



Department of  
Computer Engineering

به نام خدا



Amirkabir University of Technology  
(Tehran Polytechnic)

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

تمرین سری دوم

محمدسپهر توکلی کرمانی	نام و نام خانوادگی
۹۸۳۱۱۱۱	شماره دانشجویی
۱۴۰۱/۲/۱۸	تاریخ ارسال گزارش

## فهرست گزارش سوالات

۳.....	سوال ۱ – سوال اول
۴.....	سوال ۲ – سوال دوم
۵.....	سوال ۳ – سوال سوم
۶.....	سناریو ۱
۱۵.....	سناریو ۲
۱۸.....	سناریو ۳

## سؤال ١ - سؤال اول

$$\begin{bmatrix} \cos\theta & -\sin\theta & x \\ \sin\theta & \cos\theta & y \\ 0 & 0 & 1 \end{bmatrix} \xrightarrow{\alpha, \beta, \gamma} \begin{bmatrix} \cos\alpha & -\sin\alpha & x \\ \sin\alpha & \cos\alpha & y \\ 0 & 0 & 1 \end{bmatrix} \quad \textcircled{1}$$

$$\rightarrow \begin{bmatrix} \cos(a+b) & -\sin(a+b) & \cos a + \cos(b-a)\sin a \\ \sin(a+b) & \cos(a+b) & \sin a + \cos(b-a)\sin b \\ 0 & 0 & 1 \end{bmatrix}$$

## سؤال ۲ - سوال دوم

$$R_{ab} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \quad P_a = (1, r, r) \rightarrow P_b = ? \quad \text{--- (1)}$$

$$P_a = R_{ab} P_b \rightarrow P_b = P_a (R_{ab})^{-1}$$

$$R_{ab} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \xrightarrow{\text{Observe}} R_{ab}^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} 1 \\ r \\ r^2 \end{bmatrix} = \begin{bmatrix} 1+0+0 \\ 0+0+r \\ 0+1+r^2 \end{bmatrix} = \begin{bmatrix} 1 \\ r \\ -r^2 \end{bmatrix}$$

Duno

$$P_b = (1, -r^2, r)$$

NOTE BOOK

### سؤال ٣ - سؤال سوم

**Forward**

$$z = a \sin(\theta_r) + d_r \cos(\theta_r) \rightarrow z = a \sin(\theta_r) + d_r \cos(\theta_r)$$

$$\left. \begin{array}{l} x = d_r \cos(\theta_r) \sin(\theta_r) \\ y = d_r \sin(\theta_r) \cos(\theta_r) \end{array} \right\} \begin{array}{l} x \rightarrow d_r \cos(\theta_r) \sin(\theta_r) \\ y \rightarrow d_r \sin(\theta_r) \cos(\theta_r) \end{array}$$

**Inverse**

$$\tan \theta_r = \frac{d_r}{a}$$

$$\theta_r = \arctan \left( \frac{d_r}{a} \right)$$

$$\theta_r = \arctan \left( \frac{a'^r + b'^r}{a} \right)$$

$$d_r = \sqrt{a'^r + b'^r}$$

$$\tan \theta_1 = \frac{b'}{a'}$$

$$\theta_1 = \arctan \left( \frac{b'}{a'} \right)$$

NOTE BOOK

## بخش عملی

سناریو ۱ :

در بخش پس از نصب پکیج های مورد نظر و ایجاد فایل controller و launch ، نود ایجاد شده را با دستور rosrun اجرا می کنیم.  
فایل : launch

```
src > ros_move > launch > controller.launch
1   <launch>
2
3     <node pkg="ros_move" type="controller.py" name="controller" output="screen">
4       <param name="linear_speed" value="$(arg linear_speed)" />
5       <!-- <param name="linear_speed" value="0.8" /> -->
6       <param name="angular_speed" value="0.5"/>
7       <param name="goal_angle" value="90"/>
8       <param name="epsilon" value="0.01"/>
9
10    </node>
11
12
13    <node pkg="ros_move" type="monitor.py" name="monitor"></node>
14
15    <include file="$(find turtlebot3_gazebo)/launch/turtlebot3_empty_world.launch">
16      <arg name="x_pos" value="0"/>
17      <arg name="y_pos" value="0"/>
18      <arg name="z_pos" value="0"/>
19    </include>
20
21    <include file="$(find turtlebot_rviz_launchers)/launch/view_robot.launch"/>
22
23  </launch>
24
```

و حالا توابع پیاده شده در فایل کنترولر را توضیح می دهیم :

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

        # getting specified parameters
        self.linear_speed = rospy.get_param("/controller/linear_speed") # m/s
        self.angular_speed = rospy.get_param("/controller/angular_speed") # rad/s
        self.goal_angle = radians(rospy.get_param("/controller/goal_angle")) # rad
        self.epsilon = rospy.get_param("/controller/epsilon")

        # define rectangle
        self.length = 6
        self.width = 4

        self.spendTime = 0

        self.pose_x = []
        self.pose_y = []
        self.pose_x.append(0)
        self.pose_y.append(0)

        # defining the states of our robot
        self.GO, self.DIRECTION = 1, 1

        # do some cleanup on shutdown
        rospy.on_shutdown(self.clean_shutdown)
```

در این فایل مقادیر کنترلی و طول و عرض مستطیل را مشخص می کنیم ، نقاط طی شده توسط ربات را در ۲ آرایه pose\_x و pose\_y قرار می دهیم. هم چنین ۲ استیت برای ربات در نظر می گیریم.

```
def get_heading(self):
    # waiting for the most recent message from topic /odom
    msg = rospy.wait_for_message("/odom", Odometry)
    orientation = msg.pose.pose.orientation
    # convert quaternion to odom
    roll, pitch, yaw = tf.transformations.euler_from_quaternion((orientation.x, orientation.y, orientation.z, orientation.w))
    return yaw
```

تابعی برای مشخص کردن heading ربات.

```
def clean_shutdown(self):
    rospy.loginfo("System is shutting down. Stopping robot...")
    twist = Twist()
    twist.linear.x = 0
    twist.angular.z = 0
    self.cmd_publisher.publish(twist)
```

تابع clean\_shutdown برای متوقف کردن ربات.

```
def rotate(self):
    rospy.loginfo("Robot is rotating to left ...")
    remaining = self.goal_angle
    prev_angle = self.get_heading()
    twist = Twist()
    twist.angular.z = self.angular_speed
    self.cmd_publisher.publish(twist)
    # rotation loop
    while remaining >= 0:
        current_angle = self.get_heading()
        delta = abs(prev_angle - current_angle)
        remaining -= delta
        prev_angle = current_angle
    rospy.loginfo("Robot rotating completed.")
```

تابع rotate برای چرخش ربات تا رسیدن به مقدار .goal\_angle.

```

def moveForward(self):
    rospy.loginfo("Robot is moving forward...")
    traveled = 0
    steps = 0
    if self.DIRECTION == 1:
        steps = self.length
        self.DIRECTION = 0
    else:
        steps = self.width
        self.DIRECTION = 1
    msg = rospy.wait_for_message("/odom", Odometry)
    start_x = msg.pose.pose.position.x
    start_y = msg.pose.pose.position.y
    while steps > traveled:
        twist = Twist()
        twist.linear.x = self.linear_speed
        msg = rospy.wait_for_message("/odom", Odometry)
        traveled = abs(msg.pose.pose.position.y - start_y) + abs(msg.pose.pose.position.x - start_x)

        self.pose_x.append(msg.pose.pose.position.x)
        self.pose_y.append(msg.pose.pose.position.y)

        self.cmd_publisher.publish(twist)
    rospy.loginfo("Robot moving completed. Steps: %d", steps)

```

تابع moveForward برای حرکت ربات به سمت جلو که ۲ حالت دارد ، یا به اندازه طول ربات به سمت جلو می رود یا به اندازه عرض آن که این توسط DIRECTION مشخص می شود. شیوه اجرا شدن نیز به این شکل است که اخیرین پیام تایپیک odom خوانده شده و با مسافت طی شده چک می شود تا به مقدار مورد نظر برسد. مقدار سرعت خطی در جهت x سُت شده و آن را twist را publish می کنیم و ربات شروع به حرکت می کند.

```

def run(self):
    while not rospy.is_shutdown():

        self.moveForward()
        self.cmd_publisher.publish(Twist())
        rospy.sleep(1)

        # 1 complete rectangle round and then shutdown
        if self.spendTime == 3 :
            rospy.signal_shutdown("We are done here!")
            break
        self.spendTime += 1

        self.rotate()
        self.cmd_publisher.publish(Twist())
        rospy.sleep(1)

        # show plots
        self.ploot()

```

تابع run که تا زمان shutdown نشدن ربات ، اجرا می شود و ربات حرکت مستقیم و چرخش انجام می دهد.

```

def ploot(self):
    plt.xlabel("X-Axis")
    plt.ylabel("Y-Axis")
    plt.title("Robot Error")

    # draw rectangle
    num = len(self.pose_x)
    X1 = np.linspace(0, self.length , num)
    Y1 = np.array([self.width]*num)

    Y2 = np.linspace(self.width, 0 , num)
    X2 = np.array([self.length]*num)

    X3 = np.linspace(self.length, 0 , num)
    Y3 = np.array([0]*num)

    Y4 = np.linspace(0, self.width , num)
    X4 = np.array([0]*num)

    plt.plot(np.concatenate([X1,X2, X3 , X4]), np.concatenate([Y1,Y2,Y3,Y4]), color='r', label='Default Path')
    plt.plot(self.pose_x,self.pose_y, color='b', label='Robot Path')

    zip_object1 = np.concatenate([X1,X2, X3 , X4])
    zip_object2 = np.concatenate([Y1,Y2,Y3,Y4])

    points = []
    for X2, Y2 in zip(self.pose_x , self.pose_y):
        minn = 1000
        for X1, Y1 in zip(zip_object1,zip_object2):
            distance = dist([X2,Y2],[X1,Y1])
            if(distance < minn ):
                minn = distance
        points.append(minn)

    plt.plot(points, color='g' , label='diff')

    plt.legend()
    plt.savefig("plots.png")
    plt.show()

```

تابع plot نیز وظیفه رسم کردن نمودار های خواسته شده را دارد ( خروجی آن را در قسمت نتایج میبینیم).

```

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

```

در نهایت تابع main را خواهیم داشت.

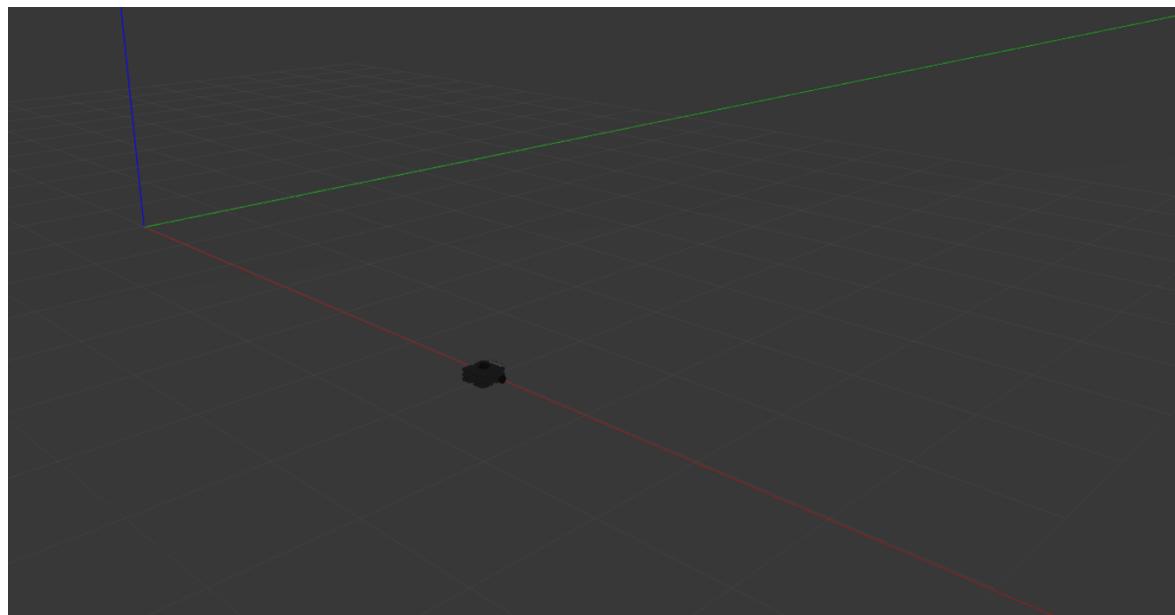
فایل monitor.py که برای نشان دادن مسیر طی شده توسط ربات استفاده می شود :

```
src > ros_move > src > monitor.py > ...
1  #!/usr/bin/python3
2
3  import rospy
4  from nav_msgs.msg import Odometry, Path
5  from geometry_msgs.msg import PoseStamped
6
7  class PathMonitor:
8
9      def __init__(self) -> None:
10         rospy.init_node("monitor" , anonymous=False)
11
12         self.path = Path()
13         self.odom_subscriber = rospy.Subscriber("/odom" , Odometry , callback=self.odom_callback)
14         self.path_publisher = rospy.Publisher("/path" , Path , queue_size=10)
15
16     def odom_callback(self, msg : Odometry):
17         self.path.header = msg.header
18         pose = PoseStamped()
19         pose.header = msg.header
20         pose.pose = msg.pose.pose
21         self.path.poses.append(pose)
22         self.path_publisher.publish(self.path)
23
24
25 if __name__ == "__main__":
26     path_monitor = PathMonitor()
27     rospy.spin()
28
```

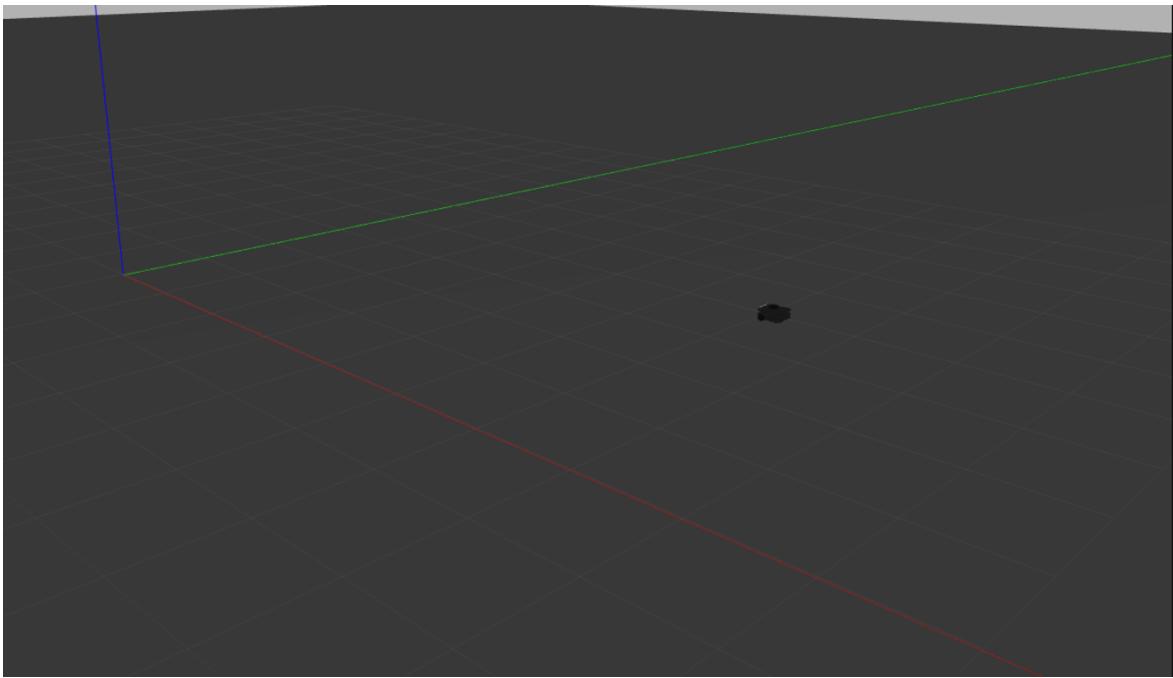
حالا برنامه را اجرا کرده و نتایج را می بینیم :

Gazebo :

حرکت روی طول مستطیل :

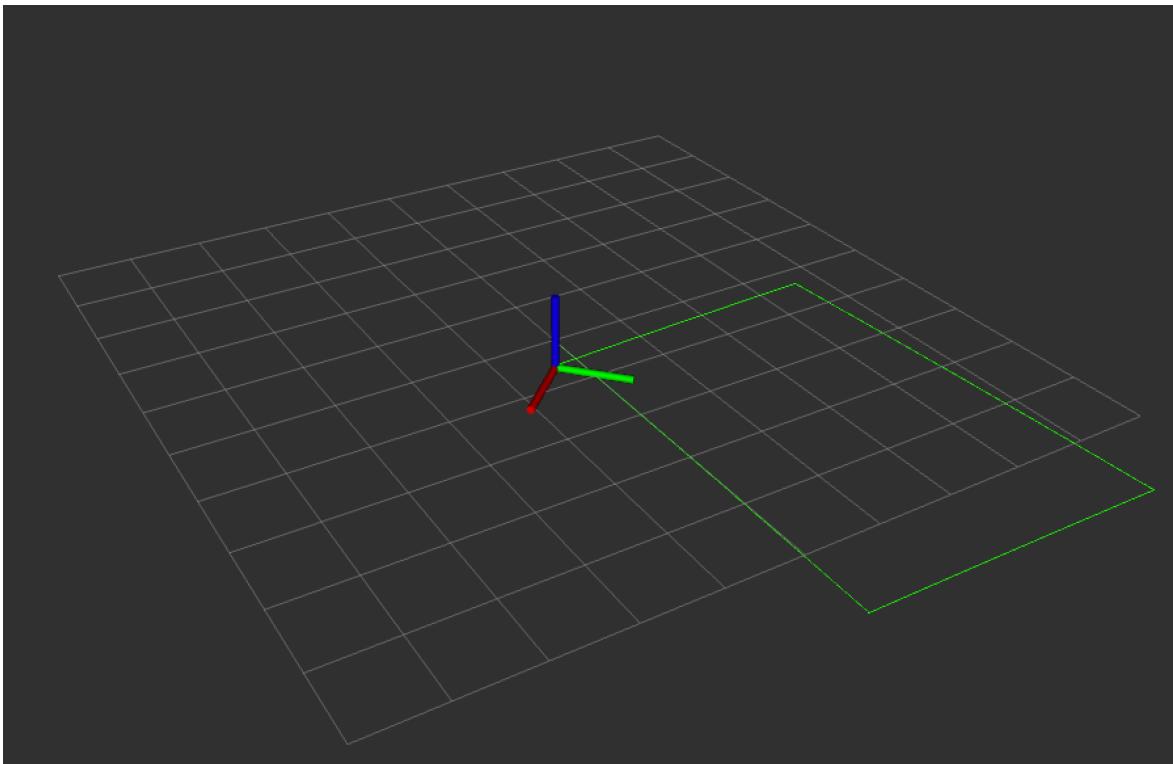


حرکت روی عرض مستطیل :

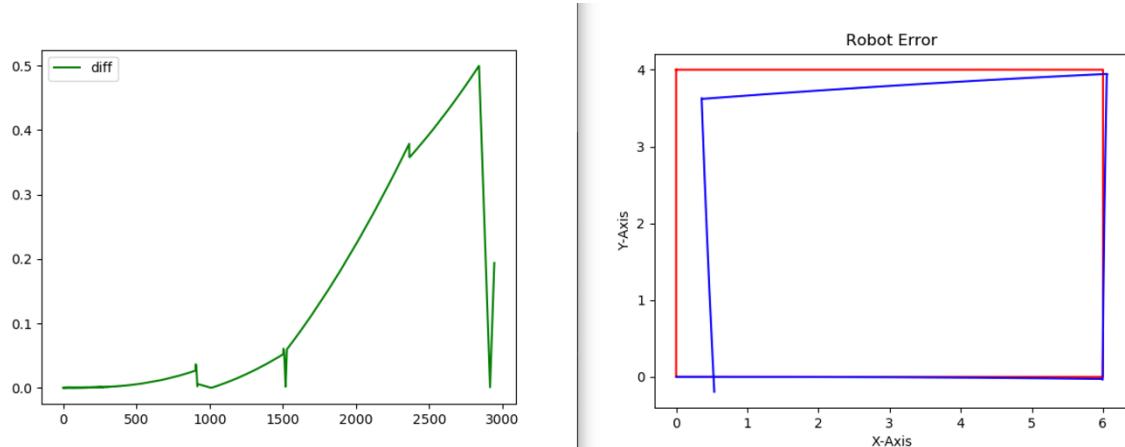


حالا با مقدار  $0.2^{\circ}$  برای سرعت خطی اجرا می کنیم :

مسیر طی شده در rviz به شکل زیر است :

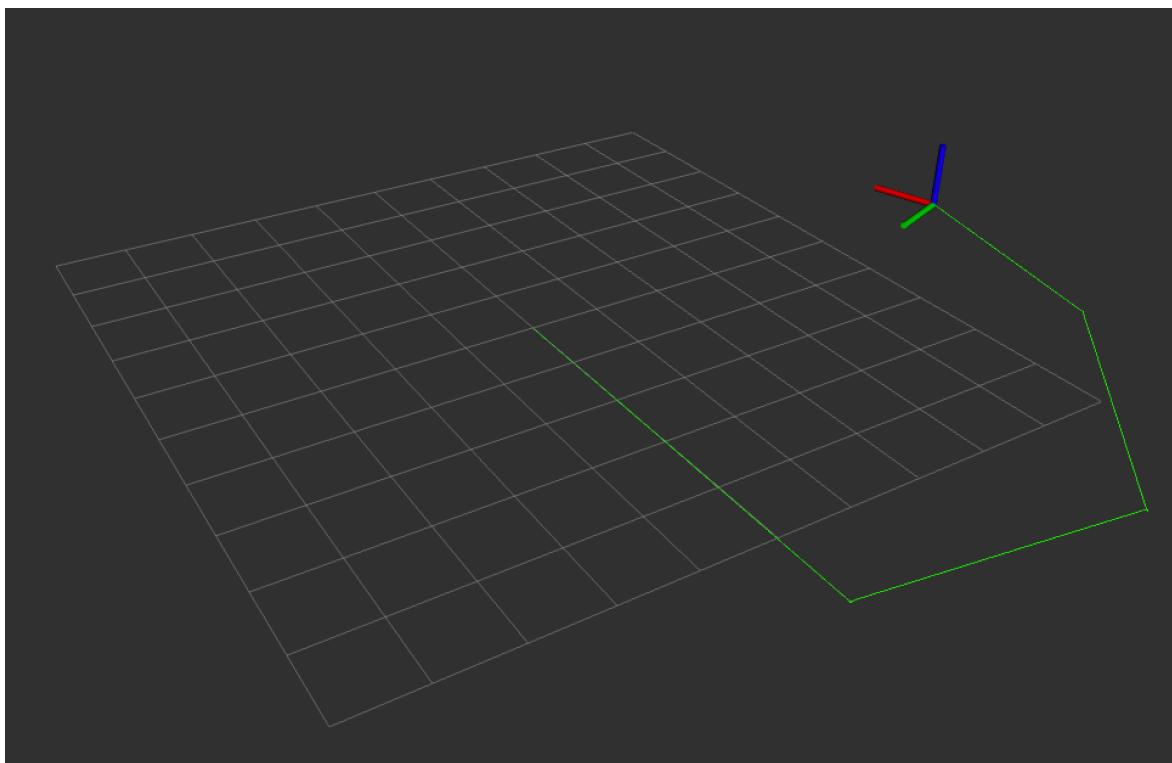


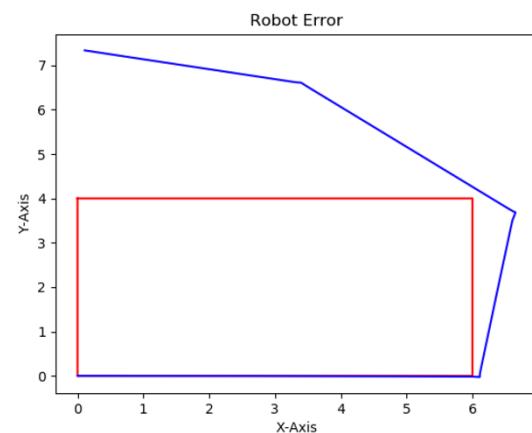
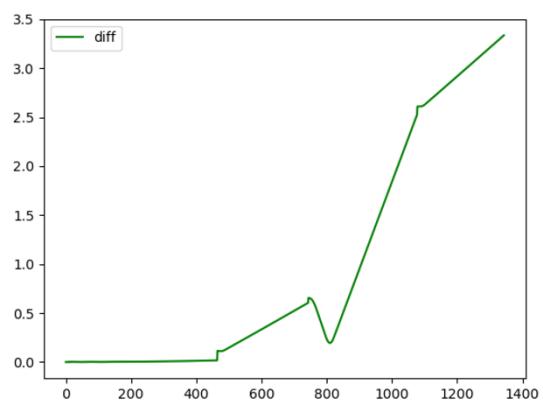
نمودار های خواسته شده نیز به شکل زیر هستند :



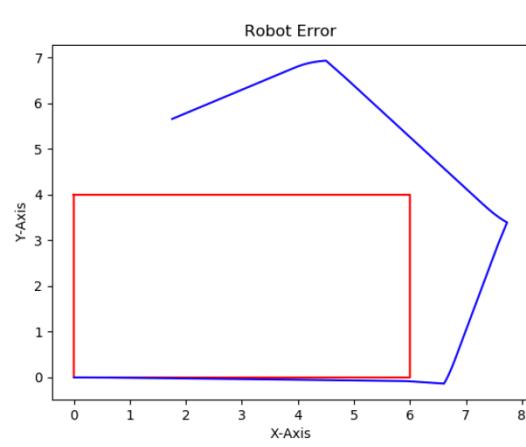
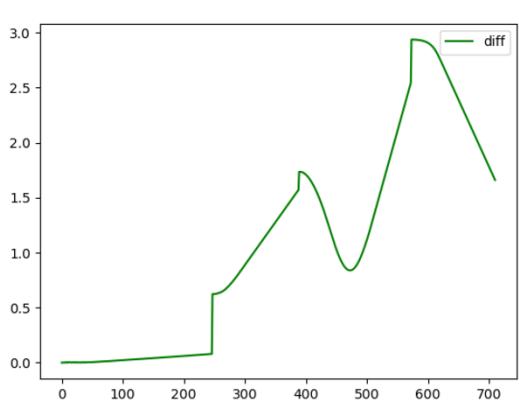
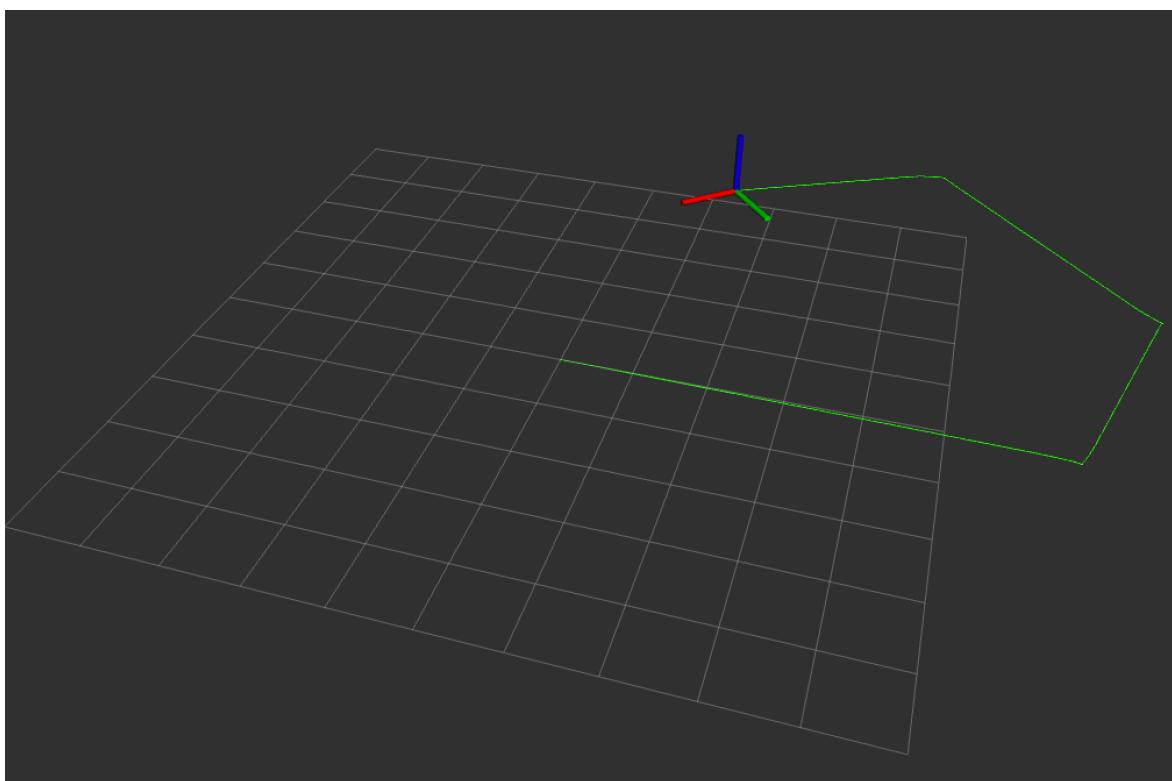
نمودار سمت چپی خط سبز خطای ربات را به ازای سرعت خطی  $0.2\text{m/s}$  نشان می دهد. در نمودار سمت راست شکل آبی مسیر طی شده و شکل قرمز مسیر اصلی ربات که مستطیل  $6\text{m} \times 4\text{m}$  است را نشان می دهد.

اکنون تمامی مراحل را به ازای سرعت خطی  $0.4\text{m/s}$  انجام می دهیم :





اکنون برای سرعت خطی ۰.۸ انجام می دهیم :



واضح است که وقتی سرعت خطی را افزایش دادیم ، خطای زیاد شد ، دلیل آن هم تاخیر به وجود آمده در تاپیک odom است که باعث می شود اگر ربات قرار است در نقطه ۶ بایستد ، کمی جلوتر متوقف می شود.

## سناریو ۲ :

در این بخش به پیاده سازی کنترولر pid می پردازیم و ۲ کنترولر داریم یکی برای فاصله و یکی برای زاویه.

برای این گام یک مستطیل به طول ۶ و عرض ۴ با استفاده از کتابخانه numpy درست می کنیم و ربات قرار است روی آن حرکت کند. علاوه بر توابع قسمت قبلی دوتابع جدید زیر را اضافه کردیم:

```
def makeRectangle(self):
    rectangle = []

    X1 = np.linspace(0, 4 , 5)
    for x in X1:
        rectangle.append([x,0.0])

    X3 = np.linspace(4, 0 , 5)
    for x in X3:
        rectangle.append([x,6.0])

    Y2 = np.linspace(0, 4 , 5)
    for y in Y2:
        rectangle.append([0.0,y])

    Y4 = np.linspace(4, 0 , 5)
    for y in Y4:
        rectangle.append([y,6.0])
    return rectangle
```

```
def getCurrentPosition(self):
    # waiting for the most recent message from topic /odom
    msg = rospy.wait_for_message("/odom" , Odometry)
    return msg.pose.pose.position
```

تابع run را نیز به این شکل تغییر دادیم تا مراحل پیاده سازی کنترولر pid در آن اجرا شود :  
( در پیاده سازی فوق از فیلم اموزش ros در یوتیوب مربوط به پیاده سازی کنترولر pid استفاده کرده ام )

با افزایش مقادیر pid زاویه تعداد و سرعت دور زدن زیاد می شود و باعث افزایش انحراف می شود ، کاهش مقادیر مربوط به فاصله سرعت را کم و دقیق را بالاتر می برد .

```

while not rospy.is_shutdown():
    last_rotation = 0
    for goal in self.rectangle:
        self.cmd_publisher.publish(Twist())

    total_distance = 0

    current_pose = self.getCurrentPosition()

    distance = dist([goal[0],goal[1]],[current_pose.x,current_pose.y])

    while distance > 0.25:

        rotation = self.get_heading()
        current_pose = self.getCurrentPosition()
        path_angle = atan2(goal[1]-current_pose.y , goal[0] - current_pose.x)
        if path_angle < -pi/4 or path_angle > pi/4:
            if goal[1] < 0 and current_pose.y < goal[1]:
                path_angle = -2*pi + path_angle
            elif goal[1] >= 0 and cu (variable) path_angle: float
                path_angle = 2*pi + path_angle
        if last_rotation > pi-0.1 and rotation <= 0:
            rotation = 2*pi + rotation
        elif last_rotation < -pi+0.1 and rotation > 0:
            rotation = -2*pi + rotation

        distance = dist([goal[0],goal[1]],[current_pose.x,current_pose.y])

        control_signal_distance = self.kp_distance*distance + self.ki_distance*total_distance

        control_signal_angle = self.kp_angle*(path_angle - rotation)

        self.twist.angular.z = (control_signal_angle)

        self.twist.linear.x = min(control_signal_distance,0.1)

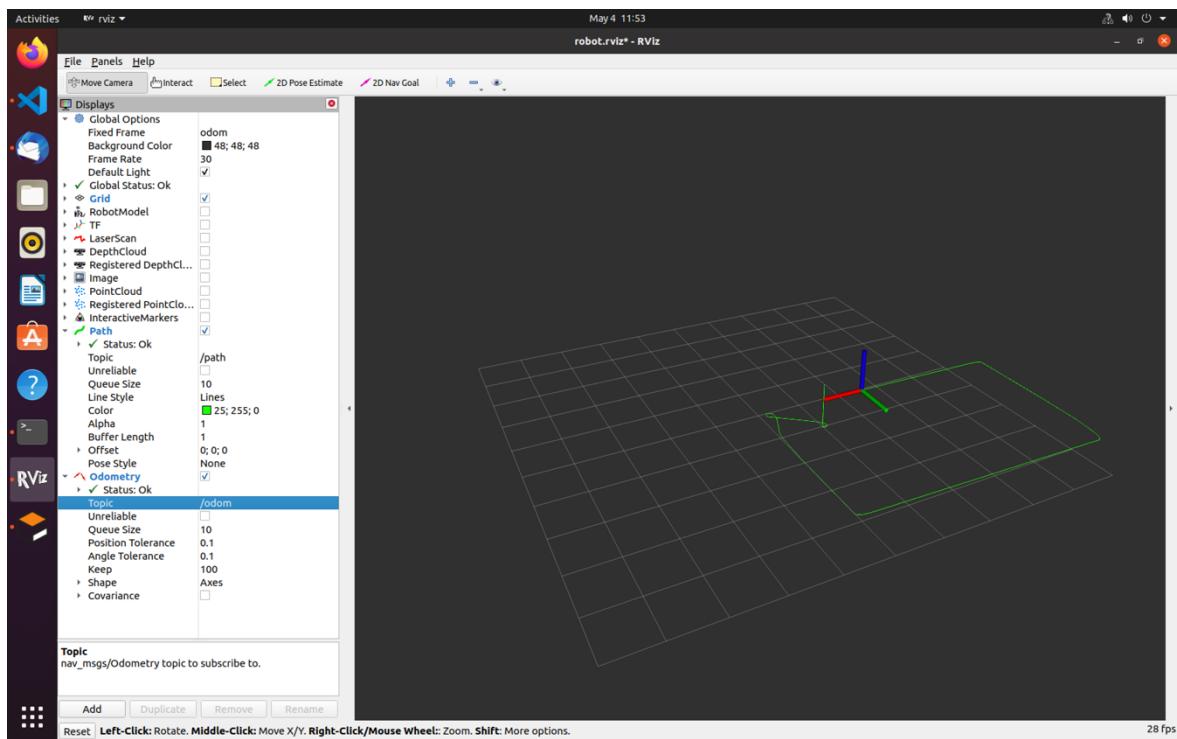
        if self.twist.angular.z > 0:
            self.twist.angular.z = min(self.twist.angular.z, 1.5)
        else:
            self.twist.angular.z = max(self.twist.angular.z, -1.5)

        last_rotation = rotation
        self.cmd_publisher.publish(self.twist)

        total_distance += distance

```

سپس نقطه شروع را ۱ و قرار داده و برنامه را اجرا می کنیم. مسیر طی شده در rviz به این شکل است:



ابتدا ربات در نقطه ۱ و ۱ به زمین می افتد و سپس به سمت نقطه ۰، ۰ مستطیل حرکت کرده و مسیر خود را روی آن ادامه می دهد.

### سناریو ۳ :

در این بخش ۴ تابع برای پیاده کردن شکل ها می سازیم و همه آن ها در آرایه shape در کلاس کنترولر قرار می دهیم.

```
def makeLogarithem(self):
    log = []
    a = 0.17
    k = math.tan(a)
    for i in range(150):
        t = i / 20 * math.pi
        dx = a * math.exp(k * t) * math.cos(t)
        dy = a * math.exp(k * t) * math.sin(t)
        log.append([dx,dy])
    self.shape.append(log)

def makeArchimedean(self):
    archi = []
    growth_factor = 0.1
    for i in range(400):
        t = i / 20 * math.pi
        dx = (1 + growth_factor * t) * math.cos(t)
        dy = (1 + growth_factor * t) * math.sin(t)
        archi.append([dx,dy])
    self.shape.append(archi)

def makeEightees(self):
    eight = []
    X2 = np.linspace(1, 1 + 2**((1/2)) , 10)
    Y2 = - (2**((1/2))) * (X2 - 1) + 3
    for i,x in enumerate(X2):
        eight.append([x,Y2[i]])

    Y3 = np.linspace(1, -1 , 10)
    X3 = np.array([1 + 2**((1/2)]*10)
    for i,x in enumerate(X3):
        eight.append([x,Y3[i]])

    X4 = np.linspace(1 + 2**((1/2)), 1, 10)
    Y4 = (2**((1/2))) * (X4 - 1 - 2**((1/2))) -1
    for i,x in enumerate(X4):
        eight.append([x,Y4[i]])

    X5 = np.linspace(1, -1 , 10)
    for i,x in enumerate(X5):
        eight.append([x,-3])

    X6 = np.linspace(-1, -1 - 2**((1/2)) , 10)
    Y6 = - (2**((1/2))) * (X6 + 1) - 3
    for i,x in enumerate(X6):
        eight.append([x,Y6[i]])

    Y7 = np.linspace(-1, 1 , 10)
    X7 = np.array([- 1 - 2**((1/2)]*10)
    for i,x in enumerate(X7):
        eight.append([x,Y7[i]])

    X8 = np.linspace(-1 - 2**((1/2)), -1, 10)
    Y8 = (2**((1/2))) * (X8 + 1 + 2**((1/2))) + 1
    for i,x in enumerate(X8):
        eight.append([x,Y8[i]])

    X1 = np.linspace(-1, 1 , 10)
    for x in X1:
        eight.append([x,3])
    self.shape.append(eight)
```

```
def makeHalfCircles(self):
    circles = []
    X1 = np.linspace(-6., -2 , 20)
    for x in X1:
        circles.append([x, 0.0])

    x_dim, y_dim = 2,2
    t = np.linspace(np.pi, 0, 30)
    for t1 in t:
        circles.append([x_dim * np.cos(t1), y_dim * np.sin(t1)])

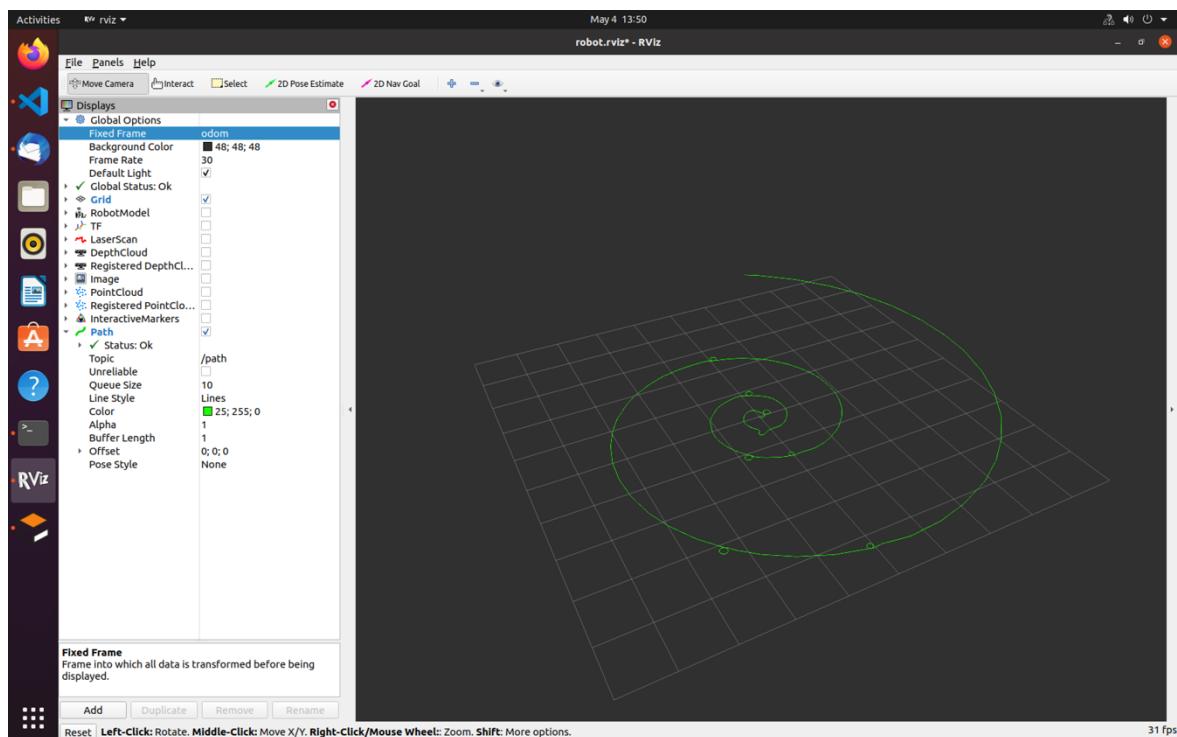
    X3 = np.linspace(2, 6 , 20)
    Y3 = np.zeros((50,))
    for x in X3:
        circles.append([x, 0.0])

    x_dim, y_dim = 6,6
    t = np.linspace(np.pi*2, np.pi, 40)
    for t2 in t:
        circles.append([x_dim * np.cos(t2), y_dim * np.sin(t2)])

    self.shape.append(circles)
```

با امتحان کردن مقادیر مختلف pid بهترین را برای هر شکل حساب می کنیم :

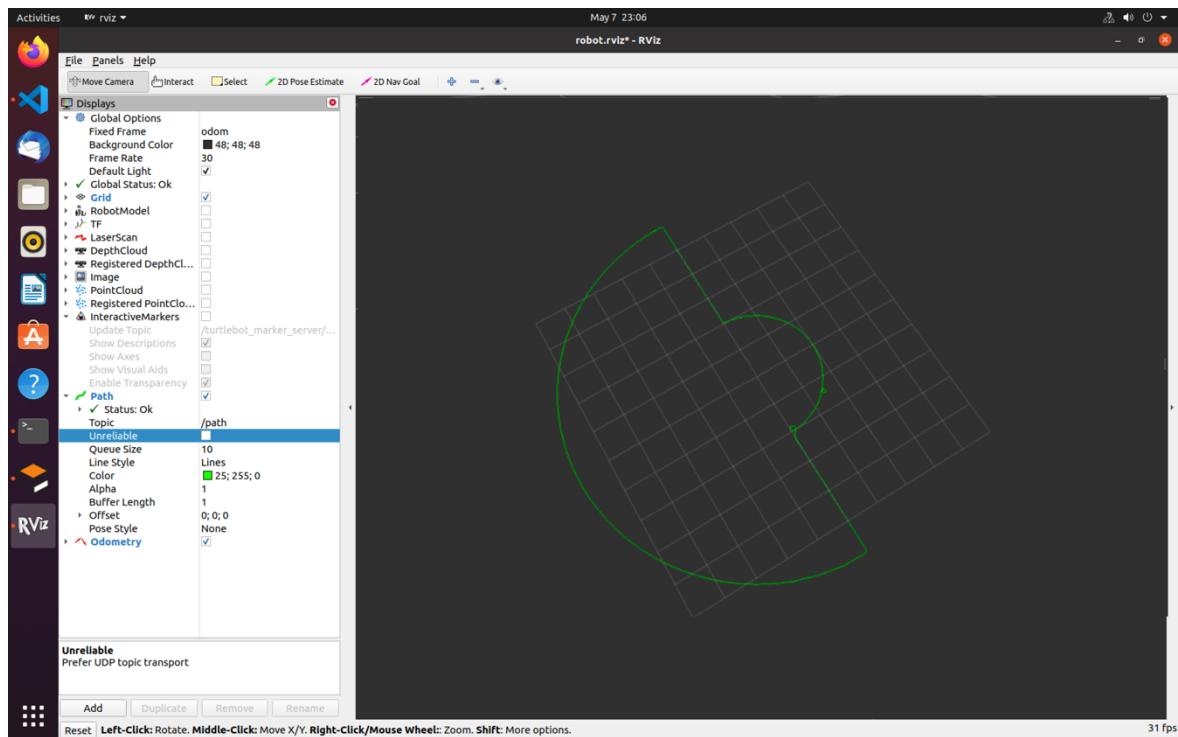
۱) شکل لوگاریتمی :



```
self.kp_distance = 15  
self.ki_distance = 20  
self.kd_distance = 1
```

```
self.kp_angle = 1  
self.ki_angle = 0.03  
self.kd_angle = 0.05
```

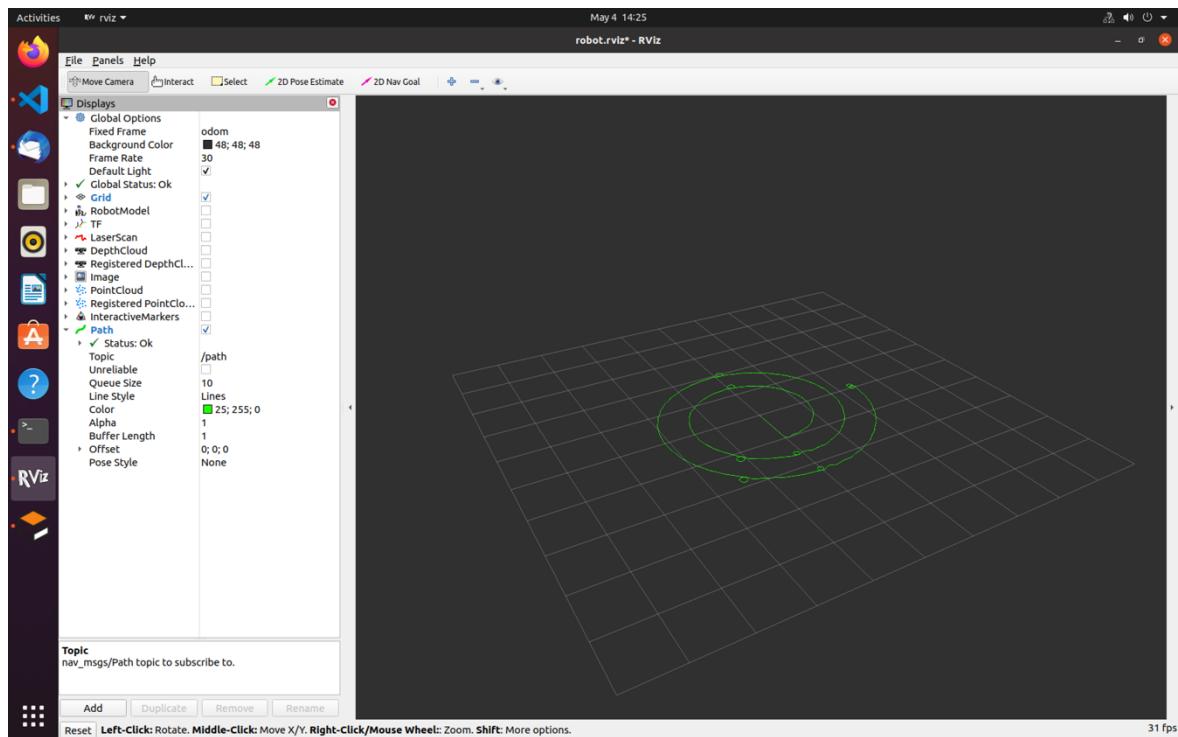
۲) شکل ترکیب دو نیم دایره :



```
self.kp_distance = 10
self.ki_distance = 20
self.kd_distance = 2

self.kp_angle = 1
self.ki_angle = 0.03
self.kd_angle = 0.05
```

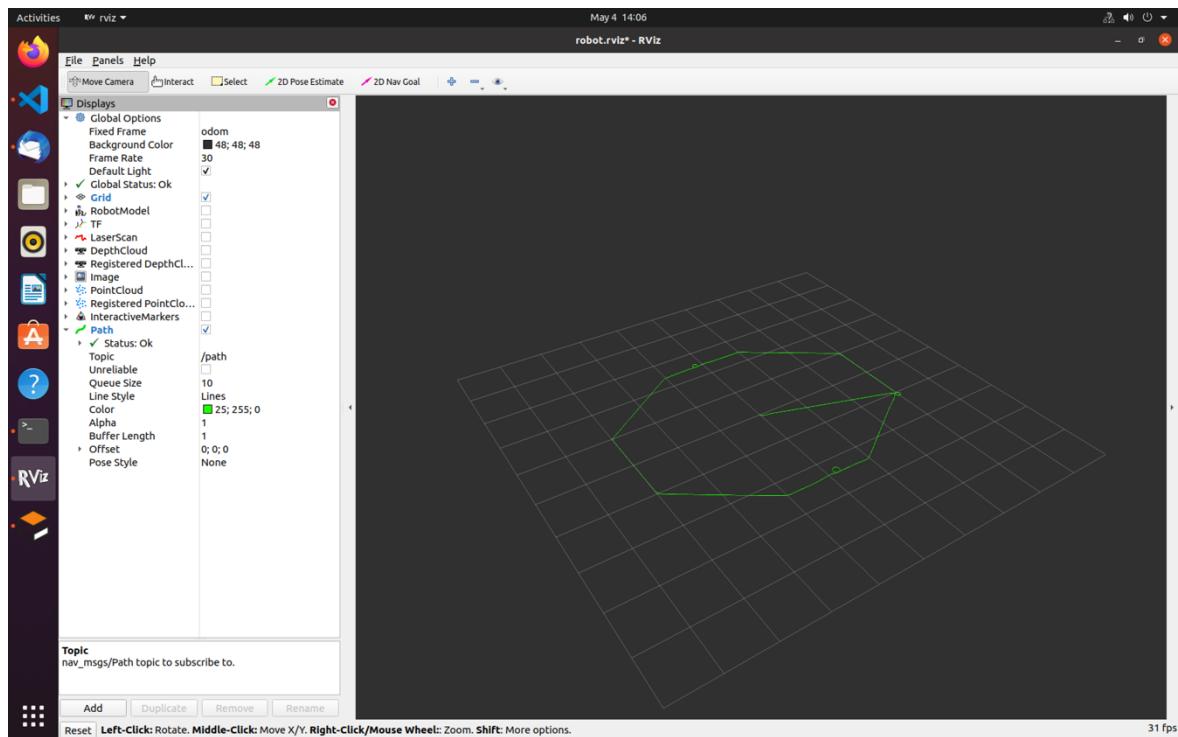
۳) شکل مارپیچ ارشمیدسی :



```
self.kp_distance = 20
self.ki_distance = 18
self.kd_distance = 1

self.kp_angle = 1.5
self.ki_angle = 0.03
self.kd_angle = 0.05
```

٤) شکل هشت ضلعی منتظم :



```
self.kp_distance = 8
self.ki_distance = 0.5
self.kd_distance = 1

self.kp_angle = 1
self.ki_angle = 0.03
self.kd_angle = 0.05
```

در تابع run نیز اول شکل مورد نظر را مشخص کرده و سپس محاسبات pid را مانند بخش قبلی انجام می دهیم :

```
def run(self):
    self.makeArchimedean()
    self.makeLogarithem()
    self.makeHalfCircles()
    self.makeEightees()

    while not rospy.is_shutdown():
        last_rotation = 0
        for i,goal in enumerate(self.shape[0]):
            self.cmd_publisher.publish(Twist())

        previous_distance = 0
        total_distance = 0

        current_pose = self.getCurrentPosition()
        distance = dist([goal[0],goal[1]],[current_pose.x,current_pose.y])

        while distance > 0.25:

            rotation = self.get_heading()
            current_pose = self.getCurrentPosition()
            path_angle = atan2(goal[1]-current_pose.y , goal[0] - current_pose.x)
            if path_angle < -pi/4 or path_angle > pi/4:
                if goal[1] < 0 and current_pose.y < goal[1]:
                    path_angle = -2*pi + path_angle
                elif goal[1] >= 0 and current_pose.y > goal[1]:
                    path_angle = 2*pi + path_angle
            if last_rotation > pi-0.1 and rotation <= 0:
                rotation = 2*pi + rotation
            elif last_rotation < -pi+0.1 and rotation > 0:
                rotation = -2*pi + rotation

            diff_distance = distance - previous_distance
            distance = dist([goal[0],goal[1]],[current_pose.x,current_pose.y])

            control_signal_distance = self.kp_distance*distance + self.ki_distance*total_distance + self.kd_distance*diff_distance
            control_signal_angle = self.kp_angle*(path_angle - rotation)
            self.twist.angular.z = (control_signal_angle)

            self.twist.linear.x = min(control_signal_distance,0.1)

            if self.twist.angular.z > 0:
                self.twist.angular.z = min(self.twist.angular.z, 1.5)
            else:
                self.twist.angular.z = max(self.twist.angular.z, -1.5)

            last_rotation = rotation
            self.cmd_publisher.publish(self.twist)

            rospy.sleep(1)
            previous_distance = distance
            total_distance = total_distance + distance
```