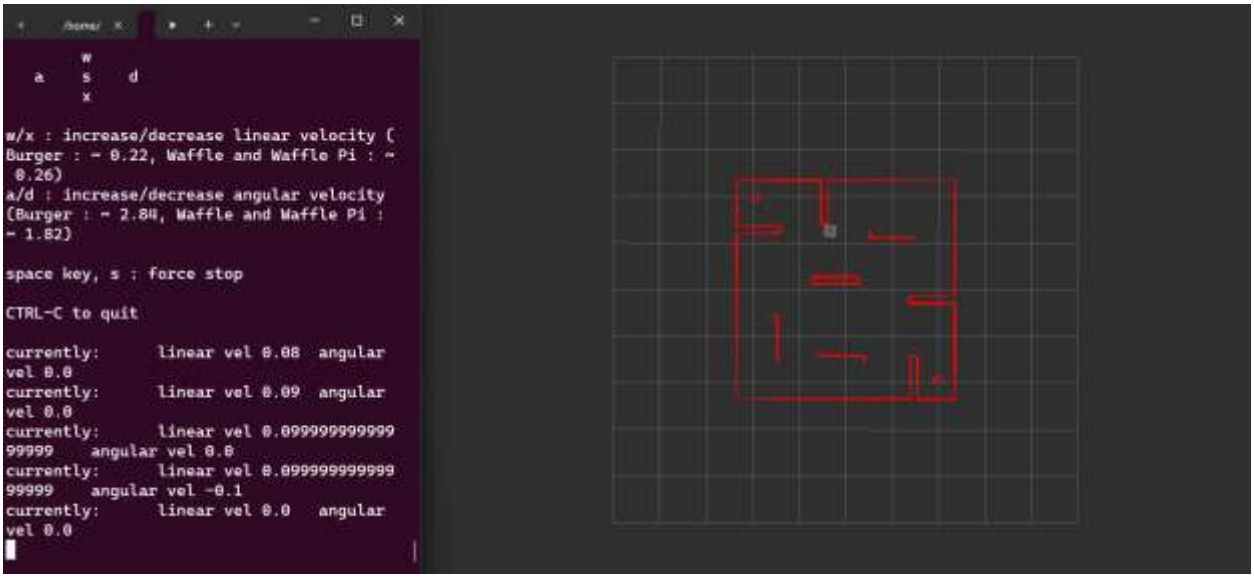


دلیل لرزش نقاط

مشکل در داده‌های ورودی : لرزش ممکن است ناشی از مشکل در داده‌هایی باشد که به rviz ارسال می‌شود.

اگر داده‌ها با تاخیر یا ناهماهنگی ارسال شوند، نقاط در نمای rviz لرزش خواهند داشت.

بخش دوم



چالش ها انجام تمرین

1 – هر دفعه ربات چرخش های متفاوتی انجام می دهد و ترم آلفا که برای اصلاح زاویه چرخش استفاده می شود باید به صورت دستی و تجربی تنظیم شود. برای رفع این مشکل نیازمند یک الگوریتم کنترلی هستیم.

2 – متاسفانه به ازای فاصله توقف از مانع 2 شرط کنترلر عمل نمی کند !!!! این واقعا مورد عجیب و به نظر باگ کامپایلر است.برای اجرا و گزارش از فاصله توقف 3 استفاده شده است.

```
if (self.distance <= 3):
  rospy.loginfo('in if')
```