



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

نام و نام خانوادگی	مهدی رحمانی
شماره دانشجویی	۹۷۳۱۷۰۱
تاریخ ارسال گزارش	۱۴۰۲/۰۳/۱۴

فهرست گزارش سوالات

بخش صفرم (آماده سازی ورک اسپیس برای پروژه).....	۳
گام اول- بخش اول (توضیحات کد و آماده سازی).....	۴
گام اول- بخش اول (اجرا و نتایج).....	۱۳
گام اول-بخش دوم(توضیحات کد و آماده سازی).....	۲۰
گام اول-بخش دوم (اجرا و نتایج).....	۳۲
گام دوم(توضیحات کد و آماده سازی).....	۳۴
گام دوم (اجرا و نتایج).....	۴۴
گام سوم(توضیحات کد و آماده سازی).....	۵۳
گام سوم(اجرا و نتایج).....	۵۹

بخش صفرم (آماده سازی ورک اسپیس برای پروژه)

ابتدا لازم است تا یک work space برای پروژه بسازیم و آن را initialize کنیم:

```
mahdi@mahdi:~/Desktop/Robotics/project3$ mkdir -p hw3_ws/src
mahdi@mahdi:~/Desktop/Robotics/project3$ cd hw3_ws/src/
mahdi@mahdi:~/Desktop/Robotics/project3/hw3_ws/src$ catkin init workspace
Creating symlink "/home/mahdi/Desktop/Robotics/project3/hw3_ws/src/CMakeLists.txt" pointing to "/opt/ros/noetic/share/catkin/cmake/toplevel.cmake"
mahdi@mahdi:~/Desktop/Robotics/project3/hw3_ws/src$ cd ..
mahdi@mahdi:~/Desktop/Robotics/project3/hw3_ws$ catkin make
Base path: /home/mahdi/Desktop/Robotics/project3/hw3_ws
Source space: /home/mahdi/Desktop/Robotics/project3/hw3_ws/src
Build space: /home/mahdi/Desktop/Robotics/project3/hw3_ws/build
Devel space: /home/mahdi/Desktop/Robotics/project3/hw3_ws/devel
Install space: /home/mahdi/Desktop/Robotics/project3/hw3_ws/install
####
### Running command: "cmake /home/mahdi/Desktop/Robotics/project3/hw3_ws/src -DCATKIN_DEVEL_PREFIX=/home/mahdi/Desktop/Robotics/project3/hw3_ws/devel -DCMAKE_INSTALL_PREFIX=/home/mahdi/Desktop/Robotics/project3/hw3_ws/install -G Unix Makefiles" in "/home/mahdi/Desktop/Robotics/project3/hw3_ws/build"
####
-- The C compiler identification is GNU 9.4.0
-- The CXX compiler identification is GNU 9.4.0
-- Check for working C compiler: /usr/bin/cc
-- Check for working CXX compiler: /usr/bin/c++
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
```

چون از turtlebot و شبیه ساز gazebo ممکن است استفاده کنیم لازم است ابتدا از لینک های زیر یک سری پکیج ها را در فولدر src خودمان دانلود کنیم:

- `git clone -b noetic-devel https://github.com/ROBOTIS-GIT/turtlebot3_simulations.git`
- `git clone -b noetic-devel https://github.com/ROBOTIS-GIT/turtlebot3.git`
- `git clone -b noetic-devel https://github.com/ROBOTIS-GIT/turtlebot3_msgs.git`

پس از دانلود اگر ls بزنیم لیست پکیج های داخل فولدر سورس را خواهیم دید:

```
mahdi@mahdi:~/Desktop/Robotics/project3/hw3_ws/src$ git clone -b noetic-devel https://github.com/ROBOTIS-GIT/turtlebot3_simulations.git
Cloning into 'turtlebot3_simulations'...
remote: Enumerating objects: 3160, done.
remote: Counting objects: 100% (721/721), done.
remote: Compressing objects: 100% (142/142), done.
remote: Total 3160 (delta 628), reused 579 (delta 579), pack-reused 2439
Receiving objects: 100% (3160/3160), 15.40 MiB | 2.71 MiB/s, done.
Resolving deltas: 100% (1861/1861), done.
mahdi@mahdi:~/Desktop/Robotics/project3/hw3_ws/src$ git clone -b noetic-devel https://github.com/ROBOTIS-GIT/turtlebot3.git
Cloning into 'turtlebot3'...
remote: Enumerating objects: 6481, done.
remote: Total 6481 (delta 0), reused 0 (delta 0), pack-reused 6481
Receiving objects: 100% (6481/6481), 119.95 MiB | 1.33 MiB/s, done.
Resolving deltas: 100% (4020/4020), done.
mahdi@mahdi:~/Desktop/Robotics/project3/hw3_ws/src$ git clone -b noetic-devel https://github.com/ROBOTIS-GIT/turtlebot3_msgs.git
Cloning into 'turtlebot3_msgs'...
remote: Enumerating objects: 409, done.
remote: Counting objects: 100% (167/167), done.
remote: Compressing objects: 100% (54/54), done.
remote: Total 409 (delta 69), reused 151 (delta 59), pack-reused 242
Receiving objects: 100% (409/409), 90.31 KiB | 1.04 MiB/s, done.
Resolving deltas: 100% (170/170), done.
mahdi@mahdi:~/Desktop/Robotics/project3/hw3_ws/src$ ls
CMakeLists.txt  turtlebot3  turtlebot3_msgs  turtlebot3_simulations
mahdi@mahdi:~/Desktop/Robotics/project3/hw3_ws/src$
```

همچنین در مسیر زیر یک فایل لانچ با نام my_empty_world.launch ایجاد میکنیم که محتوای آن شبیه فایل empty_world که به صورت دیفالت در این مسیر است میباشد ولی کمی تغییرات دارد.

- `hw3_ws/src/turtlebot3_simulations/turtlebot3_gazebo/launch`

```
1 [launch]
2 <arg name="model" default="$(env TURTLEBOT3_MODEL)" doc="model type [burger, waffle, waffle_pi]" />
3 <arg name="x_pos" default="0.0" />
4 <arg name="y_pos" default="0.0" />
5 <arg name="z_pos" default="0.0" />
6 <arg name="yaw" default="0.0" />
7
8 <include file="$(find gazebo_ros)/launch/empty_world.launch">
9   <arg name="world_name" value="$(find turtlebot3_gazebo)/worlds/empty_world" />
10   <arg name="paused" value="false" />
11   <arg name="use_sim_time" value="true" />
12   <arg name="gui" value="true" />
13   <arg name="headless" value="false" />
14   <arg name="debug" value="false" />
15 </include>
16
17 <param name="robot_description" command="$(find xacro)/xacro --inorder $(find turtlebot3_description)/urdf/turtlebot3_$(arg model).urdf.xacro" />
18
19 <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 $(arg yaw) -param robot_description" />
20 </launch>
```

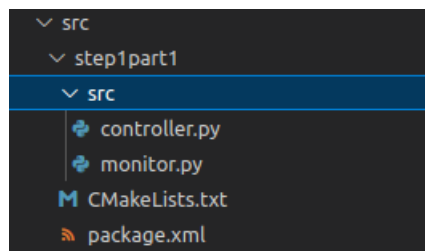
گام اول - بخش اول (توضیحات کد و آماده سازی)

حال لازم است که یک پکیج با نام step1part1 برای گام اول در فولدر سورس بسازیم. همچنین dependency های لازم را که ممکن است در نوشتن نودها و کد زنی نیاز شود به آن میدهیم:

- `catkin_create_pkg step1part1 rospy std_msgs sensor_msgs nav_msgs`

```
mahti@mahti:~/Desktop/Robotics/project3/hw3_ws/src$ catkin_create_pkg step1part1 rospy std_msgs sensor_msgs nav_msgs
Created file step1part1/package.xml
Created file step1part1/CMakeLists.txt
Created folder step1part1/src
Successfully created files in /home/mahti/Desktop/Robotics/project3/hw3_ws/src/step1part1. Please adjust the values in package.xml.
mahti@mahti:~/Desktop/Robotics/project3/hw3_ws/src$
```

حال به سراغ ساخت نودها میرویم. برای این منظور در فولدر src مربوط به step1part1 میرویم و دو فایل پایتون با نام های controller.py و monitor.py میسازیم. نود monitor.py نیز برای نمایش مسیر ربات در rviz به کار گرفته خواهد شد.



حال در گام بعد به سراغ نوشتن کد مربوط به هریک از نودها میرویم. ابتدا کد مربوط به controller را مینویسیم. در این قسمت، در تابع init خود نود را initialize کرده و در گام بعد این نود باید سرعت خطی و زاویه ای را برای ربات در قالب twist پابلیش کند پس نود را به عنوان پابلیشر معرفی میکنیم. سپس ضرایب Kp و Kd و Ki را برای کنترلرهای مختلف تعریف میکنیم تا آنها را tune کنیم.

سپس مقدار dt که time step را برای گرفتن فیدبک مشخص میکند، را تعیین کردیم. مقدار D را هم تعیین کردیم و میخواهیم در حقیقت به مقدار فاصله ۰ از هدف و همچنین ارور زاویه ای ۰ برسیم. ما در اینجا کنترلر را برای سرعت خطی و هم سرعت زاویه ای مینویسیم و میخواهیم در نهایت فاصله ما تا هدف برابر ۰ باشد و ارور زاویه ای هم به مقدار ۰ برسد. بنابراین gain هایی مثل k_{d_l} , k_{p_l} و k_{i_l} که به زیروند l مشخص شدند مربوط به سرعت خطی میباشند و k_{d_a} و k_{p_a} و k_{i_a} که با زیروند a مشخص شدند برای سرعت زاویه ای هستند. همچنین مقدار متغیر threshold برای آن است که بگوییم اگر تا این حد (چه از نظر زاویه و چه از نظر فاصله) ربات به مقصد و مقدار دلخواه

نزدیک شد برنامه را تمام کن و نمودارها را نمایش بده. نمیتوان شرط دقیقا ۰ گذاشت چراکه اولاً همیشه یک مقدار کمی خطا وجود خواهد داشت و اگر هم به نحوی قابل دسترس باشد زمانبر خواهد بود.

سپس یک تابع `get_heading` و همچنین `get_pose` داریم که به ترتیب زاویه `yaw` ربات را به رادیان و دیگری مکان `x` و `y` ربات را برحسب متر با خواندن مقادیر از تاپیک `odom` برمیگرداند.

یک تابع `distance_from_goal` داریم که فاصله اقلیدسی تا هدف را حساب میکند و خروجی میدهد. تابع دیگر `angle_from_goal` میباشد. در این تابع ما ابتدا زاویه `heading` ربات را میابیم و همچنین زاویه نقطه هدف نسبت به ربات را هم می‌یابیم. اختلاف زاویه هدینگ و این زاویه `desired` میشود ارور زاویه‌ای ما. باید دقت شود چنانچه این مقدار بیشتر از 180° درجه یا کمتر از -180° درجه شد باید آن را از 360° کم کنیم و به عنوان ارور اعلام کنیم. چراکه به جای آنکه ربات از سمت زاویه منفی به چرخش در بیاید که خطا را بالا میبرد و ربات ناپایدار میشود میتوان از زاویه حاده استفاده کرد.

در تابع `controller` مقدار ارور را برای فاصله و محاسبه سرعت خطی برابر با اختلاف `distance` حساب شده بین نقطه فعلی تا مقصد و همچنین مقدار `D` تعریف میکنیم که ۰ بود. یک مقدار `sum_i` داریم که برابر خطای جمع شونده است و درواقع مساحت و انتگرال زیر نمودار `err` را حساب میکند که زمانی که بخواهیم از ترم انتگرال گیر استفاده کنیم به کار می‌آید. همین پارامترها را برای زاویه و محاسبه سرعت زاویه‌ای نیز جداگانه تعریف میشوند.

سپس باتوجه به مطالب داخل درس و فیلم‌های قرار داده شده، مقدار هریک از ترم‌های `P` و `I` و `D` را تعیین میکنیم. باتوجه به تعریف ارروی که داشتیم مقدار ترم `P` میشود حاصل ضرب ضریب `Kp` در آن ارور و همچنین مقدار ترم `D` میشود حاصل ضرب، ضریب `Kd` در نرخ تغییرات ارور (حالت مشتقی دارد) و مقدار `I` میشود ضرب ضریب `Ki` در مقدار انتگرال گیری شده روی خطا که همان `sum_i` بود. مجموع این ۳ ترم در حالتی که مربوط به فاصله باشند سرعت خطی و درحالتی که مربوط به زاویه باشند سرعت زاویه‌ای را میسازند.

در نهایت هم میگوییم پس از تمام شدن اجرا مقدار خطا را در طول اجرا پلات کند. این کار به کمک تابع `on_shutdown` انجام میشود. در کد زیر مقادیر نهایی ضرایب آمده است و در ادامه طریقه به دست آوردن هریک را میگوییم. دقت شود که برای استفاده از هر کنترلر باید مقدار ضرایب آن `uncomment` شود و بقیه کامنت شوند.

```
#!/usr/bin/python3

import rospy
import tf
from geometry_msgs.msg import Twist
import matplotlib.pyplot as plt
from nav_msgs.msg import Odometry
from math import atan2, sqrt, radians

class PIDController():

    def __init__(self):
        # initialize node and define it as a publisher
        rospy.init_node('controller', anonymous=False)
        rospy.on_shutdown(self.on_shutdown)
        self.cmd_publisher = rospy.Publisher('/cmd_vel' , Twist , queue_size=10)

        # (linear Velocity)

        # P controller
        #self.k_p_l = 0.08
        #self.k_i_l = 0.0
        #self.k_d_l = 0.0

        # PD controller
        # self.k_p_l = 0.1
        # self.k_i_l = 0.0
        # self.k_d_l = 0.2

        # PID controller
        self.k_p_l = 0.1
        self.k_i_l = 0.0005
        self.k_d_l = 0.2

        # (angular Velocity)

        # P controller
        #self.k_p_a = 0.3
        #self.k_i_a = 0.0
        #self.k_d_a = 0.0

        # PD controller
        # self.k_p_a = 0.3
        # self.k_i_a = 0.0
        # self.k_d_a = 0.9

        # PID controller
        self.k_p_a = 0.3
```

```

self.k_i_a = 0.121
self.k_d_a = 0.9

# goal pose
self.x_goal = 10
self.y_goal = 0

self.D = 0
self.threshold = 0.005

self.dist_errs = []
self.angle_errs = []

self.dt = 0.005
rate = 1/self.dt
self.r = rospy.Rate(rate)

def get_pose(self):
    """
    get x and y coordinate of position of the robot
    """
    # waiting for the most recent message from topic /odom
    msg = rospy.wait_for_message("/odom" , Odometry)

    position = msg.pose.pose.position

    return position.x, position.y

def get_heading(self):
    """
    get the yaw angle of robot in world.
    We call it, heading of the robot.
    """
    # 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

def distance_from_goal(self):
    """
    this function get us the current Euclidean distance from goal

```

```

    ...

    x_curr, y_curr = self.get_pose()
    distance = sqrt((self.x_goal-x_curr)**2 + (self.y_goal-y_curr)**2)

    return distance

def angle_from_goal(self):
    ...

    function below first calculate the heading angle and then the desired angle from current
pose to goal pose.

    it means that robot heading must be equal to this angle for being in a
correct direction. then return the difference between heading and desired angle.
    ...

    # find x and y of current position and find relative x and y to goal point
    x_curr, y_curr = self.get_pose()
    relative_x = self.x_goal - x_curr
    relative_y = self.y_goal - y_curr
    # get heading of robot in radian
    heading = self.get_heading()
    # now we should find the desired angle
    # desired angle tell us the angle of goal point relative to current point
    desired_angle = 0
    if (relative_x == 0 and relative_y ==0):
        # this state (x=0 , y =0) is undefined so we handle it separately
        desired_angle = heading
    else:
        desired_angle = atan2(relative_y, relative_x)
    # the angle express howmuch we should rotate to reach the desired angle
    angle = heading - desired_angle
    # but we design controller and if the angle is bigger than 180 or less than -180
    # the robot must rotate alot so we should find its complementary to 360 degrees
    if angle < radians(-180):
        angle = radians(360)-abs(angle)
    elif angle > radians(180):
        angle = angle-radians(360)

    return angle

def control(self):
    ...

    this function is the main function of this code. we calculate the angular and
linear distance error and try to calculate P, I, D terms. the summation of these
terms define our linear and angular velocity.

    also we define a threshold and if the robot close to goal point and the goal and
angle distance are less than that threshold the programe is terminated.
    ...

    distance = self.distance_from_goal()

```



```

sum_i_dist = 0
prev_error_dist = 0

angle = self.angle_from_goal()
sum_i_angle = 0
prev_error_angle = 0

move_cmd = Twist()

while not rospy.is_shutdown():

    # linear velocity
    err_dist = distance - self.D
    self.dist_errs.append(err_dist)
    sum_i_dist += err_dist * self.dt

    P_l = self.k_p_l * err_dist
    I_l = self.k_i_l * sum_i_dist
    D_l = self.k_d_l * (err_dist - prev_error_dist)

    move_cmd.linear.x = P_l + I_l + D_l
    prev_error_dist = err_dist
    distance = self.distance_from_goal()

    rospy.loginfo(f"linear velocity")
    rospy.loginfo(f"P_l : {P_l} I_l : {I_l} D_l : {D_l}")

    # angular velocity
    err_angle = angle - self.D
    self.angle_errs.append(err_angle)
    sum_i_angle += err_angle * self.dt

    P_a = self.k_p_a * err_angle
    I_a = self.k_i_a * sum_i_angle
    D_a = self.k_d_a * (err_angle - prev_error_angle)

    move_cmd.angular.z = -(P_a + I_a + D_a)
    self.cmd_publisher.publish(move_cmd)
    prev_error_angle = err_angle
    angle = self.angle_from_goal()

    rospy.loginfo(f"angular velocity")
    rospy.loginfo(f"P_a : {P_a} I_a : {I_a} D_a : {D_a}")

    rospy.loginfo(f"error_angle : {err_angle} error_dist: {err_dist} angular speed :
{move_cmd.angular.z} linear speed : {move_cmd.linear.x}")

    if err_dist < self.threshold and err_angle < self.threshold:

```

```

        break

        self.r.sleep()

def on_shutdown(self):
    """
    this method plot error of linear and angular velocity separately.
    """
    rospy.loginfo("Stopping the robot...")
    self.cmd_publisher.publish(Twist())
    # linear
    plt.plot(list(range(len(self.dist_errs))), self.dist_errs, label='dist_errs')
    plt.axhline(y=0,color='R')
    plt.draw()
    plt.legend(loc="upper left", frameon=False)
    plt.savefig(f"errors_dist_{self.k_p_l}_{self.k_d_l}_{self.k_i_l}.png")
    plt.show()

    # angular
    plt.plot(list(range(len(self.angle_errs))), self.angle_errs, label='angle_errs')
    plt.axhline(y=0,color='R')
    plt.draw()
    plt.legend(loc="upper left", frameon=False)
    plt.savefig(f"errors_angle_{self.k_p_a}_{self.k_d_a}_{self.k_i_a}.png")
    plt.show()
    rospy.sleep(1)

if __name__ == '__main__':
    try:
        pidc = PIDController()
        pidc.control()
    except rospy.ROSInterruptException:
        rospy.loginfo("Navigation terminated.")

```

در monitor کد پایتون مربوط به رسم مسیری که ربات آن را طی میکند میباشد که از تایپیک path استفاده میکند. مکان هایی که می رود را در قالب یک آرایه یا لیست ذخیره میکنیم و در rviz آن را نشان میدهیم.

```
#!/usr/bin/python3

import rospy
from nav_msgs.msg import Odometry, Path
from geometry_msgs.msg import PoseStamped
class PathMonitor:

    def __init__(self) -> None:

        rospy.init_node("monitor" , anonymous=False)
        self.path = Path()
        self.odom_subscriber = rospy.Subscriber("/odom" , Odometry , callback=self.odom_callback)
        self.path_publisher = rospy.Publisher("/path" , Path , queue_size=10)

    def odom_callback(self, msg : Odometry):
        self.path.header = msg.header
        pose = PoseStamped()
        pose.header = msg.header
        pose.pose = msg.pose.pose
        self.path.poses.append(pose)
        self.path_publisher.publish(self.path)

if __name__ == "__main__":
    path_monitor = PathMonitor()
    rospy.spin()
```

در مرحله بعد باید تمامی کدهای پایتون را executable کنیم. برای این کار لازم است در ترمینال در پکیج step1part1 کد زیر را اجرا کنیم:

chmod +x src/*.py

```
mahdi@mahdi:~/Desktop/Robotics/project3/hw3_ws/src/step1part1$ chmod +x src/*.py
mahdi@mahdi:~/Desktop/Robotics/project3/hw3_ws/src/step1part1$ cd src/
mahdi@mahdi:~/Desktop/Robotics/project3/hw3_ws/src/step1part1/src$ ls
controller.py  monitor.py
```

سپس به سراغ نوشتن لانچ فایل میرویم. در همین پکیج step1part1 لازم است تا یک فولدر launch ایجاد کنیم و داخل آن یک step1part1.launch ایجاد کنیم. لانچ فایل به صورت زیر است:

```
<launch>

  <include file="$(find turtlebot3_gazebo)/launch/my_empty_world.launch">
    <arg name="x_pos" value="0.0"/>
    <arg name="y_pos" value="0.0"/>
    <arg name="z_pos" value="0.0"/>
    <arg name="yaw" value="0.0"/>
  </include>

  <include file="$(find turtlebot3_gazebo)/launch/turtlebot3_gazebo_rviz.launch"/>

  <node pkg="step1part1" type="controller.py" name="controller" output="screen">
    <param name="angular_speed" value="0.02"/>
  </node>

  <node pkg="step1part1" type="monitor.py" name="monitor"></node>

</launch>
```

سپس در آخر لازم است تا به دایرکتوری ورک اسپیس برویم و catkin_make را صدا بزنیم. سپس برای استفاده لازم است تا ابتدا سورس کنیم و سپس ربات را اکسپورت کنیم و در نهایت roslaunch را صدا بزنیم:

- . devel/setup.bash
- export TURTLEBOT3_MODEL=waffle
- roslaunch step1part1 control.launch

گام اول - بخش اول (اجرا و نتایج)

توضیحات به دست آوردن ضرایب برای هر کنترلر:

باتوجه به اسلاید زیر مقدار ضرایب را تعیین میکنیم.

TABLE 1 Effects of independent P, I, and D tuning on closed-loop response.
For example, while K_I and K_D are fixed, increasing K_P alone can decrease rise time, increase overshoot, slightly increase settling time, decrease the steady-state error, and decrease stability margins.

	Rise Time	Overshoot	Settling Time	Steady-State Error	Stability
Increasing K_P	Decrease	Increase	Small Increase	Decrease	Degrade
Increasing K_I	Small Decrease	Increase	Increase	Large Decrease	Degrade
Increasing K_D	Small Decrease	Decrease	Decrease	Minor Change	Improve

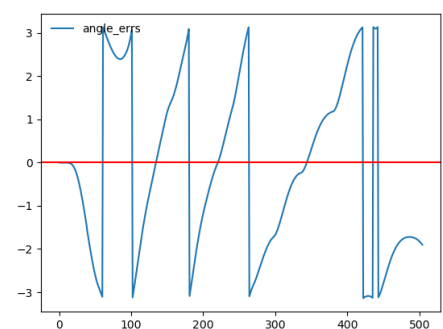
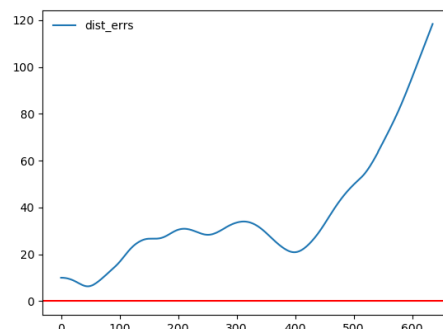
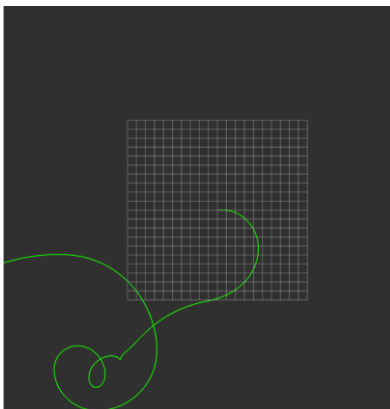
Challenge: Their effects are intertwined; they need to be set altogether!

(One typical, **manual**) Heuristic recipe:

1. Start with a **P controller** and set P gain to a value K_P that's low enough to prevent appearing of major oscillations in system's response.
2. Add derivative **D**, that will help to damp oscillations when P gain will be increased (to get faster response). Start with $K_P = cK_P$, if not oscillations appear, increase it, until oscillations happen, or, if oscillations are there, decrease it until they disappear (e.g., $c = 10$)
3. Adjust P's gain to **possibly increase it** (by a factor 2 or 3), until oscillations appear.
4. Start the **I gain** setting it to about 1/100 of P's gain. With oscillations, decrease K_I ; with no oscillations, increase it until oscillations happen.
5. ... stop! → Try it out ...

• کنترلر P

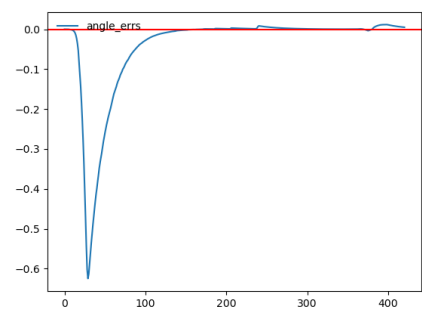
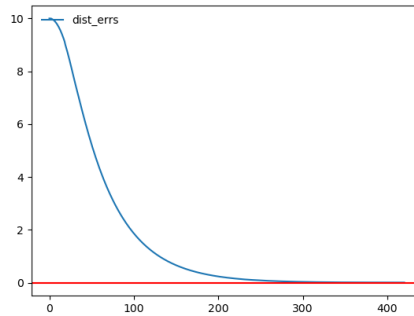
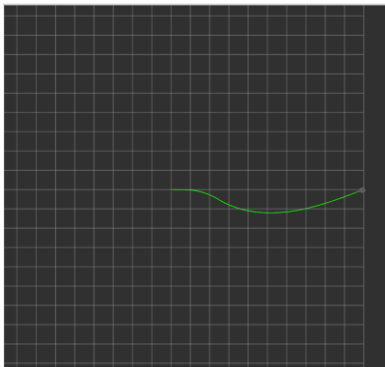
حالت اول) طبیعتاً در این کنترلر مقادیر مربوط به K_i و K_d برابر با ۰ میباشد. برای تعیین ترم proportional دقت شود که در اسلایدهای هم اشاره شده بود که از مقدار زیاد K_p پرهیز شود. اگر مثلاً مقدار K_{p_l} (برای سرعت خطی) و K_{p_a} (برای سرعت زاویه ای) را به ترتیب برابر ۳ و ۶ بگیریم، در این صورت خواهیم دید که oscillation زیاد خواهد بود و ارور ما زیاد است. این مورد را میتوانید در شکل مشاهده کنید.



```
# (linear Velocity) # (angular Velocity)

# P controller      # P controller
self.k_p_l = 3       self.k_p_a = 6
self.k_i_l = 0.0     self.k_i_a = 0.0
self.k_d_l = 0.0     self.k_d_a = 0.0
```

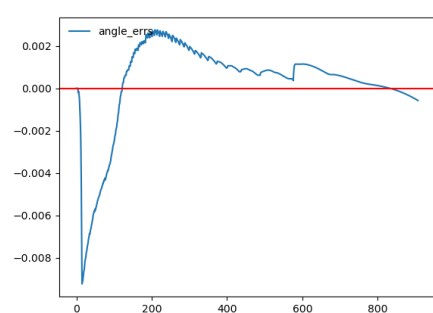
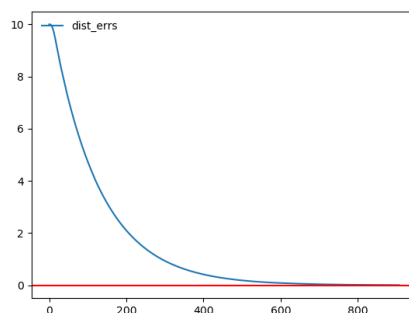
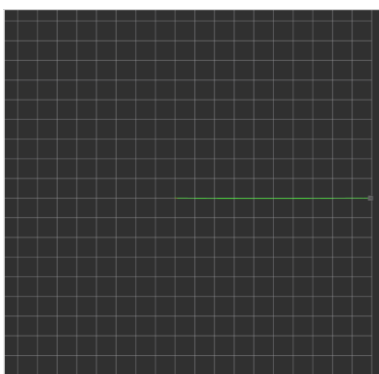
حالت دوم) پس مقدار آنها را کمتر کرده و برای سرعت خطی و زاویه‌ای به ترتیب برابر با 0.2 و 0.6 میگذاریم. سپس نتیجه را مشاهده میکنیم. همانطور که دیده میشود ابتدای مسیر انحراف از مسیر مستقیم زیاد شده است ولی در نهایت به مقصد رسیده است و ارور نزدیک به ۰ شده است.



```
# (linear Velocity)      # (angular Velocity)

# P controller          # P controller
self.k_p_l = 0.2        self.k_p_a = 0.6
self.k_i_l = 0.0        self.k_i_a = 0.0
self.k_d_l = 0.0        self.k_d_a = 0.0
```

حالت سوم (نهایی)) در حالت قبل به مقصد رسیدیم ولی در مسیر انحرافات داشتیم. مقدار را خیلی کمتر کرده و پس از کمی سعی و خطا به مقادیر 0.08 و 0.3 برای سرعت خطی و زاویه‌ای میرسیم. همانطور که میبینیم مقدار خطا به مرور کم شده و نمودار به ۰ همگرا شده. در این جا همانطور که دیده میشود ربات روی یک خط صاف حرکت کرده و از مسیر مستقیم منحرف نشدیم. برای دقت بیشتر میتوان threshold و همچنین مقادیر gain را کمتر کرد تا ربات اهسته تر و با دقت بیشتر پیش برود.



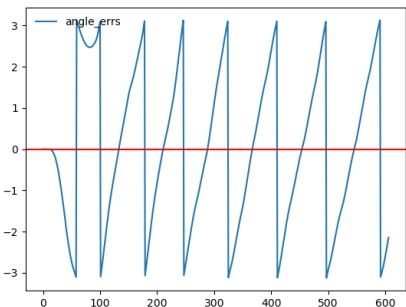
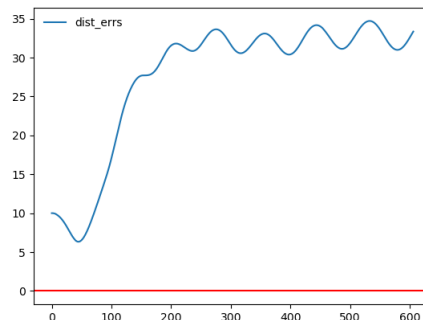
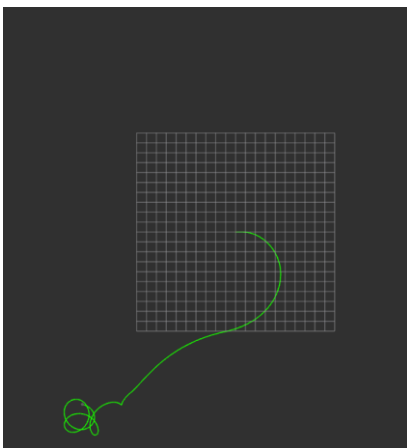
```
# (linear Velocity)      # (angular Velocity)

# P controller          # P controller
self.k_p_l = 0.08       self.k_p_a = 0.3
self.k_i_l = 0.0        self.k_i_a = 0.0
self.k_d_l = 0.0        self.k_d_a = 0.0
```

• کنترلر PD

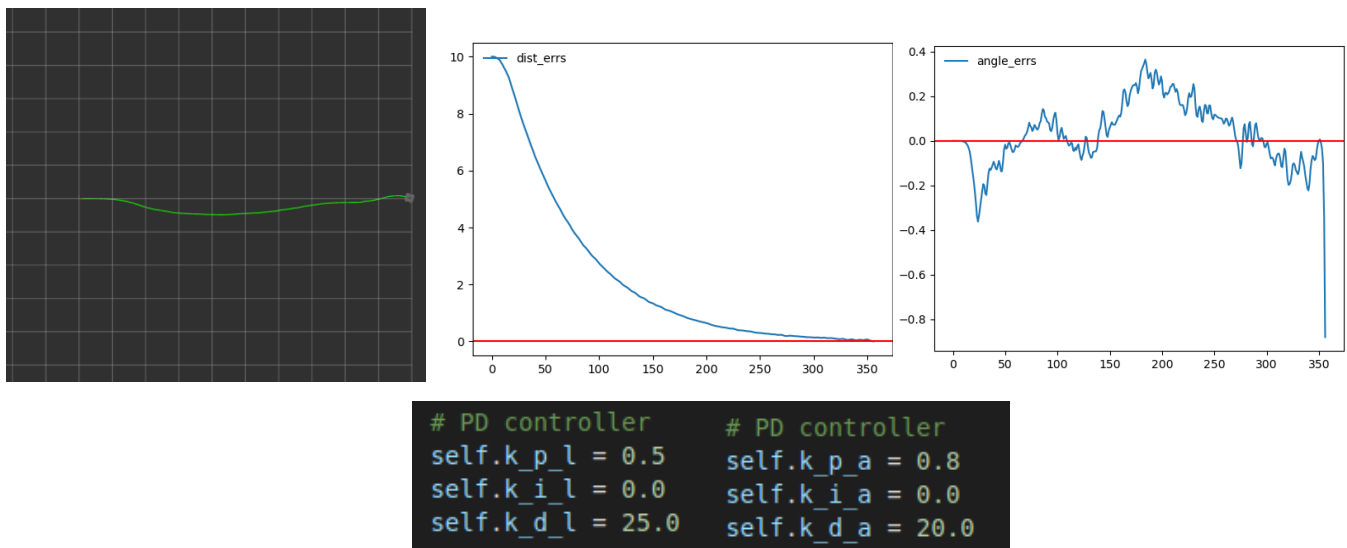
میدانیم کاربرد D برای جلوگیری از Overshoot میباشد. طبق توضیحات اسلاید یک مقدار اولیه مناسب برای مقدار D برابر با 0.1 مقدار k_p میباشد. در قسمت قبل چون صرفاً کنترلر P داشتیم باید gain را کم میکردیم تا به دقت کافی برسیم اما اکنون که میتوان D داشت پس میتوان کمی ضرایب P را بالاتر برد و جلوی اورشوت را با D گرفت. درواقع هم در زمان کمتری به مقصد برسیم و هم دقت نسبتاً خوبی داشته باشیم. (طبیعتاً با مقادیر K_p پایین دقت بهتر است و مقدار K_d کم هم جوابگوست) حتی نبودشان هم اوکیه ☺))

حالت اول) در این حالت ما مقدار K_p را برای سرعت خطی و زاویه ای به ترتیب برابر با 0.5 و 0.8 میگذاریم و مقدار K_d را برابر با 0.05 و 0.08 برای شروع تنظیم میکنیم. خواهیم دید که نتایج قابل قبول نیستند مقدار ارور بالا و نوسانی میباشد و درحقیقت مقدار gain مربوط به K_d برای این مقادیر proportional خیلی کم است.

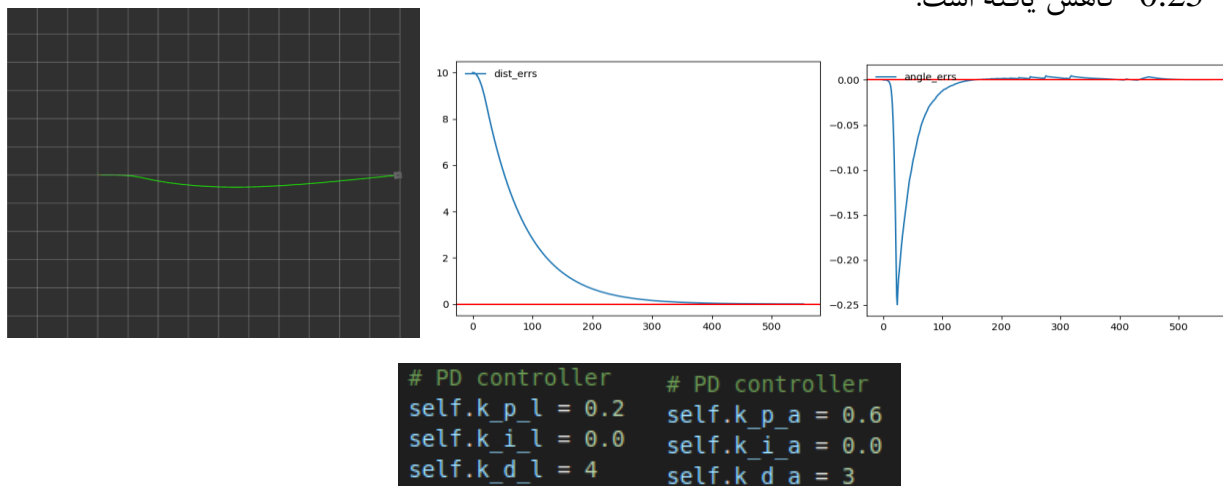


```
# PD controller      # PD controller
self.k_p_l = 0.5      self.k_p_a = 0.8
self.k_i_l = 0.0      self.k_i_a = 0.0
self.k_d_l = 0.05     self.k_d_a = 0.08
```

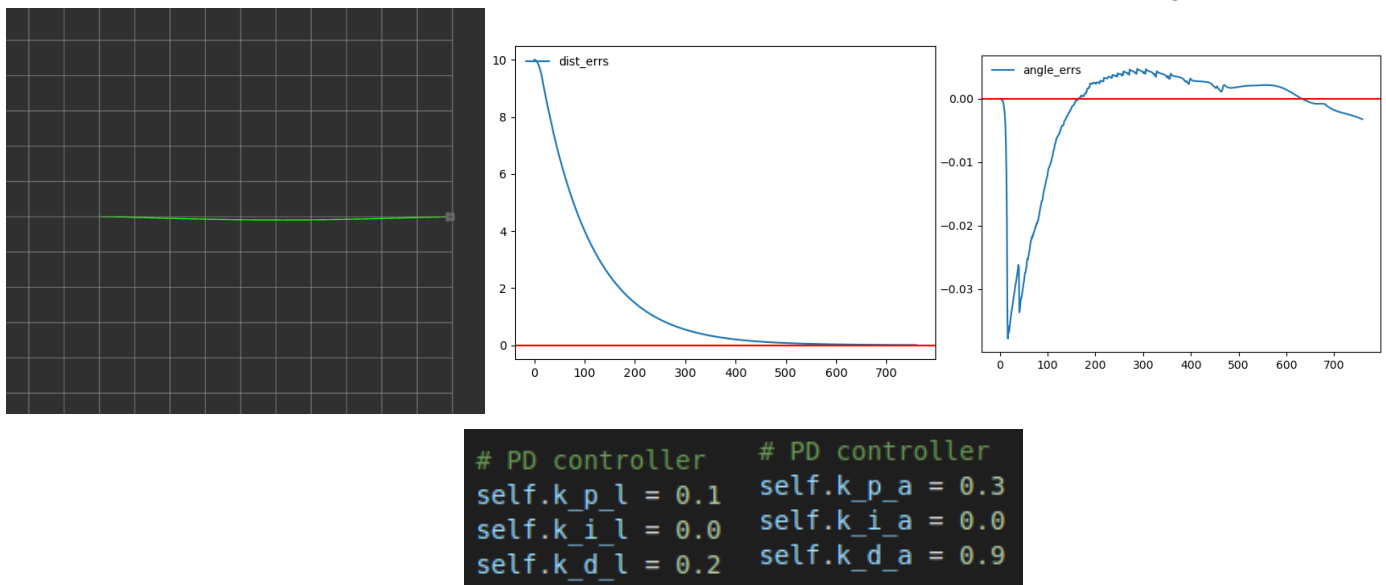
حالت دوم) حال با سعی و خطا اگر برای همین مقادیر K_p مقادیر مربوط به K_d را زیادتر کنیم و به ترتیب برای سرعت خطی و زاویه‌ای برابر با ۲۵ و ۲۰ شکل و نمودارهای زیر را مشاهده خواهید کرد. در این صورت در عین حال که سرعت بالا هست به مقصد هم میرسیم ولی همانطور که دیده میشود از مسیر انحراف داریم و همچنین نوسانات زیاد است. درواقع در اینجا چون مقادیر K_d gain های بالا هست نویز پذیری سیستم نیز افزایش میابد و درواقع ترم مشتق گیر اهمیت پیدا میکند و به تغییرات خیلی حساس است. بنابراین لازم است سرعت را کمتر کرده و از K_d gain های کمتری استفاده کنیم.



حالت سوم) برای همان حالتی که در P-controller داشتیم که مقادیر K_p gain های برای سرعت خطی و زاویه‌ای به ترتیب برابر با ۰.۲ و ۰.۶ بودند اگر K_d gain های را به ترتیب ۰.۰۲ و ۰.۰۶ گذاشته و با حدس و خطا آن را زیاد کنیم با مقادیر K_d کمتر نسبت به قبل که برابر با ۴ و ۳ هستند میتوان نتیجه زیر را گرفت. میتوان دید که مقدار Overshoot و انحراف از مسیر نسبت به حالتی که فقط K_p داشتیم کمتر شده است. در نمودار ارور زاویه ای هم مقدار اورشوت از -۰.۶ در قسمت قبل به -۰.۲۵ کاهش یافته است.



حالت چهارم (نهایی) اگر بخواهیم روی مسیر مستقیم پیش برویم و در عین حال سیستم ما نویز پذیر نباشد لازم است مقادیر K_p و K_d کمتر باشند. در این حالت اگرچه سرعت کمتر است ولی دقت بیشتر است هرچند مقادیر $gain$ کمتر از این و مقدار $threshold$ کمتر در رسیدن به نقطه انتهایی کمتر از این هم باشند دقت بیشتر میشود. در اینجا برای ارور زاویه اگر دقت شود نسبت به حالت سوم برای P -controller مقدار $overshoot$ نصف شده است. (با وجود اینکه مقدار K_p هم زیاد شده و سرعت بیشتر است) همچنین در این حالت مقادیر K_d مناسب است و خیلی بزرگ نیست که خیلی به نویز حساس باشد.

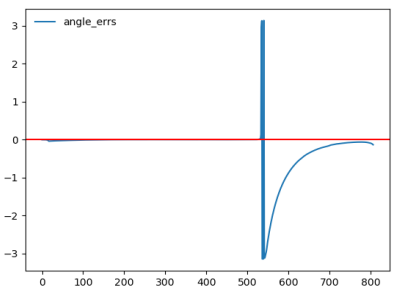
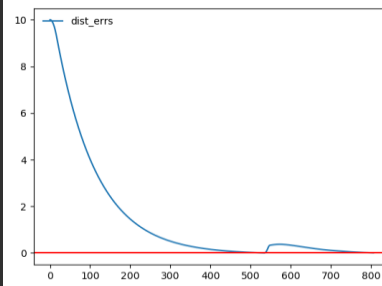
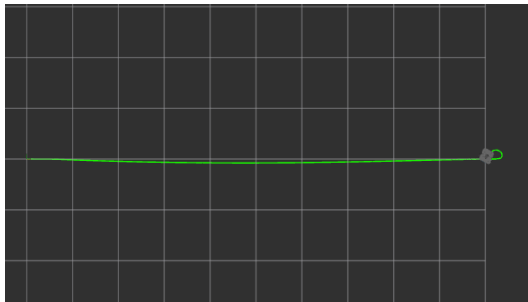


• کنترلر PID

میدانیم که ترم I در PID برای برطرف کردن $steady\ state\ error$ میباشد. همانطور که در فیلم هندزان هم دیدیم در اینجا ما خیلی با $steady\ state\ error$ روبرو نیستیم و مثلاً توی یک کوادکوپتر که گرانش g داریم شاید بیشتر مطرح باشد. به هر حال به $stability$ روبات کمک میکند. همچنین دقت شود که I میتواند منجر به $overshoot$ هم شود پس باید در استفاده از آن دقت داشت. طبق اسلاید مقدار اولیه مناسب برای شروع 0.01 مقدار K_p میباشد.

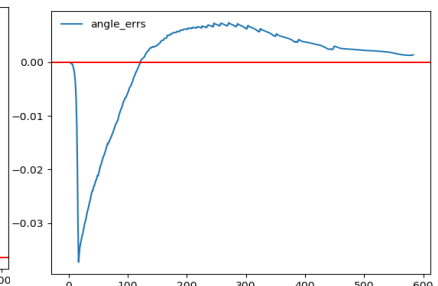
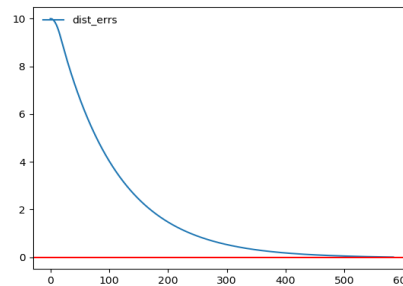
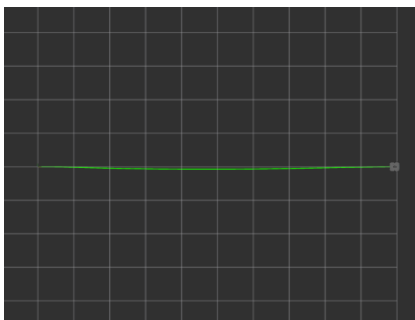
حالت اول مقادیر K_p و K_d مانند حالت چهارم کنترلر PD میماند و مقدار K_i برای سرعت خطی و زاویه‌ای به ترتیب برابر با 0.001 و 0.003 میگیریم.

نتایج در صفحه بعد قابل مشاهده است. در این حالت به نظر میرسد مقدار k_{i_l} باید کمتر شود و به علت انباشت خطاهای قبلی باعث شده است که در انتها ربات به آرامی پیش نرود و مقصد را رد کند.



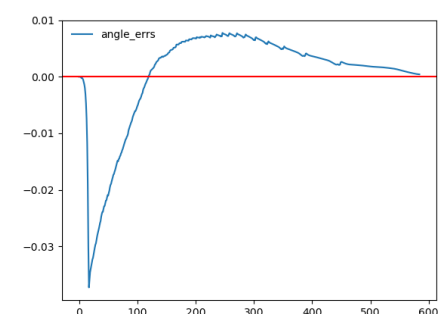
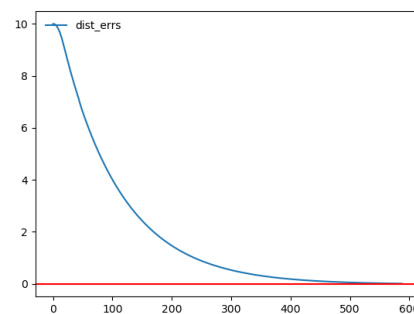
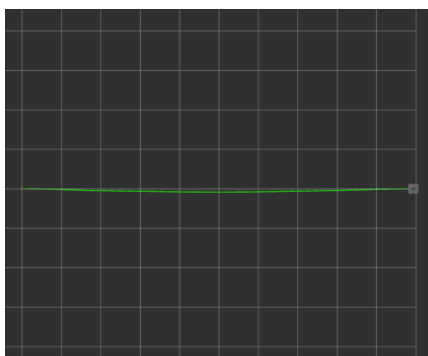
```
# PID controller # PID controller
self.k_p_l = 0.1 self.k_p_a = 0.3
self.k_i_l = 0.001 self.k_i_a = 0.003
self.k_d_l = 0.2 self.k_d_a = 0.9
```

حالت دوم) دقت شود همانطور که از نمودارهای خطا میتوان متوجه شد خطای فاصله خیلی نیازی به اصلاح ندارد و باید مقدار k_{i_l} کمتر شود. مقدار k_{i_a} را بیشتر میکنیم. به ترتیب این دو را برابر با 0.0005 و 0.1 میگذاریم. در این صورت خروجی به صورت زیر میشود. همانطور که میبینیم کمی خطای steady state در زاویه داریم.



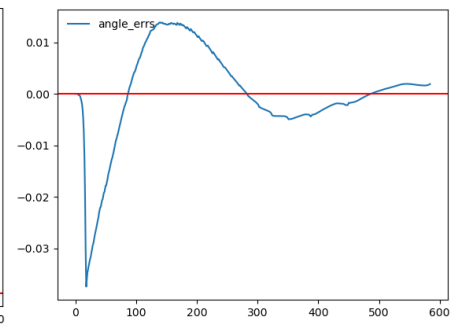
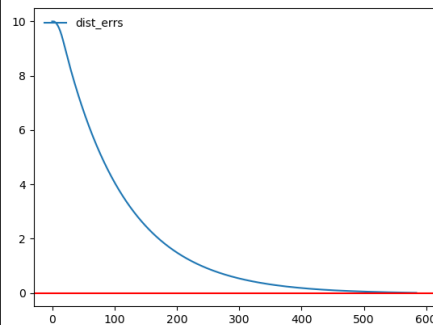
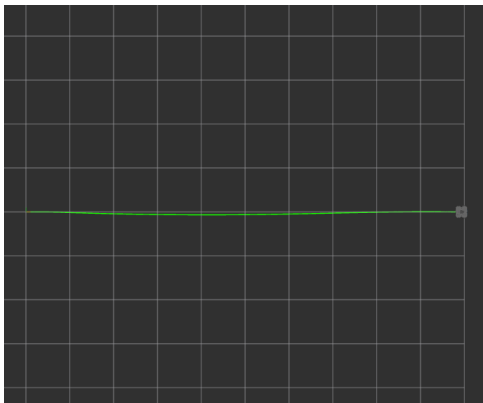
```
# PID controller # PID controller
self.k_p_l = 0.1 self.k_p_a = 0.3
self.k_i_l = 0.0005 self.k_i_a = 0.1
self.k_d_l = 0.2 self.k_d_a = 0.9
```

حالت سوم (نهایی): حال برای برطرف کردن خطای steady state در زاویه با سعی و خطا کمی آن را افزایش میدهم و به مقدار 0.121 میرسیم. اگر دقت شود در نهایت به آرامی خطای زاویه به 0 میرسد و مقادیر overshoot هم کم میباشد.



```
# PID controller # PID controller
self.k_p_l = 0.1 self.k_p_a = 0.3
self.k_i_l = 0.0005 self.k_i_a = 0.121
self.k_d_l = 0.2 self.k_d_a = 0.9
```

حالت چهارم) حال اگر مقدار k_{i_a} را بیشتر کنیم و برابر با ۰.۶ بگذاریم خواهیم دید که هم overshoot در نمودار خطای زاویه‌ای بیشتر میشود و هم ربات کمی حول خطای 0 نوسان کرده است و به آرامی همگرا نشده است که خوب نیست.



```
# PID controller
self.k_p_l = 0.1
self.k_i_l = 0.0005
self.k_d_l = 0.2

# PID controller
self.k_p_a = 0.3
self.k_i_a = 0.6
self.k_d_a = 0.9
```

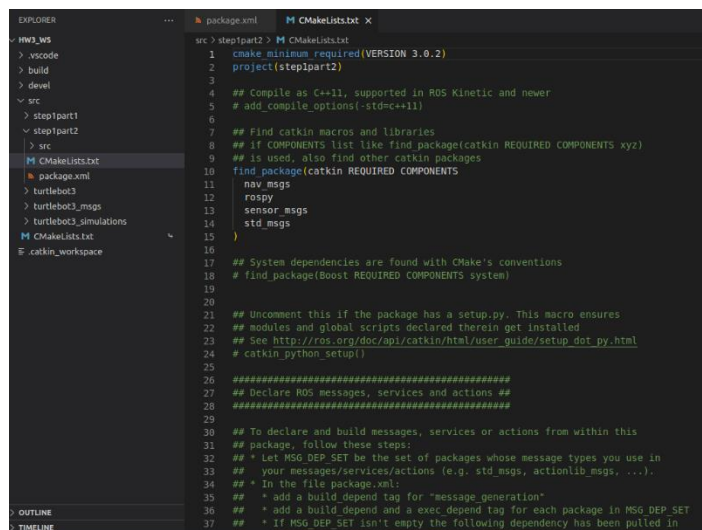
گام اول-بخش دوم(توضیحات کد و آماده سازی)

ابتدا لازم است یک پکیج برای این بخش نیز بسازیم و dependencyهایی که ممکن است به کار بیایند را هم اضافه کنیم.

- catkin_create_pkg step1part2 rospys std_msgs sensor_msgs nav_msgs

```
mahdi@mahdi:~/Desktop/Robotics/project3/hw3_ws$ cd src/
mahdi@mahdi:~/Desktop/Robotics/project3/hw3_ws/src$ catkin_create_pkg step1part2 rospys std_msgs sensor_msgs nav_msgs
Created file step1part2/package.xml
Created file step1part2/CMakeLists.txt
Created folder step1part2/src
Successfully created files in /home/mahdi/Desktop/Robotics/project3/hw3_ws/src/step1part2. Please adjust the values in
package.xml.
mahdi@mahdi:~/Desktop/Robotics/project3/hw3_ws/src$
```

حال باید سرویس مورد نظر را درون پکیج step1part2 ایجاد کنیم. برای این کار کل ورک اسپیس را درون vs code باز میکنیم. از درون پکیج step1part2 باید دو فایل CMakeLists.txt و packages.xml را باز کنیم تا تغییرات لازم را اعمال کنیم:



```
EXPLORER
src
  step1part2
    src
      CMakeLists.txt
      package.xml
    turtlebot3
    turtlebot3_msgs
    turtlebot3_simulations
  catkin_workspace

src > step1part2 > M CMakeLists.txt
1 cmake_minimum_required(VERSION 3.0.2)
2 project(step1part2)
3
4 ## Compile as C++11, supported in ROS Kinetic and newer
5 # add_compile_options(-std=c++11)
6
7 ## Find catkin macros and libraries
8 ## if COMPONENTS list like find_package(catkin REQUIRED COMPONENTS xyz)
9 ## is used, also find other catkin packages
10 find_package(catkin REQUIRED COMPONENTS
11   nav_msgs
12   rospys
13   sensor_msgs
14   std_msgs
15 )
16
17 ## System dependencies are found with CMake's conventions
18 # find_package(Boost REQUIRED COMPONENTS system)
19
20
21 ## Uncomment this if the package has a setup.py. This macro ensures
22 ## modules and global scripts declared therein get installed
23 ## See http://ros.org/doc/api/catkin/html/user_guide/setup_dot_py.html
24 # catkin_python_setup()
25
26 #####
27 ## Declare ROS messages, services and actions ##
28 #####
29
30 ## To declare and build messages, services or actions from within this
31 ## package, follow these steps:
32 ## * Let MSG_DEP_SET be the set of packages whose message types you use in
33 ##   your messages/services/actions (e.g. std_msgs, actionlib_msgs, ...).
34 ## * In the file package.xml:
35 ##   * add a build_depend tag for "message_generation"
36 ##   * add a build_depend and a exec_depend tag for each package in MSG_DEP_SET
37 ##   * If MSG_DEP_SET isn't empty the following dependency has been pulled in
```

۱- ابتدا در فایل package.xml در خط ۵۵ اینتر زده و بعد build_depend جدید برای message_generation ایجاد کنیم و در خط ۶۵ هم یک exec_depend ایجاد میکنیم:

```
50 <!-- <doc_depend>doxygen</doc_depend> -->
51 <buildtool_depend>catkin</buildtool_depend>
52 <build_depend>nav_msgs</build_depend>
53 <build_depend>rospys</build_depend>
54 <build_depend>sensor_msgs</build_depend>
55 <build_depend>std_msgs</build_depend>
56 <build_depend>message_generation</build_depend>
57 <build_export_depend>nav_msgs</build_export_depend>
58 <build_export_depend>rospys</build_export_depend>
59 <build_export_depend>sensor_msgs</build_export_depend>
60 <build_export_depend>std_msgs</build_export_depend>
61 <exec_depend>nav_msgs</exec_depend>
62 <exec_depend>rospys</exec_depend>
63 <exec_depend>sensor_msgs</exec_depend>
64 <exec_depend>std_msgs</exec_depend>
65 <exec_depend>message_runtime</exec_depend>
66
```

۲- سپس در CMakeLists هم در خط ۱۵ message_generation را میگذاریم:

```
> devel
  > src
    > step1part1
      > step1part2
        > src
          M CMakeLists.txt
          package.xml
        > turtlebot3
        > turtlebot3_msgs
        > turtlebot3_simulations
      7
      8 ## Find catkin macros and libraries
      9 ## if COMPONENTS list like find_package(catkin REQUIRED COMPONENTS xyz)
      10 ## is used, also find other catkin packages
      11 find_package(catkin REQUIRED COMPONENTS
      12   nav_msgs
      13   rospy
      14   sensor_msgs
      15   std_msgs
      16   message_generation
      17 )
```

۳- خطوط ۵۹ تا ۶۳ که مربوط به سرویس هست را uncomment میکنیم و تغییرات را اعمال میکنیم:

```
58 ## Generate services in the 'srv' folder
59 add_service_files(
60   FILES
61   GetNextDestination.srv
62 )
63
```

۴- خطوط ۷۲ تا ۷۷ هم uncomment شوند:

```
70
71 ## Generate added messages and services with any dependencies listed here
72 generate_messages(
73   DEPENDENCIES
74   nav_msgs
75   sensor_msgs
76   std_msgs
77 )
78
```

۵- برای آنکه کارمان تکمیل شود، در همان پوشه step1part2 یک پوشه به نام srv درست میکنیم و بعد داخل آن یک فایل با نام اونی که در استپ ۳ مشخص کردیم میذاریم.

```
> step1part2
  > src
    > srv
      GetNextDestination.srv
    M CMakeLists.txt
    package.xml
  > turtlebot3
```

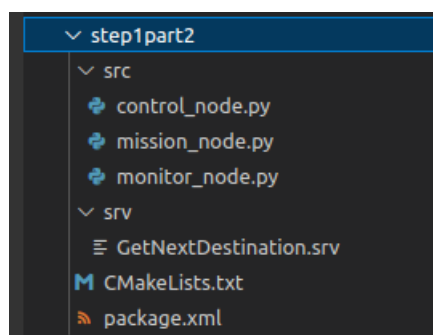
۶- توی فایل فوق لازمه که ورودی ها و خروجی های سرویس را بگوییم. بین ورودی ها و خروجی ها با --- جدا میشود. اگر ورودی نداره خط اول همون --- میشود.

```
package.xml  CMakeLists.txt  GetNextDestination.srv X
src > step1part2 > srv > GetNextDestination.srv
1 float64 current_x
2 float64 current_y
3 ---
4 float64 next_x
5 float64 next_y
```

۷- حال به پوشه hw3_ws در ترمینال برگشته و catkin_make میکنیم.

```
mahdi@mahdi:~/Desktop/Robotics/project3/hw3_ws/src$ cd ..
mahdi@mahdi:~/Desktop/Robotics/project3/hw3_ws$ catkin_make
Base path: /home/mahdi/Desktop/Robotics/project3/hw3_ws
Source space: /home/mahdi/Desktop/Robotics/project3/hw3_ws/src
Build space: /home/mahdi/Desktop/Robotics/project3/hw3_ws/build
Devel space: /home/mahdi/Desktop/Robotics/project3/hw3_ws/devel
Install space: /home/mahdi/Desktop/Robotics/project3/hw3_ws/install
####
### Running command: "cmake /home/mahdi/Desktop/Robotics/project3/hw3_ws/src -DCATKIN_DEVEL_PREFIX=/home/mahdi/Desktop/Robotics/project3/hw3_ws/devel -DCMAKE_INSTALL_PREFIX=/home/mahdi/Desktop/Robotics/project3/hw3_ws/install -G Unix Makefiles" in "/home/mahdi/Desktop/Robotics/project3/hw3_ws/build"
####
-- Using CATKIN_DEVEL_PREFIX: /home/mahdi/Desktop/Robotics/project3/hw3_ws/devel
-- Using CMAKE_PREFIX_PATH: /opt/ros/noetic
-- This workspace overlays: /opt/ros/noetic
-- Found PythonInterp: /usr/bin/python3 (found suitable version "3.8.10", minimum required is "3")
-- Using PYTHON_EXECUTABLE: /usr/bin/python3
-- Using Debian Python package layout
-- Using empy: /usr/lib/python3/dist-packages/em.py
-- Using CATKIN_ENABLE_TESTING: ON
```

حال به سراغ ساخت نودها میرویم. برای این منظور در فولدر src مربوط به step1part2 میرویم و دو فایل پایتون با نام های mission_node.py و control_node.py میسازیم. همچنین یک نود monitor_node.py نیز برای نمایش مسیر ربات در rviz میسازیم.



حال در گام بعد به سراغ نوشتن کد مربوط به هریک از نودها میرویم. ابتدا کد مربوط به mission را مینویسیم. کد آن به صورت زیر است. در آن یک تابع get_next_dest را داریم که به عنوان call_back برای سرویس ما میباشد. در این متد با توجه به اینکه میخواهیم مقدار پردازش کمینه باشد. ابتدا x را در بازه ۲۰- تا ۲۰ به طور رندوم تولید میکنیم و برای y با توجه به اینکه استیت بعدی باید حداقل ۱۰ متری

این یکی باشد پس باید در بیرون از یک دایره با شعاع ۱۰ متری آن باشد. روش دیگر این بود که دائما عدد رندوم تولید کنیم به نحوی که فاصله بیشتر از ۱۰ شود ولی خب ممکن است بدشانس باشیم و خیلی طولانی شود و میدانیم که برای هندل کردن سرویس نباید کار پردازشی زیادی داشت.

```
#!/usr/bin/python3

import rospy
from step1part2.srv import GetNextDestination, GetNextDestinationResponse
import random

class Mission():

    def __init__(self):
        # minimum distance from next point
        self.min_distance = 10
        # maximum and minimum x and y in grid
        self.max_x_y = 20
        self.min_x_y = -20

    def get_next_dest(self, current_loc):
        cur_x = current_loc.current_x
        cur_y = current_loc.current_y

        rospy.loginfo(f"Service : NEW CALL: {cur_x, cur_y}")

        next_x = random.uniform(self.min_x_y, self.max_x_y)
        # if random x is far enough so we don't need to calculate Euclidean distance and considering
        y
        if abs(next_x - cur_x) >= self.min_distance:
            next_y = random.uniform(self.min_x_y, self.max_x_y)
        else:
            y_dist = (self.min_distance**2 - (next_x - cur_x)**2)**0.5
            valid_min_y1 = cur_y + y_dist
            valid_min_y2 = cur_y - y_dist
            if valid_min_y1 >= self.max_x_y:
                next_y = random.uniform(self.min_x_y, valid_min_y2)
            elif valid_min_y2 <= self.min_x_y:
                next_y = random.uniform(valid_min_y1, self.max_x_y)
            else:
                next_y = random.choice([random.uniform(self.min_x_y, valid_min_y2),
random.uniform(valid_min_y1, self.max_x_y)])

        result = GetNextDestinationResponse()
        result.next_x = next_x
        result.next_y = next_y
        return result
```

```
def listener():

    rospy.init_node('mission', anonymous=True)
    mi = Mission()
    s = rospy.Service('get_destination', GetNextDestination, mi.get_next_dest)

    rospy.spin()

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

کد بعدی مربوط به کنترلر میباشد. کد کنترلر نیز تا حد زیادی شبیه به بخش قبل میباشد. در این قسمت، در تابع init خود نود را initialize کرده و سپس این نود را به عنوان client برای سرویس معرفی میکنیم و در گام بعد این نود باید سرعت خطی و زاویه ای را برای ربات در قالب twist پابلیش کند. همچنین متغیرهایی برای نگهداری موقعیت فعلی و همچنین هدف فعلی تعریف میشوند و همچنین متغیرهایی برای ضرایب gainهای PID مربوط به هریک از کنترلرهای سرعت خطی و زاویه ای تعریف میشوند. دو تابع get_pose و get_heading به ترتیب مکان فعلی و همچنین زاویه هدینگ ربات را مشخص میکنند و برمیگردانند. تابع distance_from_goal در حقیقت فاصله فعلی ربات را تا نقطه هدف تعیین شده به کمک فرمول فاصله اقلیدسی به دست می آورد و همچنین تابع angle_from_goal ابتدا زاویه ای که ربات باید نسبت به نقطه هدف داشته باشد را میابد و سپس مقدار زاویه هدینگ را از آن کم میکند و حاصل را برمیگرداند که در حقیقت زاویه ای است که ربات باید بچرخد تا در جهت درست قرار گیرد. در تابع control لازم است تا در ۴ مرتبه مقصد جدید تعیین کنیم که شرط تعیین مقصد جدید آن است که فاصله و زاویه از هدف از یک thresholdی کمتر باشد. همچنین باتوجه به gainهای تعیین شده و خطای فاصله و زاویه ای که داریم در هر time step مقادیر P و I و D تعیین شده و از جمع این مقادیر، مقدار سرعت خطی و زاویه ای تعیین میشود. برای هر مسیر جدید نیز دقت شود که پارامترهای مربوط به ارورهای قبلی و ارورهای جمع شوند باید reset شوند و در مسیر جدید از ۰ شروع شوند.

```
#!/usr/bin/python3

import rospy
import tf
import math
from nav_msgs.msg import Odometry
from geometry_msgs.msg import Twist
import matplotlib.pyplot as plt
from step1part2.srv import GetNextDestination, GetNextDestinationRequest
```



```

class PIDController():
    def __init__(self) -> None:

        rospy.init_node("controller" , anonymous=False) # initialize node
        rospy.wait_for_service('get_destination')        # wait for response from service
        self.calc_client = rospy.ServiceProxy('get_destination', GetNextDestination)
        self.cmd_publisher = rospy.Publisher('/cmd_vel' , Twist , queue_size=10) # this node also
is a publisher
        rospy.on_shutdown(self.on_shutdown)

        self.current_x = 0
        self.current_y = 0

        # linear velocity PID gains
        self.k_p_l = 0.05
        self.k_i_l = 0.0005
        self.k_d_l = 0.1

        # angular velocity PID gains
        self.k_p_a = 0.2
        self.k_i_a = 0.005
        self.k_d_a = 0.4

        # threshold for getting next point
        self.dist_threshold = 0.035
        self.angle_threshold = 0.7

        # define goal variables
        self.x_goal = 0
        self.y_goal = 0

        # arrays contains all errors in simulation time used for plotting
        self.dist_errs = []
        self.angle_errs = []

        # time step of get feedback
        self.dt = 0.005
        # the desired value
        self.D = 0

        rate = 1/self.dt
        self.r = rospy.Rate(rate)

    def get_heading(self):
        ...

        get the yaw angle of robot in world.
        We call it, heading of the robot.

```

```

'''
# 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

def get_pose(self):
'''
get x and y coordinate of position of the robot
'''
# waiting for the most recent message from topic /odom
msg = rospy.wait_for_message("/odom" , Odometry)

position = msg.pose.pose.position

return position.x, position.y

def distance_from_goal(self):
'''
this function get us the current Euclidean distance from goal
'''
x_curr, y_curr = self.get_pose()
distance = math.sqrt((self.x_goal-x_curr)**2 + (self.y_goal-y_curr)**2)

return distance

def angle_from_goal(self):
'''
function below first calculate the heading angle and then the desired angle from current
pose to goal pose.
it means that robot heading must be equal to this angle for being in a
correct direction. then return the difference between heading and desired angle.
'''
# find x and y of current position and find relative x and y to goal point
x_curr, y_curr = self.get_pose()
relative_x = self.x_goal - x_curr
relative_y = self.y_goal - y_curr
# get heading of robot in radian
heading = self.get_heading()
# now we should find the desired angle
# desired angle tell us the angle of goal point relative to current point

```

```

desired_angle = 0
if (relative_x == 0 and relative_y ==0):
    # this state (x=0 , y =0) is undefined so we handle it separately
    desired_angle = heading
else:
    desired_angle = math.atan2(relative_y, relative_x)
# the angle express howmuch we should rotate to reach the desired angle
angle = heading - desired_angle
# but we design controller and if the angle is bigger than 180 or less than -180
# the robot must rotate alot so we should find its complementary to 360 degrees
if angle < math.radians(-180):
    angle = math.radians(360)-abs(angle)
elif angle > math.radians(180):
    angle = angle-math.radians(360)
return angle

def get_next_goal(self):
    """
    This method first find current x and y and then
    use them as inputs for send request to service and
    get next goal. The goal must be in range (-20,20) for
    x and y and also has minimum distance of 10 at least.
    """
    self.current_x, self.current_y = self.get_pose()
    req = GetNextDestinationRequest()
    req.current_x = self.current_x
    req.current_y = self.current_y
    rospy.loginfo(f"Client : current pose: {req.current_x, req.current_y}")
    resp = self.calc_client(req)
    self.x_goal, self.y_goal = resp.next_x, resp.next_y
    rospy.loginfo(f"Client : Goal pose : {resp.next_x, resp.next_y}")

def control(self):
    """
    this function is the main function of this code. we calculate the angular and
    linear distance error and try to calculate P, I, D terms. the summation of these
    terms define our linear and angular velocity.
    also we define two thresholds and if the robot close to goal point and the
    goal distance and angle distance are less than that thresholds then we should
    find new goal and move toward it. This process is happened 4 times and then
    the program is terminated.
    """
    distance = self.distance_from_goal()
    sum_i_dist = 0
    prev_error_dist = 0
    err_dist = distance - self.D

```

```

angle = self.angle_from_goal()
sum_i_angle = 0
prev_error_angle = 0
err_angle = angle - self.D

move_cmd = Twist()

counter = 0

while not rospy.is_shutdown():

    # If distance from current goal is less than threshold then define new goal
    if err_dist < self.dist_threshold and err_angle < self.angle_threshold:

        counter += 1

        if counter > 1:
            rospy.loginfo(f"Goal{counter-1} final error dist: {err_dist } final err_angle :
{err_angle}")

            rospy.loginfo(f"-----")

        # In this task we should go to a new goal for 4 times
        if counter == 5:
            break

        self.get_next_goal()
        # the previous path errors aren't important for us.
        # So initialize related parameters again
        sum_i_angle = 0
        prev_error_angle = 0
        sum_i_dist = 0
        prev_error_dist = 0

        # linear velocity
        distance = self.distance_from_goal()
        err_dist = distance - self.D
        self.dist_errs.append(err_dist)
        sum_i_dist += err_dist * self.dt

        P_l = self.k_p_l * err_dist
        I_l = self.k_i_l * sum_i_dist
        D_l = self.k_d_l * (err_dist - prev_error_dist)

        move_cmd.linear.x = P_l + I_l + D_l
        prev_error_dist = err_dist

        #rospy.loginfo(f"linear velocity")

```

```

        #rospy.loginfo(f"P_1 : {P_1} I_1 : {I_1} D_1 : {D_1}")

        # angular velocity
        angle = self.angle_from_goal()
        err_angle = angle - self.D
        self.angle_errs.append(err_angle)
        sum_i_angle += err_angle * self.dt

        P_a = self.k_p_a * err_angle
        I_a = self.k_i_a * sum_i_angle
        D_a = self.k_d_a * (err_angle - prev_error_angle)

        move_cmd.angular.z = -(P_a + I_a + D_a)
        self.cmd_publisher.publish(move_cmd)
        prev_error_angle = err_angle

        #rospy.loginfo(f"angular velocity")
        #rospy.loginfo(f"P_a : {P_a} I_a : {I_a} D_a : {D_a}")

        #rospy.loginfo(f"error_angle : {err_angle} error_dist: {err_dist} angular speed :
{move_cmd.angular.z} linear speed : {move_cmd.linear.x}")
        #rospy.loginfo(f"error_angle : {err_angle} error_dist: {err_dist}")
        self.r.sleep()

def on_shutdown(self):
    """
    this method plot error of linear and angular velocity separately.
    """
    rospy.loginfo("Stopping the robot...")
    self.cmd_publisher.publish(Twist())
    # linear
    plt.plot(list(range(len(self.dist_errs))), self.dist_errs, label='dist_errs')
    plt.axhline(y=0,color='R')
    plt.draw()
    plt.legend(loc="upper left", frameon=False)
    plt.savefig(f"errors_dist_{self.k_p_l}_{self.k_d_l}_{self.k_i_l}.png")
    plt.show()

    # angular
    plt.plot(list(range(len(self.angle_errs))), self.angle_errs, label='angle_errs')
    plt.axhline(y=0,color='R')
    plt.draw()
    plt.legend(loc="upper left", frameon=False)
    plt.savefig(f"errors_angle_{self.k_p_a}_{self.k_d_a}_{self.k_i_a}.png")
    plt.show()
    rospy.sleep(1)

if __name__ == '__main__':
    try:

```

```

pidc = PIDController()
pidc.control()
except rospy.ROSInterruptException:
    rospy.loginfo("Navigation terminated.")

```

در monitor کد پایتون مربوط به رسم مسیری که ربات آن را طی میکند میباشد که چون کد آن تغییری نکرده است و در بخش ۱ توضیح دادیم از توضیح مجدد آن خودداری میکنم.

در مرحله بعد باید تمامی کدهای پایتون را executable کنیم. برای این کار لازم است در ترمینال در پکیج get_destinations کد زیر را اجرا کنیم:

- `chmod +x src/*.py`

```

mahdi@mahdi:~/Desktop/Robotics/project3/hw3_ws$ cd src/step1part2/
mahdi@mahdi:~/Desktop/Robotics/project3/hw3_ws/src/step1part2$ chmod +x src/*.py
mahdi@mahdi:~/Desktop/Robotics/project3/hw3_ws/src/step1part2$ cd src/
mahdi@mahdi:~/Desktop/Robotics/project3/hw3_ws/src/step1part2/src$ ls
control_node.py mission_node.py monitor_node.py

```

سپس به سراغ نوشتن لانچ فایل میرویم. در همین پکیج step1part2 لازم است تا یک فولدر launch ایجاد کنیم و داخل آن یک control.launch ایجاد کنیم. لانچ فایل به صورت زیر است. ابتدا mission_node.py را لازم است بالا بیاوریم. میدانیم پکیج آن step1part2 است و همچنین خروجی‌های آن را میخواهیم در ترمینال چاپ کند پس screen مینویسیم.

سپس یک لانچ فایل دیگر را include میکنیم و میگوییم در آن آرگومان‌های ورودی مثل مکان اولیه ربات و زاویه اولیه چه باشند. اصلاً در کدام نقشه (که اینجا empty_world بود) را بالا بیاورد. و ربات و gazebo را آماده کند. همچنین لانچ فایل مربوط به rviz را هم صدا میزنیم.

سپس mission_node.py سپس control_node.py را لازم است بالا بیاوریم. در نود کنترلر پارامتر linear_speed که میتواند توسط کاربر وارد شود را مینویسیم. در نهایت لازم است نود monitor که برای رسم مسیر لازم بود بالا بیاوریم.

در زیر میتوانید لانچ فایل را مشاهده کنید:

```
<launch>

  <node pkg="step1part2" type="mission_node.py" name="mission" output="screen"></node>

  <include file="$(find turtlebot3_gazebo)/launch/my_empty_world.launch">
    <arg name="x_pos" value="0.0"/>
    <arg name="y_pos" value="0.0"/>
    <arg name="z_pos" value="0.0"/>
    <arg name="yaw" value="0.0"/>
  </include>

  <include file="$(find turtlebot3_gazebo)/launch/turtlebot3_gazebo_rviz.launch"/>

  <node pkg="step1part2" type="control_node.py" name="controller" output="screen"></node>

  <node pkg="step1part2" type="monitor_node.py" name="monitor"></node>

</launch>
```

سپس در آخر لازم است تا به دایرکتوری ورک اسپیس برویم و catkin_make را صدا بزنیم. سپس برای استفاده لازم است تا ابتدا سورس کنیم و سپس ربات را اکسپورت کنیم و در نهایت roslaunch را صدا بزنیم:

- . devel/setup.bash
- export TURTLEBOT3_MODEL=waffle
- roslaunch step1part2 control.launch

```
mahdi@mahdi:~/Desktop/Robotics/project3/hw3_ws$ . devel/setup.bash
mahdi@mahdi:~/Desktop/Robotics/project3/hw3_ws$ export TURTLEBOT3_MODEL=waffle
mahdi@mahdi:~/Desktop/Robotics/project3/hw3_ws$ roslaunch step1part2 control.launch
... logging to /home/mahdi/.ros/log/598d2556-017f-11ee-b6bc-c53a202ffaaf/roslaunch-mahdi-391242.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.
```

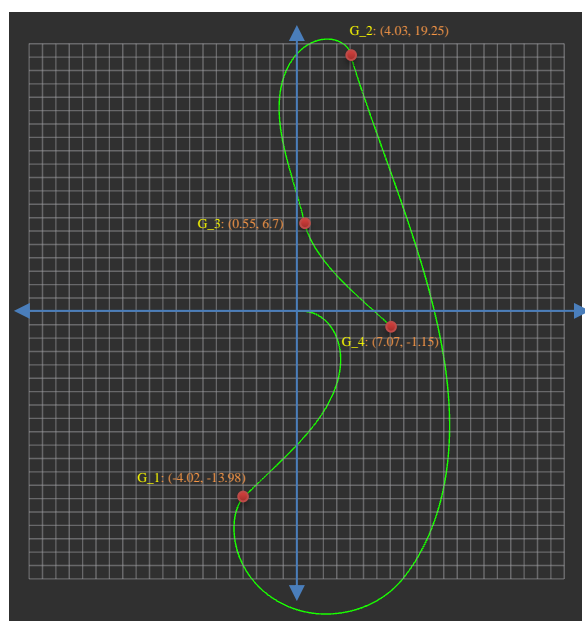
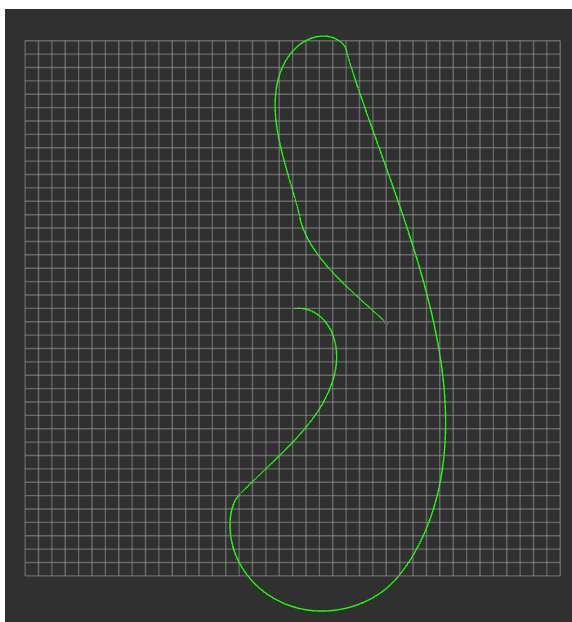
گام اول-بخش دوم (اجرا و نتایج)

در این قسمت gain های PID مربوط به بخش ۱ را مبنای کار خود قرار دادیم ولی خب در بخش قبل چون ربات با زاویه $yaw=0$ قرار داده شده بود و روبروی goal point بود کمی حالت خاص محسوب میشد و اینجا ممکن است زاویه دوران و همچنین فاصله برای رسیدن به نقطه هدف بیشتر باشد لازم است تا کمی gain ها را تغییر دهیم. برای آنکه دقت بیشتر شود و مسیرهای طی شده تا رسیدن به مقصد خیلی عجیب و پر از پیچ نباشد کمی مقدار gain های k_{p_l} و k_{p_a} را کم کردیم تا ربات با سرعت خطی و دورانی کمتری پیش برود و تا خطا کمتر شود. به همین نسبت مقادیر مربوط به k_i و K_d برای هردو سرعت خطی و زاویه‌ای نیز تغییر میکند و کمتر میشود. در نهایت مقادیر gain های زیر به نظر خوب میباشند. لازم به ذکر است که این مقادیر با سعی و خطا به دست آمدند.

```
# linear velocity PID gains
self.k_p_l = 0.05
self.k_i_l = 0.0005
self.k_d_l = 0.1

# angular velocity PID gains
self.k_p_a = 0.2
self.k_i_a = 0.005
self.k_d_a = 0.4
```

حال اگر لانچ فایل مربوطه را ران کنیم خروجی زیر را میگیریم. (فیلم ضبط شده نیز مربوط به همین اجرا میباشد.) در تصویر راست نقاط goal به همراه مختصات مشخص شده اند. همانطور که میبینید ربات با دقت خوبی به هدف ها رسیده است.

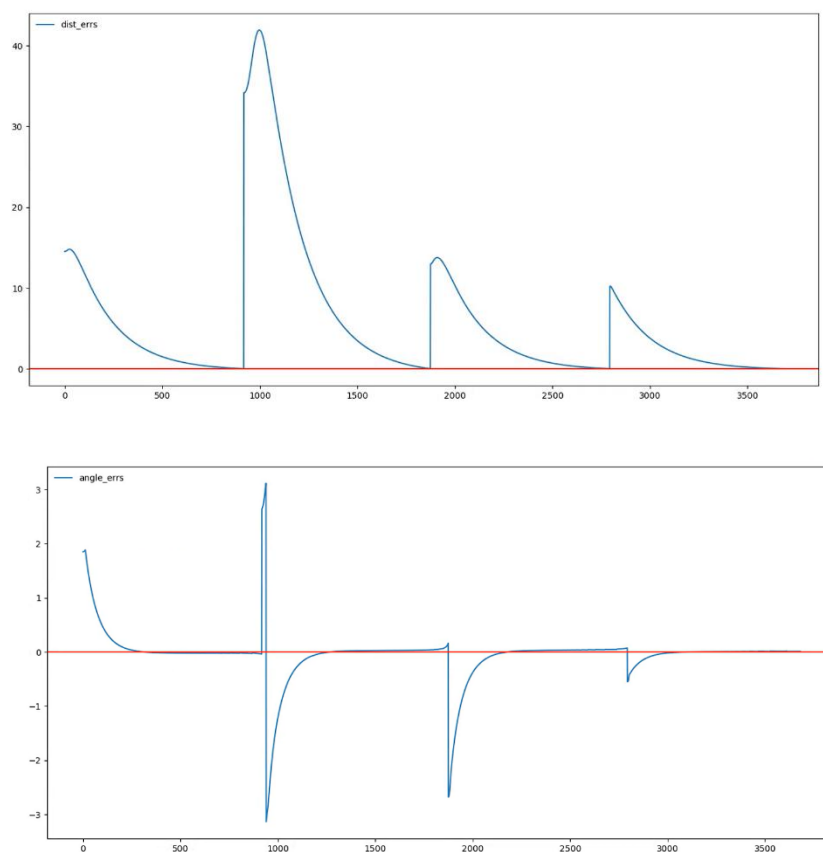


در تصویر زیر میتوانید نقاط هدفی که توسط سرویس به ربات داده شده است، به همراه خطای فاصله و زاویه (البته توجه شود که در این بخش اینکه با چه زاویه‌ای به مقصد برسد مهم نبوده است) در نقطه‌ای که ربات به threshold مد نظر رسیده است را مشاهده کنید. این فرآیند ۴ بار تکرار شده است.

```
[INFO] [1685735701.290745764, 0.126000000]: DiffDrive(ns = //): Advertise odom on odom
[INFO] [1685735701.513258, 0.304000]: Client : current pose: (-5.097724476874239e-06, 2.943032844672686e-07)
[INFO] [1685735701.524691, 0.313000]: Service : NEW CALL: (-5.097724476874239e-06, 2.943032844672686e-07)
[INFO] [1685735701.526432, 0.314000]: Client : Goal pose : (-4.023396031003031, -13.980627768501638)
[spawn_urdf-5] process has finished cleanly
log file: /home/nahti/.ros/log/598d2556-01ff-11ee-b6bc-c53a202ffaaf/spawn_urdf-5*.log
context mismatch in svga_surface_destroy
context mismatch in svga_surface_destroy

libcurl: (35) OpenSSL SSL_connect: Connection reset by peer in connection to fuel.gazebosim.org:443
[INFO] [1685735796.807028, 92.207000]: Goal1 final error dist: 0.03462039025669011 final err_angle : -0.03793418169413476
[INFO] [1685735796.808284, 92.208000]: -----
[INFO] [1685735796.835791, 92.235000]: Client : current pose: (-4.001916684130687, -13.954870507085166)
[INFO] [1685735796.843425, 92.243000]: Service : NEW CALL: (-4.001916684130687, -13.954870507085166)
[INFO] [1685735796.845348, 92.245000]: Client : Goal pose : (4.030612654803253, 19.256586608177304)
[INFO] [1685735896.280243, 187.930000]: Goal2 final error dist: 0.03407954128933001 final err_angle : 0.16350023735544372
[INFO] [1685735896.282923, 187.931000]: -----
[INFO] [1685735896.322746, 187.972000]: Client : current pose: (4.031323561278477, 19.22569500252348)
[INFO] [1685735896.329477, 187.976000]: Service : NEW CALL: (4.031323561278477, 19.22569500252348)
[INFO] [1685735896.331950, 187.981000]: Client : Goal pose : (0.5501752314155155, 6.707618854245062)
[INFO] [1685735992.829699, 279.864000]: Goal3 final error dist: 0.03394288649039172 final err_angle : 0.07610788940225932
[INFO] [1685735992.832057, 279.865000]: -----
[INFO] [1685735992.862878, 279.897000]: Client : current pose: (0.548057354711723, 6.740419107861885)
[INFO] [1685735992.894881, 279.923000]: Service : NEW CALL: (0.548057354711723, 6.740419107861885)
[INFO] [1685735992.898062, 279.930000]: Client : Goal pose : (7.077272023887325, -1.1529946327823666)
[INFO] [1685736086.435220, 368.762000]: Goal4 final error dist: 0.03459286809312011 final err_angle : 0.01028737294791171
[INFO] [1685736086.437445, 368.764000]: -----
[INFO] [1685736086.439687, 368.767000]: Stopping the robot...
```

در نهایت نیز نمودارهای خطای فاصله‌ای و زاویه‌ای به صورت زیر میباشند. با رسیدن به مقصد خطای زاویه و فاصله هردو به سمت ۰ میروند.



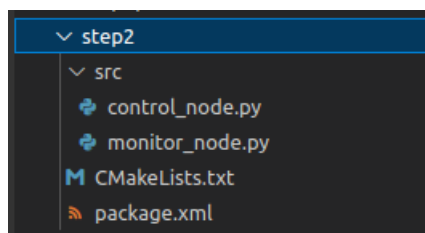
گام دوم (توضیحات کد و آماده سازی)

در این قسمت نیز ابتدا یک پکیج با نام step2 ایجاد میکنیم و dependency هایی که ممکن است به کار بیایند را هم اضافه کنیم.

- `catkin_create_pkg step2 rospy std_msgs nav_msgs`

```
maahdi@maahdi:~/Desktop/Robotics/project3/hw3_ws$ cd src/
maahdi@maahdi:~/Desktop/Robotics/project3/hw3_ws/src$ catkin_create_pkg step2 rospy std_msgs nav_msgs
Created file step2/package.xml
Created file step2/CMakeLists.txt
Created folder step2/src
Successfully created files in /home/maahdi/Desktop/Robotics/project3/hw3_ws/src/step2. Please adjust the values in package.xml.
maahdi@maahdi:~/Desktop/Robotics/project3/hw3_ws/src$
```

حال به سراغ ساخت نودها میرویم. برای این منظور در فولدر src مربوط به step2 میرویم و دوفایل پایتون با نام های `control_node.py` و `monitor_node.py` میسازیم. نود `monitor_node.py` نیز برای نمایش مسیر ربات در rviz به کار گرفته خواهد شد.



حال در گام بعد به سراغ نوشتن کد مربوط به هریک از نودها میرویم. کد مربوط به نود `monitor_node` مانند قبل میباشد و توضیح آن در قسمت های قبل داده شد. در این جا فقط به کد نود `control_node` میپردازیم. اکثر توضیحات کنترلر نیز مانند قبل میباشد. تغییراتی که این کد دارد به این قرار است: در اینجا یک متغیر `shape` داریم که کاربر به عنوان ورودی در ترمینال وارد میکند و مقدار دیفالت آن `rectangle` میباشد. بر اساس اینکه چه شکلی انتخاب شود توابع مرتبط با آن شکل یک سری `way point` روی آن شکل با ابعاد مشخص ایجاد میکنند و `x` و `y` نقاط را در قالب یک لیست برمیگردانند و در متغیر `points` ذخیره میشود. کلیات کار مانند قبل است. در اینجا کمی تغییرات در مقادیر `gain` های سرعت زاویه ای و سرعت خطی داشتیم که در قسمت اجرا و نتایج علت آن ها توضیح داده شده است. در تابع `control` کلیات مانند قبل است با این تفاوت که در اینجا ابتدا ربات باید نزدیک ترین نقطه روی شکل را پیدا کند و به سمت آن برود که به کمک تابع `find_nearest_point` انجام میشود و زمانی که از یک حد `threshold` ای به شکل نزدیک شد دیگر باید نقاط روی شکل را دنبال کند. دقت شود که تابع `find_nearest_point` را باید دائما تا زمانی که به آن `threshold` برسیم صدا بزنیم چراکه ربات ما وقتی شروع به حرکت میکند لزوما روی یک خط مستقیم به سمت هدف نمیرود و بنابراین ممکن است نزدیکترین

نقطه دائما تغییر کند. همچنین برای دنبال کردن نقاط روی شکل لزومی به دنبال کردن تک تک نقاط نیست و میتوان ۲ تا یکی نقاط را دنبال کرد. همچنین برای آنکه بتواند دوباره از نقاطی که یک بار رد شده باز هم رد شود ما برای iterate کردن روی لیست از `index%len(points)` استفاده کردیم و مقدار `index` نیز به مرور زیاد میشود. فقط در حالتی که شکل ما `logarithmic_spiral` میباشد در صورت سوال ذکر نشده که مجدد به مسیرش ادامه دهد (که منطقا یک شکل بسته هم نیست که این کار را انجام دهد) پس وقتی به انتهای لیست نقاط رسید متوقف میشود.

```
#!/usr/bin/python3

import rospy
import tf
import math
from nav_msgs.msg import Odometry
from geometry_msgs.msg import Twist
import matplotlib.pyplot as plt
import numpy as np

class PIDController():

    def __init__(self) -> None:

        rospy.init_node("controller" , anonymous=False) # initialize node
        self.cmd_publisher = rospy.Publisher('/cmd_vel' , Twist , queue_size=10) # this node also is a publisher
        rospy.on_shutdown(self.on_shutdown)

        # linear velocity PID gains
        self.k_p_l = 0.1
        self.k_i_l = 0.001
        self.k_d_l = 0.02
        # angular velocity PID gains
        self.k_p_a = 0.3
        self.k_i_a = 0.003
        self.k_d_a = 0.1

        # its not necessary to check all points if we have
        # alot of sample on path.
        self.index_increment = 2

        # the points list on a desired shape
        self.shape = rospy.get_param("/controller/shape")
        if self.shape == "rectangle":
            self.points = self.make_rectangle()
        elif self.shape == "star":
            self.points = self.make_star()
```

```

elif self.shape == "logarithmic_spiral":
    self.points = self.make_logarithmic_spiral()
    self.index_increment = 1
else:
    rospy.loginfo(f"please enter a valid shape")

# threshold for getting next point
self.dist_threshold = 0.25
self.angle_threshold = 1.2

# define goal variables
self.x_goal = 0
self.y_goal = 0

# arrays contains all errors in simulation time used for plotting
self.dist_errs = []
self.angle_errs = []

self.index = 0

# time step of get feedback
self.dt = 0.005

# the desired value
self.D = 0

rate = 1/self.dt
self.r = rospy.Rate(rate)

def get_heading(self):
    """
    get the yaw angle of robot in world.
    We call it, heading of the robot.
    """
    # 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

def get_pose(self):
    """
    get x and y coordinate of position of the robot

```

```

'''
# waiting for the most recent message from topic /odom
msg = rospy.wait_for_message("/odom" , Odometry)

position = msg.pose.pose.position

return position.x, position.y

def get_distance(self, point_target, point_curr):
'''
this function calculate Euclidean distance between
a given current and target point.
'''
x1, y1 = point_curr
x2, y2 = point_target

return math.sqrt((x2 - x1)**2 + (y2 - y1)**2)

def distance_from_goal(self):
'''
this function get us the current Euclidean distance from goal
'''
x_curr, y_curr = self.get_pose()
distance = math.sqrt((self.x_goal-x_curr)**2 + (self.y_goal-y_curr)**2)

return distance

def angle_from_goal(self):
'''
function below first calculate the heading angle and then the desired angle from current pose to goal
pose.

it means that robot heading must be equal to this angle for being in a
correct direction. then return the difference between heading and desired angle.
'''

# find x and y of current position and find relative x and y to goal point
x_curr, y_curr = self.get_pose()
relative_x = self.x_goal - x_curr
relative_y = self.y_goal - y_curr
# get heading of robot in radian
heading = self.get_heading()
# now we should find the desired angle
# desired angle tell us the angle of goal point relative to current point
desired_angle = 0
if (relative_x == 0 and relative_y ==0):
    # this state (x=0 , y =0) is undefined so we handle it seperately
    desired_angle = heading

```

```

else:
    desired_angle = math.atan2(relative_y, relative_x)
    # the angle express howmuch we should rotate to reach the desired angle
    angle = heading - desired_angle
    # but we design controller and if the angle is bigger than 180 or less than -180
    # the robot must rotate alot so we should find its complementary to 360 degrees
    if angle < math.radians(-180):
        angle = math.radians(360)-abs(angle)
    elif angle > math.radians(180):
        angle = angle-math.radians(360)

    return angle

def find_nearest_point(self):
    '''
    When the robot is placed in a world it should find the closest point
    of given path and move forward it. this function find the closest point.
    also we consider that, the robot when starting to move the nearest point
    maybe changed so we should update the nearest point.
    '''
    curr_point = self.get_pose()
    goal_point = self.points[0]
    my_min_dist = self.get_distance(goal_point, curr_point)
    counter = 0
    for point in self.points:
        dist = self.get_distance(point, curr_point)
        if dist < my_min_dist:
            my_min_dist = dist
            goal_point = point
            self.index = counter
        counter += 1
    self.x_goal, self.y_goal = goal_point

def make_rectangle(self):
    '''
    below code make sample points on a rectangle shape and
    if we plot them show us a rectangle. we return the list of
    this points and each point also is a list like [x,y].
    length: 6
    width: 4
    '''
    X1 = np.linspace(-3, 3 , 100)
    Y1 = np.array([2]*100)

    Y2 = np.linspace(2, -2 , 100)
    X2 = np.array([3]*100)

```

```

X3 = np.linspace(3, -3 , 100)
Y3 = np.array([-2]*100)

Y4 = np.linspace(-2, 2 , 100)
X4 = np.array([-3]*100)

X, Y = np.concatenate([X1,X2, X3 , X4]), np.concatenate([Y1,Y2,Y3,Y4])
points = [[x, y] for x, y in zip(X, Y)]
return points

def make_star(self):
    """
    below code make sample points on a star shape and
    if we plot them show us a star. we return the list of
    this points and each point also is a list like [x,y].
    """
    X1 = np.linspace(0, 3 , 100)
    Y1 = - (7/3) * X1 + 12

    X2 = np.linspace(3, 10 , 100)
    Y2 = np.array([5]*100)

    X3 = np.linspace(10, 4 , 100)
    Y3 = (5/6) * X3 - (10/3)

    X4 = np.linspace(4, 7 , 100)
    Y4 = -(3) * X4 + 12

    X5 = np.linspace(7, 0 , 100)
    Y5 = -(4/7) * X5 - 5

    X6 = np.linspace(0, -7 , 100)
    Y6 = (4/7) * X6 - 5

    X7 = np.linspace(-7, -4 , 100)
    Y7 = 3 * X7 + 12

    X8 = np.linspace(-4, -10 , 100)
    Y8 = -(5/6) * X8 - (10/3)

    X9 = np.linspace(-10, -3 , 100)
    Y9 = np.array([5]*100)

    X10 = np.linspace(-3, 0 , 100)
    Y10 = (7/3) * X10 + 12

```

```

X, Y = np.concatenate([X1, X2, X3, X4, X5, X6, X7, X8, X9, X10]), np.concatenate([Y1, Y2, Y3, Y4, Y5, Y6,
Y7, Y8, Y9, Y10])

points = [[x, y] for x, y in zip(X, Y)]

return points

def make_logarithmic_spiral(self):
    '''
    below code make sample points on a logarithmic spiral shape and
    if we plot them show us a logarithmic spiral. we return the list of
    this points and each point also is a list like [x,y].
    '''
    a = 0.17
    k = math.tan(a)
    X, Y = [], []

    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)
        X.append(dx)
        Y.append(dy)

    points = [[x, y] for x, y in zip(X, Y)]
    return points

def control(self):

    close_enough = False
    self.find_nearest_point()

    distance = self.distance_from_goal()
    sum_i_dist = 0
    prev_error_dist = 0
    err_dist = distance - self.D

    angle = self.angle_from_goal()
    sum_i_angle = 0
    prev_error_angle = 0
    err_angle = angle - self.D

    move_cmd = Twist()

    while not rospy.is_shutdown():

        # linear velocity
        distance = self.distance_from_goal()
        err_dist = distance - self.D

```



```

self.dist_errs.append(err_dist)
sum_i_dist += err_dist * self.dt

P_l = self.k_p_l * err_dist
I_l = self.k_i_l * sum_i_dist
D_l = self.k_d_l * (err_dist - prev_error_dist)

move_cmd.linear.x = P_l + I_l + D_l
prev_error_dist = err_dist

#rospy.loginfo(f"linear velocity")
#rospy.loginfo(f"P_l : {P_l} I_l : {I_l} D_l : {D_l}")

# angular velocity
angle = self.angle_from_goal()
err_angle = angle - self.D
self.angle_errs.append(err_angle)
sum_i_angle += err_angle * self.dt

P_a = self.k_p_a * err_angle
I_a = self.k_i_a * sum_i_angle
D_a = self.k_d_a * (err_angle - prev_error_angle)

move_cmd.angular.z = -(P_a + I_a + D_a)
self.cmd_publisher.publish(move_cmd)
prev_error_angle = err_angle

#rospy.loginfo(f"angular velocity")
#rospy.loginfo(f"P_a : {P_a} I_a : {I_a} D_a : {D_a}")

#rospy.loginfo(f"error_angle : {err_angle} error_dist: {err_dist} angular speed :
{move_cmd.angular.z} linear speed : {move_cmd.linear.x}")

# If distance from current goal is less than threshold then define new goal
if err_dist < self.dist_threshold and err_angle < self.angle_threshold:
    close_enough = True
    self.index += self.index_increment

    # Note that for logarithmic_spiral if arrive to last point we dont continue
    # but for star and rectangle we continue until user make interrupt
    if self.shape == "logarithmic_spiral" and self.index > len(self.points)-1:
        break
    self.x_goal, self.y_goal = self.points[self.index%len(self.points)]

elif not close_enough:
    self.find_nearest_point()

self.r.sleep()

```

```

def on_shutdown(self):
    rospy.loginfo("Stopping the robot...")
    self.cmd_publisher.publish(Twist())

    # linear
    plt.plot(list(range(len(self.dist_errs))), self.dist_errs, label='dist_errs')
    plt.axhline(y=0,color='R')
    plt.draw()
    plt.legend(loc="upper left", frameon=False)
    plt.savefig(f"errors_dist_{self.k_p_l}_{self.k_d_l}_{self.k_i_l}.png")
    plt.show()

    # angular
    plt.plot(list(range(len(self.angle_errs))), self.angle_errs, label='angle_errs')
    plt.axhline(y=0,color='R')
    plt.draw()
    plt.legend(loc="upper left", frameon=False)
    plt.savefig(f"errors_angle_{self.k_p_a}_{self.k_d_a}_{self.k_i_a}.png")
    plt.show()
    rospy.sleep(1)

if __name__ == '__main__':
    try:
        pidc = PIDController()
        pidc.control()
    except rospy.ROSInterruptException:
        rospy.loginfo("Navigation terminated.")

```

در مرحله بعد باید تمامی کدهای پایتون را executable کنیم. برای این کار لازم است در ترمینال در پکیج step2 کد زیر را اجرا کنیم:

- `chmod +x src/*.py`

```

mahdi@mahdi:~/Desktop/Robotics/project3/hw3_ws/src$ cd step2/
mahdi@mahdi:~/Desktop/Robotics/project3/hw3_ws/src/step2$ chmod +x src/*.py
mahdi@mahdi:~/Desktop/Robotics/project3/hw3_ws/src/step2$ cd src/
mahdi@mahdi:~/Desktop/Robotics/project3/hw3_ws/src/step2/src$ ls
control_node.py  monitor_node.py
mahdi@mahdi:~/Desktop/Robotics/project3/hw3_ws/src/step2/src$ █

```

سپس به سراغ نوشتن لانچ فایل میرویم. در همین پکیج step2 لازم است تا یک فولدر launch ایجاد کنیم و داخل آن یک control.launch ایجاد کنیم. لانچ فایل به صورت زیر است.

ابتدا یک لانچ فایل دیگر را include میکنیم و میگوییم در آن آرگومان های ورودی مثل مکان اولیه ربات و زاویه اولیه چه باشند. اصلا در کدام نقشه (که اینجا empty_world بود) را بالا بیاورد و ربات و gazebo را آماده کند. سپس launch file مربوط به rviz را include میکنیم تا آن را هم برای نمایش مسیر ربات بالا بیاورد.

همچنین نودهای control_node.py و monitor_node.py را با نام‌های controller و monitor بالا بیاورد. همچنین در نود controller ما نام شکل مسیر را به عنوان ورودی از کاربر میگیریم و در این لانچ فایل مشخص شده است. نود monitor که برای رسم مسیر لازم است و آن هم باید بالا بیاید.

```
<launch>

  <arg name="shape" default="rectangle"/>

  <include file="$(find turtlebot3_gazebo)/launch/my_empty_world.launch">
    <arg name="x_pos" value="0.0"/>
    <arg name="y_pos" value="0.0"/>
    <arg name="z_pos" value="0.0"/>
    <arg name="yaw" value="0.0"/>
  </include>

  <include file="$(find turtlebot3_gazebo)/launch/turtlebot3_gazebo_rviz.launch"/>

  <node pkg="step2" type="control_node.py" name="controller" output="screen">
    <param name="shape" value="$(arg shape)" />
  </node>

  <node pkg="step2" type="monitor_node.py" name="monitor"></node>

</launch>
```

سپس در آخر لازم است تا به دایرکتوری ورک اسپیس برویم و catkin_make را صدا بزنیم. سپس برای استفاده لازم است تا ابتدا سورس کنیم و سپس ربات را اکسپورت کنیم و در نهایت roslaunch را صدا بزنیم:

- . devel/setup.bash
- export TURTLEBOT3_MODEL=waffle
- roslaunch step2 control.launch shape:=rectangle

```
mahti@mahti:~/Desktop/Robotics/project3/hw3_ws/src$ cd ..
mahti@mahti:~/Desktop/Robotics/project3/hw3_ws$ . devel/setup.bash
mahti@mahti:~/Desktop/Robotics/project3/hw3_ws$ export TURTLEBOT3_MODEL=waffle
mahti@mahti:~/Desktop/Robotics/project3/hw3_ws$ roslaunch step2 control.launch shape:=rectangle
... logging to /home/mahti/.ros/log/6c4dd7d6-0084-11ee-a209-47d91e61adaa/roslaunch-mahti-5449.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

xacro: in-order processing became default in ROS Melodic. You can drop the option.
xacro: in-order processing became default in ROS Melodic. You can drop the option.
started roslaunch server http://mahti:45275/
```

گام دوم (اجرا و نتایج)

موارد زیر را در گزارش خود قرار دهید:

۱- نمای شکل تولید شده از حرکت ربات در شبیه ساز Rviz را نشان دهید.

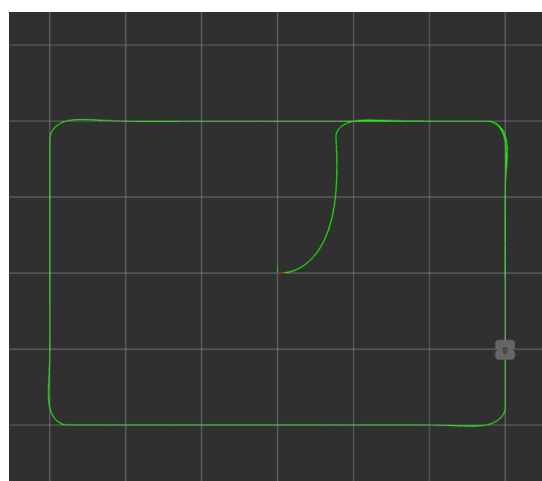
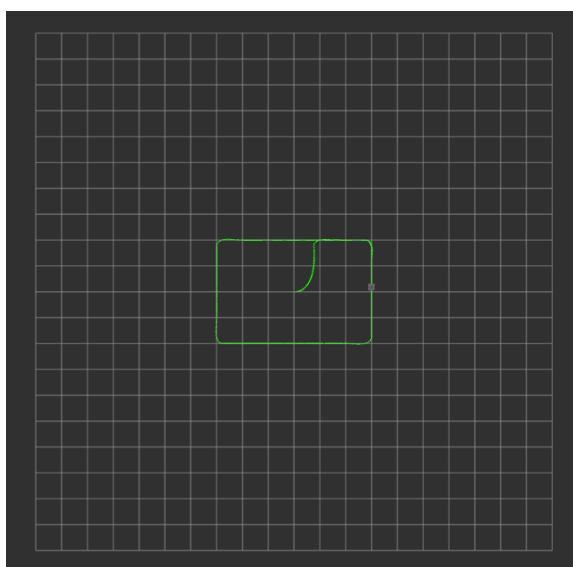
• مسیر اول: مستطیل

حالت اول (مختصات اولیه (0,0)) ابتدا مختصات اولیه را در لانچ فایل برابر با (0,0) قرار می‌دهیم و آن را اجرا می‌کنیم:

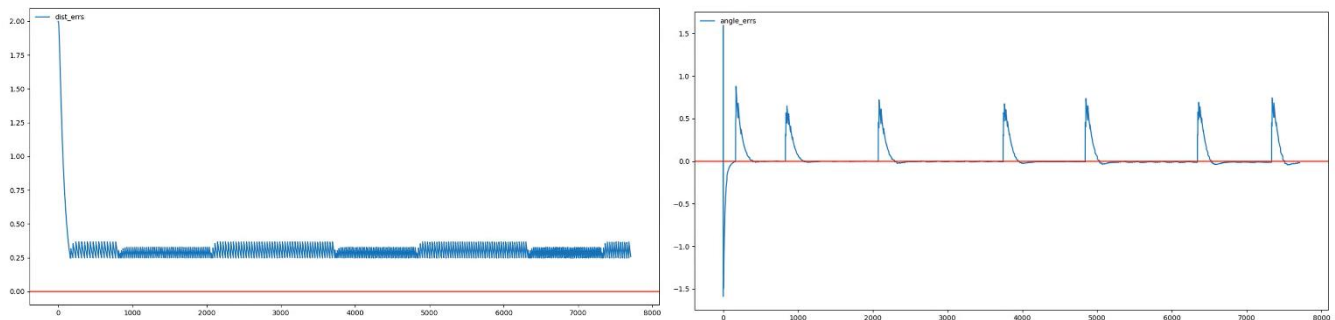
```
control_node.py  control.launch X
src > step2 > launch > control.launch
1  <launch>
2
3  <arg name="shape" default="rectangle"/>
4
5  <include file="$(find turtlebot3_gazebo)/launch/my_empty_world.launch">
6    <arg name="x_pos" value="0.0"/>
7    <arg name="y_pos" value="0.0"/>
8    <arg name="z_pos" value="0.0"/>
9    <arg name="yaw" value="0.0"/>
10 </include>
11
mahdi@mahdi:~/Desktop/Robotics/project3/hw3_ws$ . devel/setup.bash
mahdi@mahdi:~/Desktop/Robotics/project3/hw3_ws$ export TURTLEBOT3_MODEL=waffle
mahdi@mahdi:~/Desktop/Robotics/project3/hw3_ws$ roslaunch step2 control.launch shape:=rectangle
... logging to /home/mahdi/.ros/log/e78f8396-01e1-11ee-b4cc-1be5bd5aea40/roslaunch-mahdi-106434.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

xacro: in-order processing became default in ROS Melodic. You can drop the option.
xacro: in-order processing became default in ROS Melodic. You can drop the option.
started roslaunch server http://mahdi:43645/
```

همانطور که دیده میشود ربات به نزدیک ترین نقطه حرکت کرده و سپس حرکت خود را تا زمانی که برنامه متوقف نشود، ادامه می‌دهد.



نمودارهای خطا به صورت زیر میشوند. در خطای فاصله ابتدا که به سمت نزدیکترین نقطه میرویم خطا بالاس ولی بعد که روی شکل قرار میگیریم باتوجه به threshold تعریف شده خطای فاصله در اوردر threshold میباشد. threshold برای آن است که اگر مقدار فاصله تا نقطه goal که جزء مختصاتهای شکل است کمتر از آن شد برنامه به ما مختصات بعدی را بدهد و به دنبال آن برویم. در نمودار خطای زاویه خطای زیاد برای زمانیست که ربات میواهد بچرخد و مثلاً گوشه‌های مستطیل را طی کند.



حالت دوم) (مختصات اولیه (1,1)) در لانچ فایل مختصات اولیه را برابر با (1,1) میگذاریم و آن را اجرا

میکنیم:

```

control_node.py  control.launch x
src > step2 > launch > control.launch
1  <launch>
2
3  <arg name="shape" default="rectangle"/>
4
5  <include file="$(find turtlebot3_gazebo)/la
6    <arg name="x_pos" value="1.0"/>
7    <arg name="y_pos" value="1.0"/>
8    <arg name="z_pos" value="0.0"/>
9    <arg name="yaw" value="0.0"/>
10  </include>
11

```

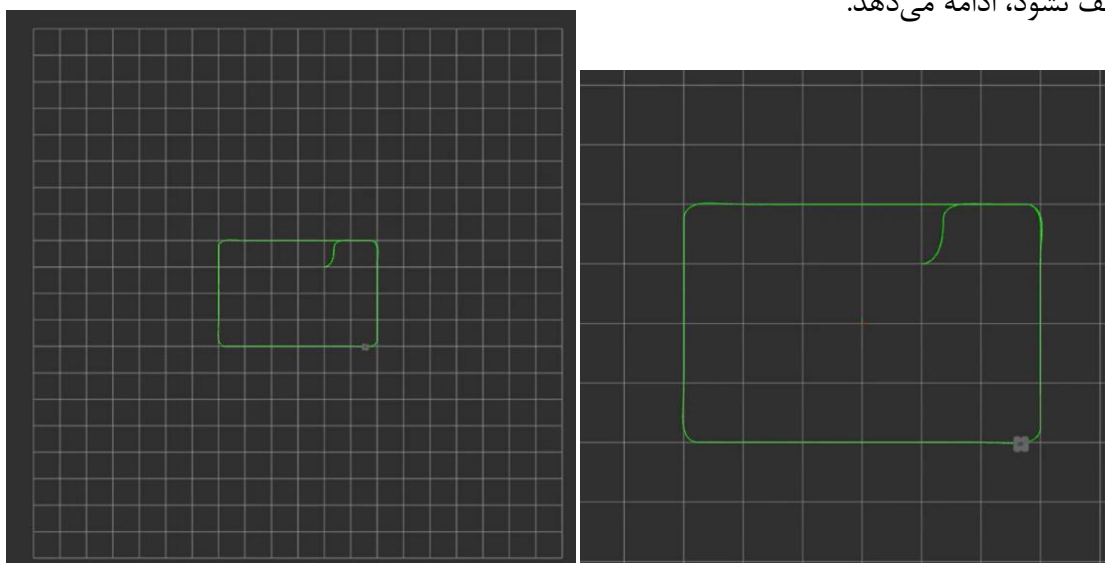
```

mahdi@mahdi:~/Desktop/Robotics/project3/hw3_ws$ . devel/setup.bash
mahdi@mahdi:~/Desktop/Robotics/project3/hw3_ws$ export TURTLEBOT3_MODEL=waffle
mahdi@mahdi:~/Desktop/Robotics/project3/hw3_ws$ roslaunch step2 control.launch shape:=rectangle
... logging to /home/mahdi/.ros/log/e78f8396-01e1-11ee-b4cc-1be5bd5aea40/roslaunch-mahdi-106434.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

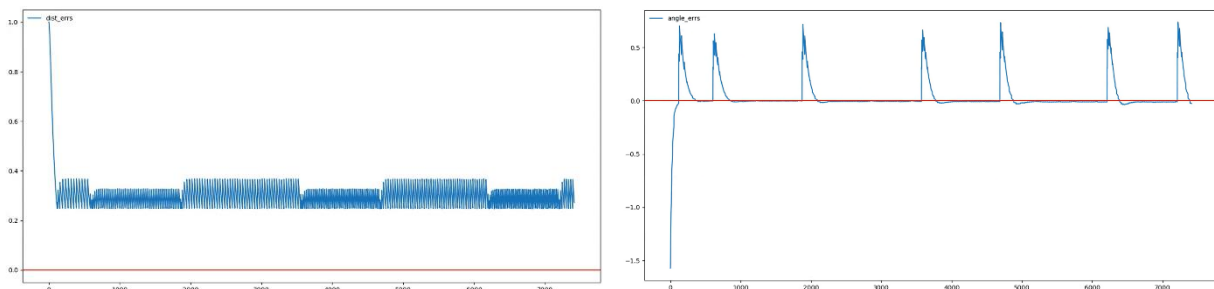
xacro: in-order processing became default in ROS Melodic. You can drop the option.
xacro: in-order processing became default in ROS Melodic. You can drop the option.
started roslaunch server http://mahdi:43645/

```

همانطور که دیده میشود ربات به نزدیک ترین نقطه حرکت کرده و سپس حرکت خود را تا زمانی که برنامه متوقف نشود، ادامه می‌دهد.



نمودارهای خطا نیز به صورت زیر خواهد بود. توضیحات نیز مانند قبل می باشد. در اینجا در خطای فاصله چون روی نقطه ۱ و ۱ بودیم خطای اولیه کمتر است.



• مسیر دوم: ستاره

در این جا یک حالت بررسی میشود و مختصات اولیه برابر با (0,0) می باشد. همچنین ربات تا زمانی که برنامه متوقف نشده به حرکت خود ادامه میدهد.

```

1 <launch>
2
3 <arg name="shape" default="rectangle"/>
4
5 <include file="$(find turtlebot3_gazebo)/launch/my_empty_world.launch">
6   <arg name="x_pos" value="0.0"/>
7   <arg name="y_pos" value="0.0"/>
8   <arg name="z_pos" value="0.0"/>
9   <arg name="yaw" value="0.0"/>
10 </include>

```

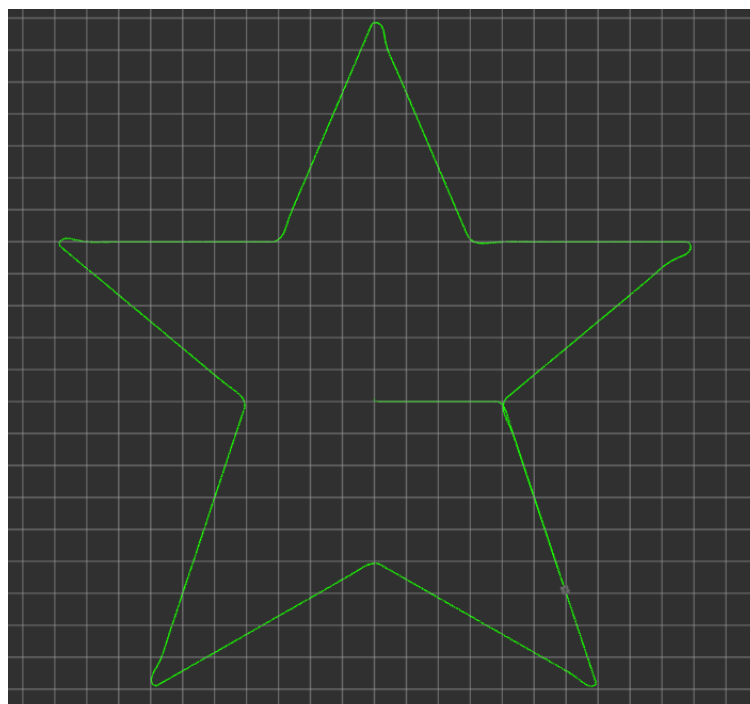
```

mahdi@mahdi:~/Desktop/Robotics/project3/hw3_ws$ . devel/setup.bash
mahdi@mahdi:~/Desktop/Robotics/project3/hw3_ws$ export TURTLEBOT3_MODEL=waffle
mahdi@mahdi:~/Desktop/Robotics/project3/hw3_ws$ roslaunch step2 control.launch shape:=star
... logging to /home/mahdi/.ros/log/e1611cca-01e9-11ee-b4cc-1be5bd5aea40/roslaunch-mahdi-214548.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is 1GB.

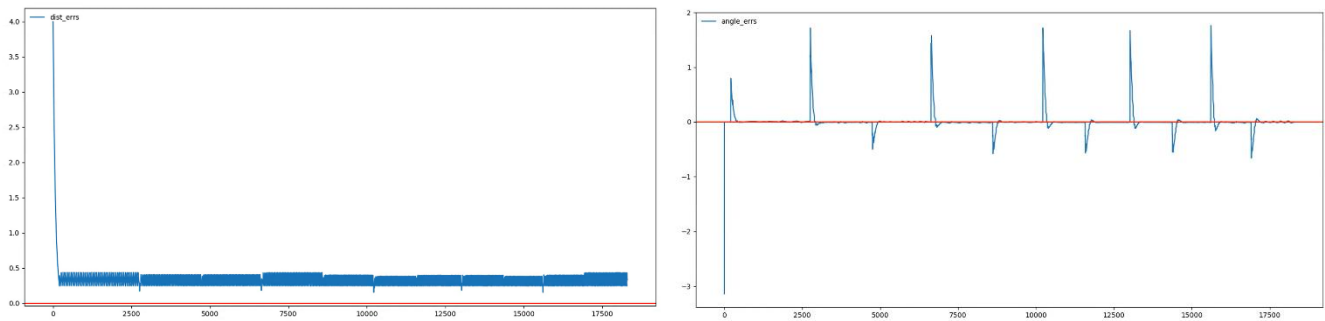
xacro: in-order processing became default in ROS Melodic. You can drop the option.
xacro: in-order processing became default in ROS Melodic. You can drop the option.

```

همانطور که دیده میشود ربات به نزدیک ترین نقطه حرکت کرده و سپس حرکت خود را تا زمانی که برنامه متوقف نشود، ادامه می دهد.



نمودارهای خطا نیز به صورت زیر خواهد بود. توضیحات نیز مانند قبل (در مسیر مستطیل توضیح داده شد) میباشد. نمودار سمت چپ نمودار خطای فاصله تا هر goal میباشد و نمودار سمت راست نمودار خطای زاویه‌ای (فاصله بین heading و زاویه desired که در قسمت کد توضیحات داده شده) میباشد.



• مسیر سوم: مارپیچ لگاریتمی

در این جا مختصات اولیه برابر با (0,0) و از در تابع تولید نقاط روی مارپیچ $a=0.17$ (در دستور کار نوشته ۱۷ که وقتی رسم کردم شکلش عجیب و خیلی بزرگ شد و به نظر غلطه) میباشد. همچنین در اینجا ذکر نشده که ربات دائماً به حرکتش ادامه دهد و در نتیجه وقتی به آخرین نقطه رسید، می ایستد.

```

1  <launch>
2
3  <arg name="shape" default="rectangle"/>
4
5  <include file="$(find turtlebot3_gazebo)/launch/my_empty_world.launch">
6    <arg name="x_pos" value="0.0"/>
7    <arg name="y_pos" value="0.0"/>
8    <arg name="z_pos" value="0.0"/>
9    <arg name="yaw" value="0.0"/>
10 </include>

```

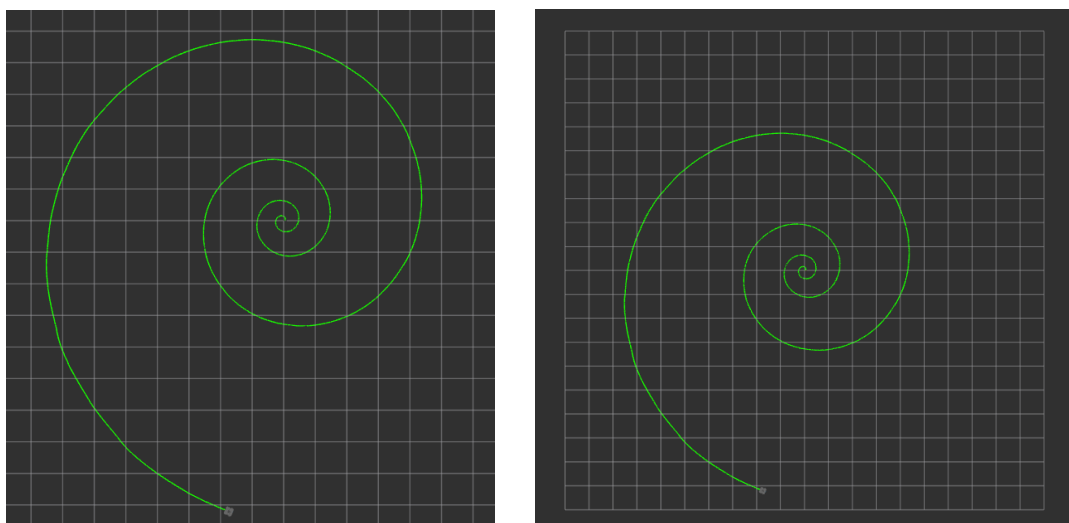
```

mahdi@mahdi:~/Desktop/Robotics/project3/hw3_ws$ . devel/setup.bash
mahdi@mahdi:~/Desktop/Robotics/project3/hw3_ws$ export TURTLEBOT3_MODEL=waffle
mahdi@mahdi:~/Desktop/Robotics/project3/hw3_ws$ roslaunch step2 control.launch shape=logarithmic_spiral
... logging to /home/mahdi/.ros/log/9f65ab78-01f4-11ee-b4cc-1be5bd5aea40/roslaunch-mahdi-335507.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

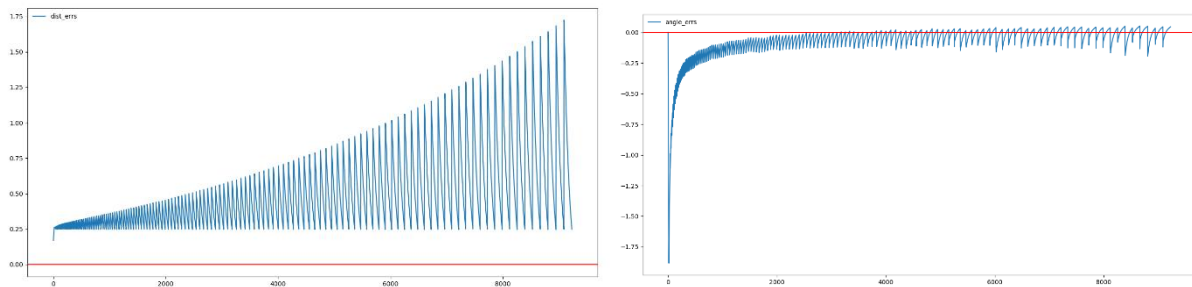
xacro: in-order processing became default in ROS Melodic. You can drop the option.
xacro: in-order processing became default in ROS Melodic. You can drop the option.
started roslaunch server http://mahdi:40653/

```

در نهایت نتیجه به صورت زیر میشود:



نمودارهای خطا نیز به صورت زیر خواهد بود. در نمودار سمت چپ که برای خطای فاصله هست، به علت اینکه به مرور فاصله نقاط روی شکل زیاد میشود خطای فاصله تا مقصد بیشتر شده ولی با نزدیک شدن به آن نقطه خطا کم میشود تا به threshold برسد و بعد نقطه بعدی روی شکل به عنوان goal جدید اعلام میشود. در نمودار سمت راست نیز خطای زاویه است که به خاطر شکل منحنی که داریم این مقدار دائماً در حال کم و زیاد شدن است و حالت نوسانی به خود گرفته است.



۲- تشریح کنید که چه ضرایبی برای P، I و D پاسخ مناسبی را ارائه میکند. (ضرایب مناسب را به صورت تجربی به دست آورید) و درباره تاثیرات افزایش و کاهش هر کدام از ضرایب بحث کنید و چند نمونه آن را تشریح کنید.

در این گام ضرایب k_p و k_i و k_d مربوط به سرعت خطی و زاویه‌ای به گونه‌ای تنظیم شدند که بتوانیم به کمک آن‌ها تمامی شکل‌های خواسته شده را تولید کنیم. ضرایب به صورت زیر میباشند:

```

23
24     # linear velocity PID gains
25     self.k_p_l = 0.1
26     self.k_i_l = 0.001
27     self.k_d_l = 0.02
28     # angular velocity PID gains
29     self.k_p_a = 0.3
30     self.k_i_a = 0.003
31     self.k_d_a = 0.1
32

```

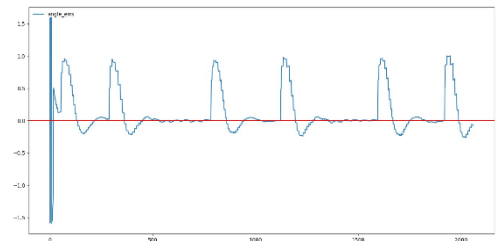
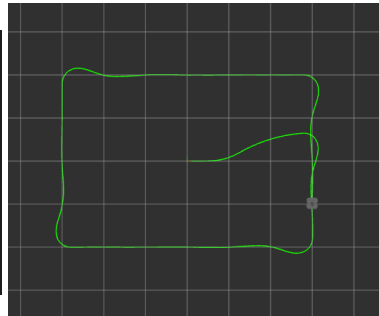
دقت شود که اگر gainها را کمی بیشتر می‌گرفتیم (به خصوص gainهای proportional) باز هم جواب قابل قبولی میداد و سرعت هم بیشتر میشد ولی برای آنکه شکل موردنظر تا حد ممکن دقیقتر شود و در گروه درس هم مطرح شد که دقت مهم هست، gainها را کم گرفتیم. همچنین می‌خواستم که یک کنترلر بنویسم که برای هر سه شکل کار کند.

• مراحل رسیدن به gainهای فوق

ابتدا مقادیر را همینطوری set کردم و خروجی زیر را دریافت کردم. به نظر میرسد که اولاً باید k_p مربوط به سرعت زاویه‌ای بیش‌تر از سرعت خطی باشد تا به ارور زاویه‌ای سریع‌تر پاسخ دهد و بتواند به نقطه

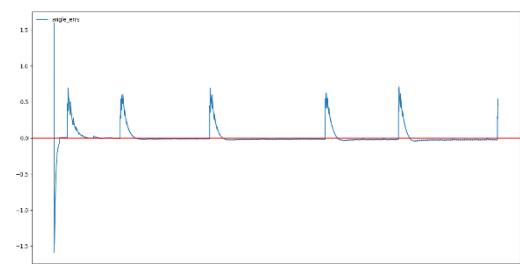
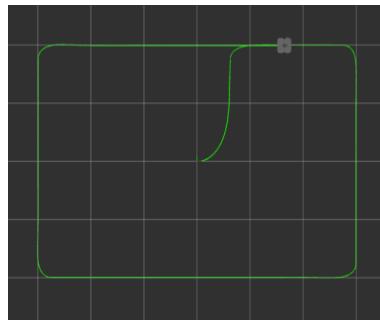
نزدیکتر که ضلع بالا و پایین است برود و در پیچ‌ها هم بهتر پیچد و خیلی منحنی نشود. همچنین نیاز است مقدار k_d ها نیز بیشتر شوند و برای زاویه کمی بیشتر، بیشتر شود ☺ در نمودار خطای زاویه هم میتوان نوسانات و overshoot را مشاهده کرد که به کمک همین gain دیفرانسیلی میتوان بهبود بخشید. در ادامه با جزئیات بیشتر بررسی میشود.

```
# linear velocity PID gains
self.k_p_l = 0.4
self.k_i_l = 0.005
self.k_d_l = 0.05
# angular velocity PID gains
self.k_p_a = 0.4
self.k_i_a = 0.003
self.k_d_a = 0.03
```



باتوجه به تجربه قسمت های قبل اگر gain های proportional کمتر شود باعث میشود ربات به خطاها با سرعت کمتری پاسخ دهد و راحت تر و دقیقتر بتوان آن را کنترل کرد. از طرفی هم طبق توضیح بالا باید k_p مربوط به سرعت زاویه ای از خطی بیشتر است. پس با این اوصاف همان ضرایب به دست آمده از گام ۱ بخش دوم را که فاصله نقاط بیشتر بود و جواب گرفتیم، معیار خود قرار دادیم و از آن‌ها استفاده کردیم و تا حد خوبی جواب می‌گرفتیم چراکه ضرایب k_p مقادیر کمی داشتند و به آرامی ربات حرکت میکرد و Overshoot چندانی نداشتیم. این ضرایب و نتایج آن در زیر آمده اند. خروجی خیلی دقیق است ولی چون gain های P خیلی کم است اجرای آن خیلی طول کشید و ربات سرعت کمی داشت.

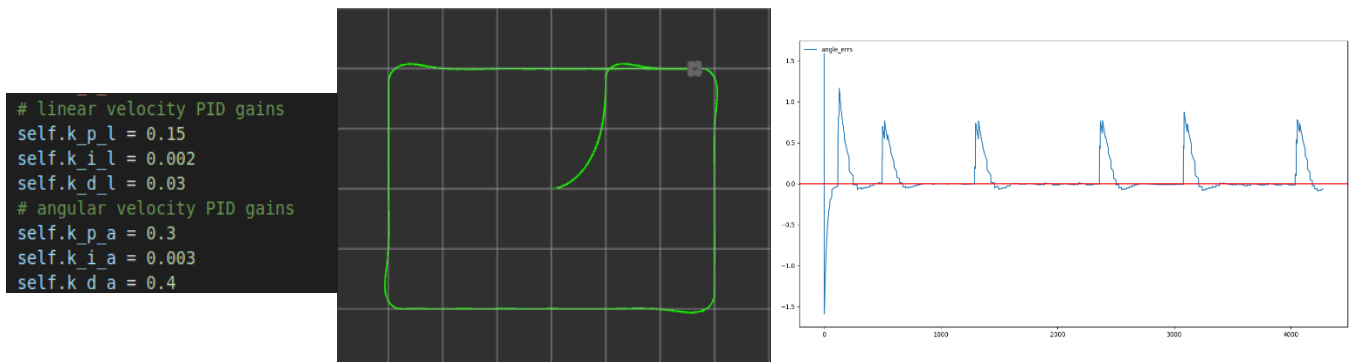
```
# linear velocity PID gains
self.k_p_l = 0.05
self.k_i_l = 0.0005
self.k_d_l = 0.1
# angular velocity PID gains
self.k_p_a = 0.2
self.k_i_a = 0.005
self.k_d_a = 0.4
```



همانطور که اشاره شد، در اینجا نقاط goal نزدیک به هم دیگر هستند و از رفتن از یکی به دیگری خیلی ارور فاصله و زاویه زیاد نمیشد. بیتترین ارور در هنگام شروع برای رفتن از مثلاً (0,0) به نزدیکترین نقطه به خصوص در شکل مستطیل بود. که با توجه به نحوه پیاده سازی و اینکه سرعت‌های خطی و زاویه‌ای زیاد نبود به خوبی کنترل شد. در بعضی مواقع در سر پیچ‌ها کمی overshoot زیاد بود و ربات خیلی فاصله می‌گرفت که به کمک gain مشتقی یعنی k_d کنترل شد و مقدارش را بیشتر کردم. همانطور که قبلاً هم

گفتم gain انتگرالی یعنی k_i ممکن است در این بخش خیلی به چشم نیایند چون steady state error نداریم و مخصوصاً یک threshold تعریف کردیم که ربات به آن برسد کافی است و به هر جهت مقدار آن‌ها طبق توضیحات اسلاید برابر با 0.01 مقدار k_p در نظر گرفته شد. با این حال این ترم به stability و پایداری ربات کمک میکند ولی خب مقدارش را کم گذاشتیم که برعکس موجب Overshoot نشود. حتی میشد مقدارش را کمتر هم بگذاریم.

پس از کمی سعی و خطا برای شکل مستطیل مقادیر gain‌ها را به صورت زیر برای سرعت خطی و زاویه‌ای گذاشتیم و نتیجه نسبتاً خوبی میداد ولی خب در گوشه‌ها به خصوص در شکل ستاره کمی به مشکل می‌خوردیم و چون ترم p زیاد میشد سرعت بیشتر بود و خطای کار بالا میرفت.



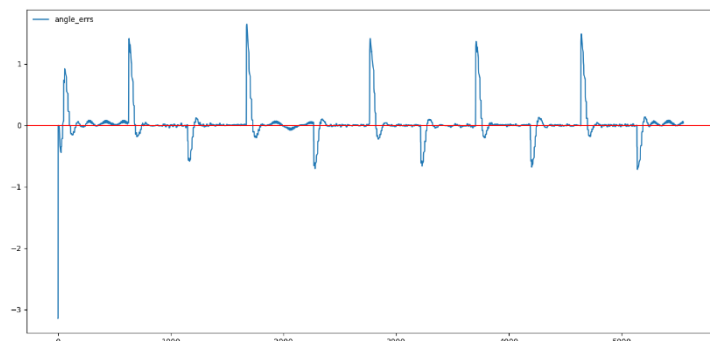
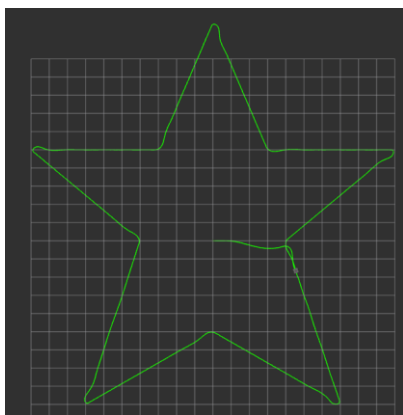
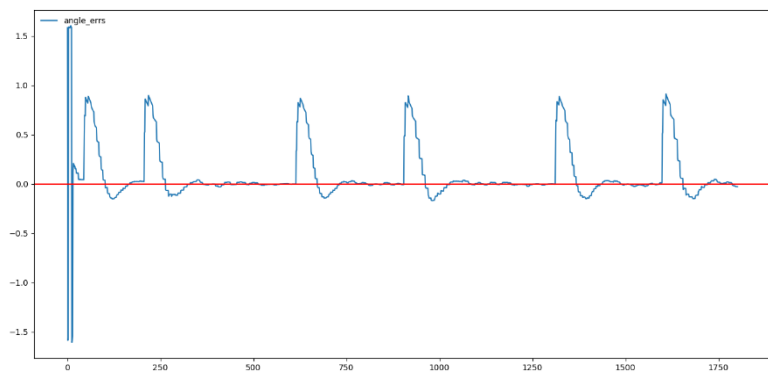
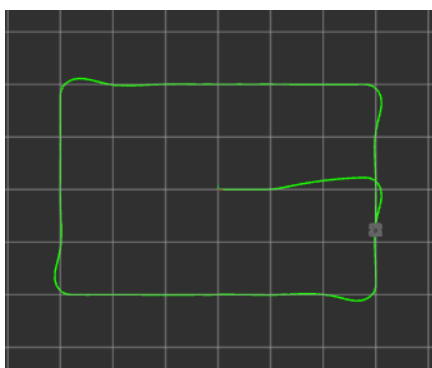
با سعی و خطای بیشتر مقدار k_{p_l} را به 0.1 رساندیم و مقدار k_{i_l} را برابر با 0.01 مقدار k_{p_a} گرفتیم و دقت شود در این مسئله خیلی ترم انتگرال گیر کمی نمیکند ولی زیاد شدنش منجر به overshoot و خطا میشود. ترم k_d ها را هم کمی کمتر کردیم چراکه تا جای ممکن باید کم باشند که نویز پذیری و نوسانات سیستم کاهش یابد.

• در ادامه توضیح کم و زیاد کردن هر ترم آمده است که منجر به رسیدن به پاسخ نهایی شد:

حال برای مثال اگر gain مربوط به k_p را زیاد کنیم خب باعث میشود که ربات سعی کند سریعتر ارور را کم کند و سریعتر ری اکشن دهد که طبیعتاً موجب Overshoot میشود. برای مثال با gain‌های زیر خروجی حاصله در شکل مستطیل و ستاره به صورت زیر میشود. در هنگام شروع چون سرعت بالاس به خوبی نمیتواند نزدیکترین نقطه را برای شروع که در ضلع‌های بالا و پایین است پیدا کند و چون سریع میرود به

جایی میرسد که ضلع چپ و راست نزدیکترین میشوند. در گوشه‌ها هم به خوبی میتوان Overshoot را دید. همچنین نمودار خطای زاویه مربوط به مستطیل به خوبی بیانگر این Overshootها و نوسانات میباشد.

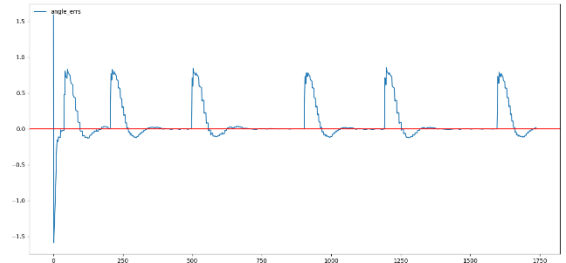
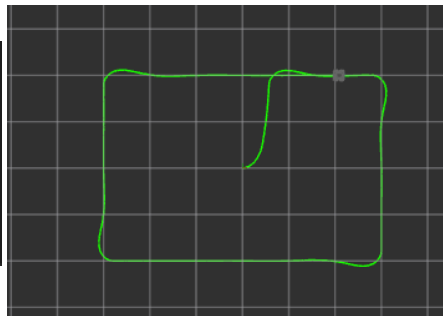
```
# linear velocity PID gains
self.k_p_l = 0.5
self.k_i_l = 0.001
self.k_d_l = 0.02
# angular velocity PID gains
self.k_p_a = 0.6
self.k_i_a = 0.003
self.k_d_a = 0.1
```



اگر مقدار K_p ها کمتر شود باعث میشود که سرعت کمتر شود و overshoot کمتر شود. همچنین دقت شود سعی میشود که k_p مربوط به سرعت زاویه‌ای بیشتر از خطی باشد تا سرعت خطی زیاد موجب نشود که حالت curve بزرگی داشته باشیم و درواقع در مواقع پیچیدن نقطه‌ای که باید بپیچیم را رد کنیم. اگر مقادیر k_d را زیاد کنیم بخشی از overshootها را میتواند جبران کند برای مثال در مستطیل به صورت زیر میشود. اگر نمودار خطا با قبلی مقایسه شود متوجه میشویم هم نوسانات کمتر شده است و هم دامنه اورشوتها کمی کم شده است و در شروع کار نیز به خوبی به سمت نقطه نزدیکتر میچرخد. به هر حال

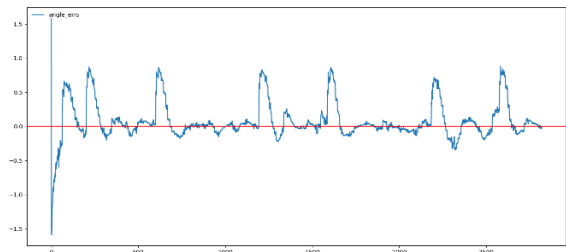
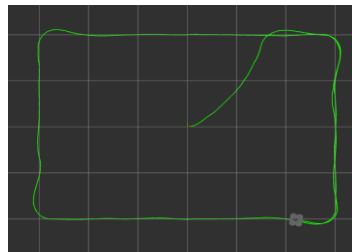
چون سرعت بالاس ضریب d هم تا حدی میتواند جلوی overshoot را بگیرد و بنابراین باید k_p را کمتر کرد.

```
# linear velocity PID gains
self.k_p_l = 0.5
self.k_i_l = 0.001
self.k_d_l = 0.05
# angular velocity PID gains
self.k_p_a = 0.6
self.k_i_a = 0.003
self.k_d_a = 2.5
```



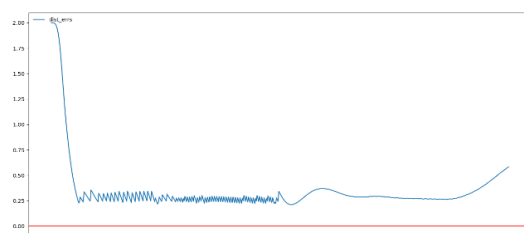
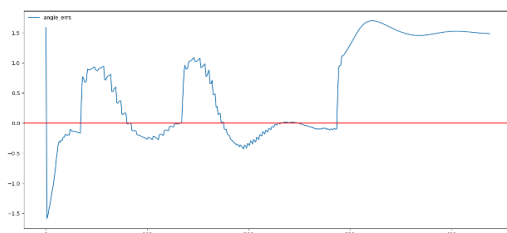
اگر مقدار k_d بیش از حد زیاد شود باعث میشود که ربات به نویز و تغییرات خیلی حساس شود و دائما نوسان کند و مسیر از smooth بودن خارج شود.

```
# linear velocity PID gains
self.k_p_l = 0.5
self.k_i_l = 0.001
self.k_d_l = 6
# angular velocity PID gains
self.k_p_a = 0.6
self.k_i_a = 0.003
self.k_d_a = 20
```



اگر مقدار k_i زیاد باشد باعث میشود که برعکس مقدار overshoot بیشتر شود و سیستم ناپایدار شود و ارور جمع شونده ای که داریم باعث به نقطه هدف نرسیم و دائما هم خطا بیشتر شود. مثلا در شکل مستطیل در گوشه انگار که ربات نقطه هدف را به نوعی رد میکند و سعی میکند که برگردد به آن برسد و دائما تلاش میکند و خطای جمع شونده هم بیشتر میشود. لذا مقدار k_i را خیلی کم میکنیم و با مقادیری که ما انتخاب کردیم از این دست مشکلات نمیگیریم.

```
# linear velocity PID gains
self.k_p_l = 0.5
self.k_i_l = 0.3
self.k_d_l = 0.05
# angular velocity PID gains
self.k_p_a = 0.6
self.k_i_a = 0.4
self.k_d_a = 2.5
```



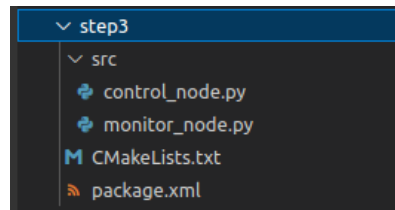
گام سوم (توضیحات کد و آماده سازی)

این تمرین بسیار شبیه به ویدیو و کد موجود در هندزان میباشد و از آن کمک گرفتیم. در این قسمت نیز ابتدا یک پکیج با نام step3 ایجاد میکنیم و dependency هایی که ممکن است به کار بیایند را هم اضافه کنیم.

- `catkin_create_pkg step3 rospy std_msgs sensor_msgs nav_msgs`

```
maahdi@maahdi:~/Desktop/Robotics/project3/hw3_ws$ cd src/
maahdi@maahdi:~/Desktop/Robotics/project3/hw3_ws/src$ catkin_create_pkg step3 rospy std_msgs sensor_msgs nav_msgs
Created file step3/package.xml
Created file step3/CMakeLists.txt
Created folder step3/src
Successfully created files in /home/maahdi/Desktop/Robotics/project3/hw3_ws/src/step3. Please adjust the values in package.xml.
maahdi@maahdi:~/Desktop/Robotics/project3/hw3_ws/src$
```

حال به سراغ ساخت نودها میرویم. برای این منظور در فولدر src مربوط به step3 میرویم و دوفایل پایتون با نام های `control_node.py` و `monitor_node.py` میسازیم. نود `monitor_node.py` نیز برای نمایش مسیر ربات در rviz خواهد شد.



حال در گام بعد به سراغ نوشتن کد مربوط به هریک از نودها میرویم. کد مربوط به نود `monitor_node` مانند قبل میباشد و توضیح آن در قسمت های قبل داده شد. در این جا فقط به کد نود `control_node` میپردازیم. توضیحات تا حد زیادی مانند قبل است. در این جا یک متغیر `direction` داریم که باتوجه به متغیر `follow_type` مقدارش میتواند ۱ یا -۱ باشد. چنانچه ربات دنبالگر راست باشد این ضریب منفی ۱ است و به عنوان ضریبی در حاصل جمع $P+I+D$ ضرب میشود. همچنین در این جا یک تابع `distance_from_wall` داریم که با توجه به اسکن های لیزر لایدار میتواند فاصله تا دیوار را برگرداند. دقت شود اگر دیوار در سمت راست باشد باید بین مقادیر در `range` بین ۱۸۰ تا ۳۶۰ مینیمم بگیریم و اگر سمت چپ باشد بین ۰ تا ۱۸۰ را بررسی کنیم. باتوجه به اینکه فاصله مطلوب تا دیوار را ۱.۵ گرفتیم بنابراین اختلاف فاصله تا دیوار و این مقدار ۱.۵ را باید سعی کنیم که ۰ کنیم و PID در این جهت تلاش میکند. بقیه روال کار مانند قسمت های قبل میباشد.

```
#!/usr/bin/python3
import rospy
from geometry_msgs.msg import Twist
from sensor_msgs.msg import LaserScan
import matplotlib.pyplot as plt
```

```

class PIDController():
    def __init__(self):

        rospy.init_node('controller', anonymous=False)
        self.cmd_vel = rospy.Publisher('/cmd_vel', Twist, queue_size=10)
        rospy.on_shutdown(self.on_shutdown)

        # if we want to follow wall on right hand the robot should rotate
        # in the inverse direction of default state. so we multiply -1 coefficient
        # to (P+I+D)
        self.follow_type = rospy.get_param("/controller/follow_type")
        if self.follow_type == "right":
            self.direction = -1
        else:
            self.direction = 1

        # angular velocity PID gains
        self.k_p = 0.8
        self.k_i = 0.0005
        self.k_d = 25

        self.dt = 0.005
        self.v = 0.1
        self.D = 1.5

        rate = 1/self.dt
        self.r = rospy.Rate(rate)
        self.errs = []

    def distance_from_wall(self):
        """
        this method give us the distance from wall with the help of scans
        coming from the lidar sensor. if direction is -1 means that we
        want to have on the right hand of robot and ranges [180:] is important to
        be checked and if direction is 1 it is vice versa.
        """
        laser_data = rospy.wait_for_message("/scan" , LaserScan)
        if self.direction==1:
            rng = laser_data.ranges[:180]
        else:
            rng = laser_data.ranges[180:]
        d = min(rng)
        return d

    def control(self):
        """
        this function is the main function of this code. we calculate the

```

```

distance error and try to calculate P, I, D terms. the summation of these
terms define our angular velocity.
'''

d = self.distance_from_wall()
sum_i_theta = 0
prev_theta_error = 0
move_cmd = Twist()
move_cmd.angular.z = 0
move_cmd.linear.x = self.v
while not rospy.is_shutdown():
    self.cmd_vel.publish(move_cmd)
    rospy.loginfo(f"d : {d}")
    err = d - self.D
    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 - prev_theta_error)

    move_cmd.angular.z = self.direction * (P + I + D)
    prev_theta_error = err
    move_cmd.linear.x = self.v

    d = self.distance_from_wall()
    self.r.sleep()

def on_shutdown(self):
    '''
    this method plot error of linear and angular velocity separately.
    '''
    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)

if __name__ == '__main__':
    try:
        pidc = PIDController()
        pidc.control()
    except rospy.ROSInterruptException:
        rospy.loginfo("Navigation terminated.")

```

در مرحله بعد باید تمامی کدهای پایتون را executable کنیم. برای این کار لازم است در ترمینال در پکیج step3 کد زیر را اجرا کنیم:

- `chmod +x src/*.py`

```
mahdi@mahdi:~/Desktop/Robotics/project3/hw3_ws/src$ cd step3/
mahdi@mahdi:~/Desktop/Robotics/project3/hw3_ws/src/step3$ chmod +x src/*.py
mahdi@mahdi:~/Desktop/Robotics/project3/hw3_ws/src/step3$ cd src/
mahdi@mahdi:~/Desktop/Robotics/project3/hw3_ws/src/step3/src$ ls
control_node.py  monitor_node.py
mahdi@mahdi:~/Desktop/Robotics/project3/hw3_ws/src/step3/src$
```

سپس به سراغ نوشتن لانچ فایل میرویم. در این جا به روشی کمی متفاوت تر با قبل کار میکنیم. در همین پکیج step3 لازم است تا یک فولدر launch ایجاد کنیم. قبلا لانچ فایل با نام my_empty_world را که در پوشه لانچ مربوط به turtle_bot بود را include میکردیم. در این جا یک فایل مشابه آن مستقیما همینجا اضافه میکنیم. اسم این لانچ فایل را turtlebot3_square_wall.launch میگذاریم. محتوای آن به صورت زیر است. دقت شود که زاویه اضافه شدن ربات به map را هم در اینجا تعریف کردیم. همچنین world_name را هم مشخص کردیم.

```
<launch>
  <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" value="0.0"/>

  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="world_name" value="$(find step3/worlds/square.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 $(arg yaw) -param robot_description" />
</launch>
```


همچنین یک control.launch ایجاد کنیم. لانچ فایل به صورت زیر است. ابتدا لانچ فایل قبلی را که نوشتیم include میکنیم و میگوییم در آن آرگومان های ورودی مثل مکان اولیه ربات و زاویه اولیه چه باشند. طبق خواسته سوال در این جا ربات در مکان (۰و۰) و با زاویه ۹۰ درجه یا همان ۱.۵۷ رادیان باید اضافه شود. سپس launch file مربوط به rviz را include میکنیم تا آن را هم برای نمایش مسیر ربات بالا بیاورد.

همچنین نودهای control_node.py و monitor_node.py را با نامهای controller و monitor بالا بیاورد. نود monitor که برای رسم مسیر لازم است و آن هم باید بالا بیاید.

```
<launch>

  <arg name="follow_type" default="left"/>

  <include file="$(find step3)/launch/turtlebot3_square_wall.launch">
    <arg name="x_pos" value="0.0"/>
    <arg name="y_pos" value="0.0"/>
    <arg name="z_pos" value="0.0"/>
    <arg name="yaw" value="1.5708"/>
  </include>

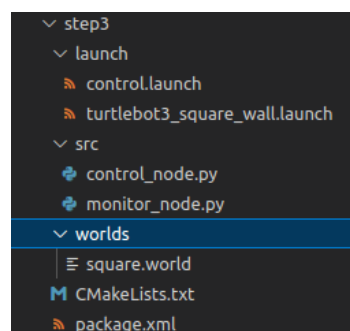
  <include file="$(find turtlebot3_gazebo)/launch/turtlebot3_gazebo_rviz.launch"/>

  <node pkg="step3" type="control_node.py" name="controller" output="screen">
    <param name="follow_type" value="$(arg follow_type)" />
  </node>

  <node pkg="step3" type="monitor_node.py" name="monitor"></node>

</launch>
```

همچنین لازم است یک فولدر به نام worlds در src مربوط به پکیج step3 ایجاد کنیم و فایل square.world را در آنجا اضافه کنیم:



سپس در آخر لازم است تا به دایرکتوری ورک اسپیس برویم و `catkin_make` را صدا بزنیم. سپس برای استفاده لازم است تا ابتدا سورس کنیم و سپس ربات را اکسپورت کنیم و در نهایت `roslaunch` را صدا بزنیم:

- `. devel/setup.bash`
- `export TURTLEBOT3_MODEL=waffle`
- `roslaunch step3 control.launch follow_type:=left`

```
mahdi@mahdi:~/Desktop/Robotics/project3/hw3_ws$ . devel/setup.bash
mahdi@mahdi:~/Desktop/Robotics/project3/hw3_ws$ export TURTLEBOT3_MODEL=waffle
mahdi@mahdi:~/Desktop/Robotics/project3/hw3_ws$ roslaunch step3 control.launch follow_type:=left
... logging to /home/mahdi/.ros/log/344a6cec-0254-11ee-b4cc-1be5bd5aea40/roslaunch-mahdi-733127.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

xacro: in-order processing became default in ROS Melodic. You can drop the option.
xacro: in-order processing became default in ROS Melodic. You can drop the option.
started roslaunch server http://mahdi:34301/

SUMMARY
=====
PARAMETERS
* /controller/follow_type: left
* /gazebo/enable_ros_network: True
* /robot_description: <?xml version="1....
* /robot_state_publisher/publish_frequency: 50.0
* /robot_state_publisher/tf_prefix:
* /roscdistro: noetic
* /rosversion: 1.15.15
* /use_sim_time: True

NODES
```

گام سوم (اجرا و نتایج)

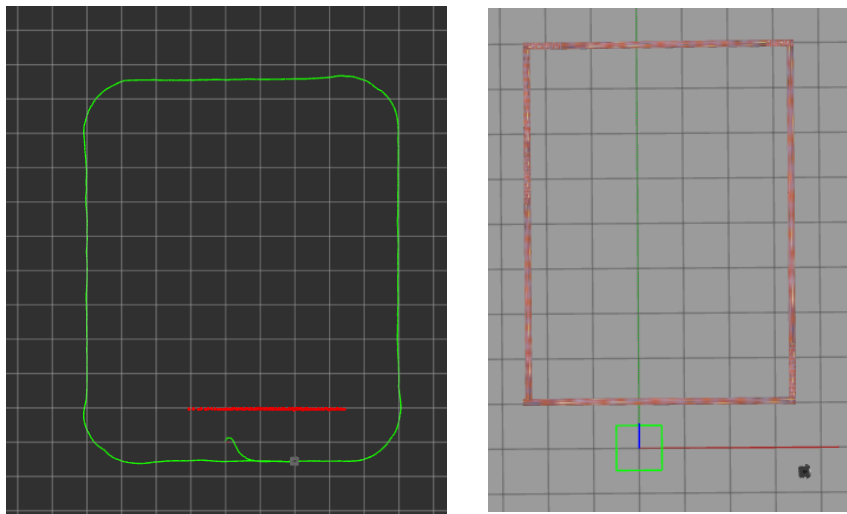
موارد زیر را در گزارش خود قرار دهید:

۱. نمای شکل تولید شده از حرکت ربات در شبیه ساز Rviz را نشان دهید.

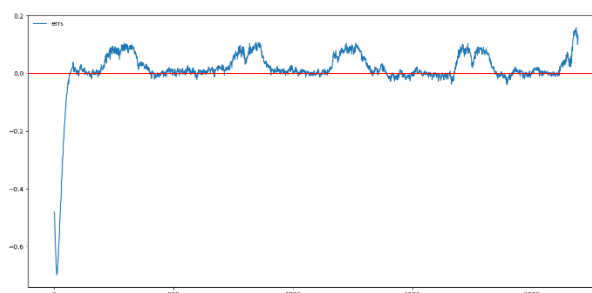
پارامترها: ۱- فاصله از دیوار: 1.5m ۲- سرعت خطی ثابت: 0.1m/s ۳- کنترلر روی سرعت زاویه‌ای در لانچ فایل براساس follow_type میتوان ربات را دنبالگر چپ یا راست تعریف کرد. این مقدار را از طریق ورودی در cmd تعریف میکنیم که میتواند مقدار left یا right بگیرد. برای دنبالگر چپ دستور زیر را به صورت زیر اجرا کردم:

```
mahdi@mahdi:~/Desktop/Robotics/project3/hw3_ws$ roslaunch step3 control.launch follow_type:=left
... logging to /home/mahdi/.ros/log/77267908-023f-11ee-b4cc-1be5bd5aea40/roslaunch-mahdi-683974.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.
xacro: in order processing became default in ROS Melodic. You can drop the option
```

نتیجه به صورت زیر شد:



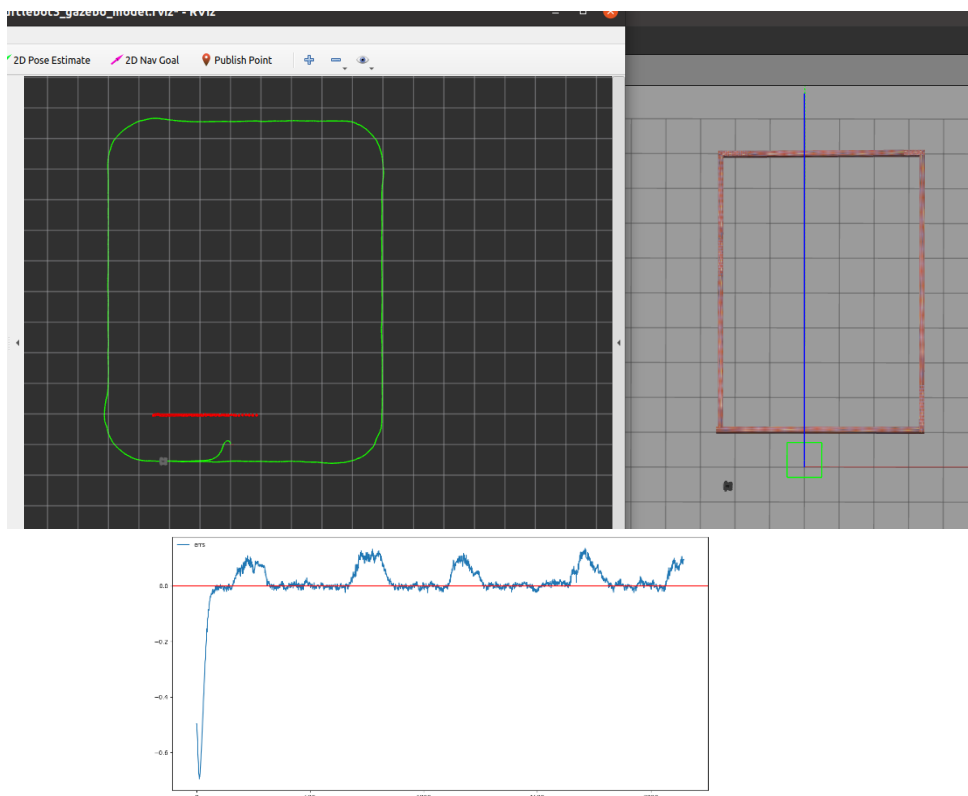
همچنین نمودار خطا به صورت زیر است. سعی شد تا حد ممکن جلوی اورشوت را با gain دیفرانسیلی (k_d) بگیرم ولی به هر حال کامل از بین نرفت و در گوشه‌ها و چرخش‌ها کمی اورشوت داریم.



برای دنبالگر راست به صورت زیر دستور را اجرا میکنیم. دقت شود که برای این حالت از همان gainهای قبلی استفاده کردیم .

```
mahti@mahti:~/Desktop/Robotics/project3/hw3_ws$ roslaunch step3 control.launch follow_type:=right
... logging to /home/mahti/.ros/log/6b6495a4-0240-11ee-b4cc-1be5bd5aea40/roslaunch-mahti-688589.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.
```

نتیجه در Rviz و نمودار خطا را در زیر میتوانید مشاهده کنید.



۲. تشریح کنید چه ضرایبی برای P , I و D پاسخ مناسبی را ارائه میکند.

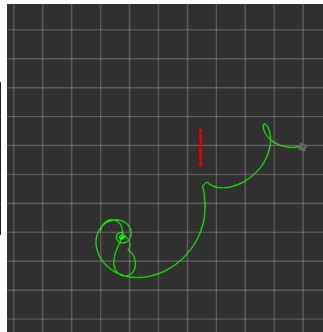
ضرایب مناسب برای gainها به صورت زیر میباشد که با آنها توانستیم نتایج بالا را بگیریم که تقریباً نتایج مطلوبی هستند.

```
# angular velocity PID gains
self.k_p = 0.8
self.k_i = 0.0005
self.k_d = 25
```

مراحل رسیدن به gain مناسب:

پارامترها : ۱- فاصله از دیوار: 1.5m ۲- سرعت خطی ثابت: 0.1m/s ۳- کنترلر روی سرعت زاویه‌ای
برای رسیدن به این gainها ابتدا با توجه به کارهای قبلی برای سرعت زاویه‌ای مقادیر زیر را برای gain انتخاب کردیم و نتیجه را هم در عکس سمت راست مشاهده میکنید.

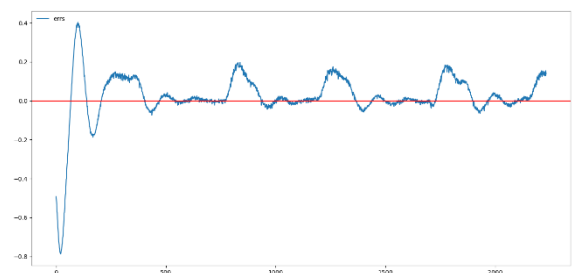
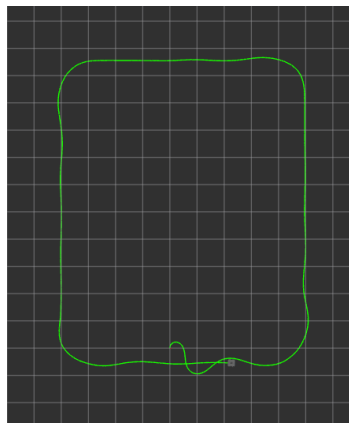
```
# angular velocity PID gains  
self.k_p = 0.3  
self.k_i = 0.003  
self.k_d = 0.9
```



```
: d : 1.1824603080749512  
: d : 1.1668511629104614  
: d : 1.1587543487548828  
: d : 1.249727725982666  
: d : inf  
: d : 1.3714531660079956  
: d : 1.0475904941558838  
: d : 1.147385835647583  
: d : 2.226879596710205
```

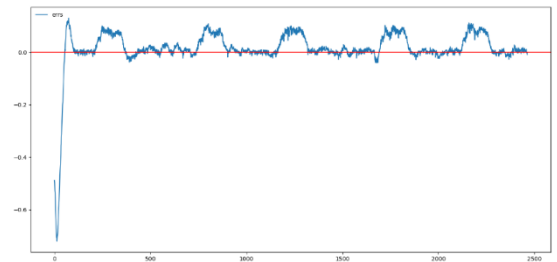
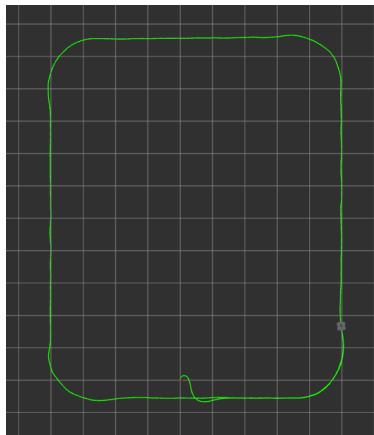
چون k_p به اندازه کافی نبود و از طرفی overshoot داشتیم که باز چون k_d کم بود باعث شد از دیوار فاصله بگیرد و حتی مقادیر inf در فاصله ظاهر شود (چون دیگر سمت چپ روبه مانع نبوده که فاصله برگرداند) طبیعتاً سرعت زاویه‌ای به بینهایت میل کند و ربات ناپایدار شود. پس مقدار k_p را کمی بیشتر میکنیم که در مواقعی که باید بپیچد با توجه به آنکه سرعت خطی 0.1 و ثابت است بتواند به خوبی بپیچد و همچنین این بالا رفتن سرعت زاویه‌ای نیازمند آن است که k_d بیشتر شود تا جلوی overshoot را بگیرد. پس مقادیر زیر را گذاشتیم و نتیجه زیر را گرفتیم:

```
# angular velocity PID gains  
self.k_p = 0.5  
self.k_i = 0.005  
self.k_d = 8
```



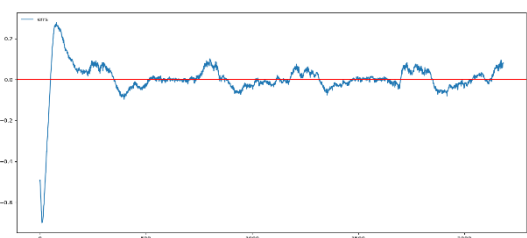
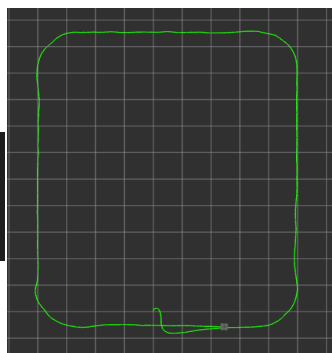
همانطور که دیده میشود اولاً در رسیدن به فاصله مناسب از دیوار در حالت اولیه مقدار ارور زیاد بوده و به نظر میرسد که کنترلر با سرعت مناسب به ارور پاسخ نمیدهد و باید کمی مقدار k_p را بیشتر کرد. از اون طرف در پیچیدن‌ها اورشوت زیاد است و کلاً نوسانات زیاد است پس باید k_d را هم بیشتر کنیم. ولی خب در کل اکنون ربات به یک پایداری نسبی رسیده است. حال مقادیر زیر با ادامه سعی و خطاها امتحان میکنیم که نتیجه نزدیک به مطلوب ماست.

```
# angular velocity PID gains
self.k_p = 0.8
self.k_i = 0.008
self.k_d = 19
```



فقط همانطور که دیده میشود در ابتدای کار وقتی از دیوار برمیگردد به فاصله ۱.۵ یک overshoot داریم. برای این منظور مقدار k_d را بالاتر برده و برابر با ۲۵ میگذاریم که نتیجه مطلوب نهایی به دست بیاید. فقط نکته دیگر ترم انتگرال گیر میباشد. این ترم خیلی در اینجا تاثیر گذار نیست و ما خطای steady state نداریم. ولی اگر مقدار این gain را زیاد بگیریم، خواهیم داشت:

```
# angular velocity PID gains
self.k_p = 0.8
self.k_i = 2
self.k_d = 25
```



همانطور که دیده میشود در این مسئله اگر مقدار k_i زیاد باشد بدتر باعث overshoot و همچنین خارج شدن از وضعیت هموار میشود. بنابراین تا جای ممکن مقدار آن را کوچک میگیریم. در نهایت به همان مقادیر گفته شده میرسیم:

```
# angular velocity PID gains
self.k_p = 0.8
self.k_i = 0.0005
self.k_d = 25
```