Middle East Technical University Robotics Graduate Program

ROB 501

Fundamentals & Emerging Topics in Robotics

Spring 2021

ROS Take Home Exam 1

Due Date: 01.05.2022 23.55

1 Introduction

This assignment aims to familiarize you with the ROS Ecosystem and the application development procedures using ROS and Python.
In this assignment, your task is to implement several algorithms for the turtlesim and also establish connections between different nodes using ROS Services and ROS Topics.
Keywords— Catkin Workspace, ROS Nodes, ROS Services, ROS Topics, turtlesim, Launch Files, roslaunch, roscore, rosrun, rosparam

2 Problem Definition

Turtlesim is the official tutorial material for ROS environment. In this application, we have a turtle(Tuturtle) on an empty map. The package has several implemented Nodes, Topics and Services to show you the inner workings of such systems.

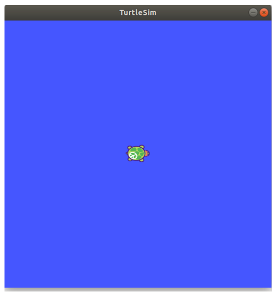

Figure 1: Screenshot of turtlesim_node

In this introductory tool, you can interact with several things, such as the movement of the turtle, background color, number of turtles, etc. In order to run turtlesim on your machine, you need to execute below commands:

First, you need to have a roscore running in order to use ROS system:
```
$ roscore
```
Then, you need to run the turtlesim_node package:
```
$ rosrun turtlesim turtlesim_node
```

With these two lines, you will have a working turtlesim node on your system.
Another interesting node about turtlesim is the turtle_teleop_key node, which is a node that enables you to move your turtle with arrows in your keyboard. In order to better understand the tasks in Section 2, you are strongly encouraged to interact with turtlesim.

In this assignment, we will extend the turtlesim with our own nodes, thus we will be able to command our turtle with scripts, instead of teleop node. In the tasks in the following section, we are going to incrementally build our application.

3 Tasks

3.1 Task 0: Warmup(0 Points)

In this task, you are required to interact with the turtlesim environment. You need to understand the roles and spesifications of Service and Topics provided in the environment. Please make sure that you interact with the following Services and Topics:

- Topics:
  1. /turtle1/cmd_vel
  2. /turtle1/pose
- Services
  1. /turtle1/teleport_absolute
  2. /turtle1/teleport_relative
  3. /clear
  4. /reset

The Services and Topics listed above are fairly simple and straightforward, and they provide you a nice overview into the workings of the ROS Ecosystem.

3.2 Task 1: Catkin Workspace(5 Points)

You need to create a catkin workspace in your machine. This workspace will allow you to implement your system in a modular way.

3.3 Task 2: Controller Node (40 points)

Controller node is our main node. In this node, we will implement our motion algorithms and our interaction with turtlesim node.
This node is mainly responsible for reading pose data(current location) from the turtlesim, goalpoint data from the Navigation Node and then execute some movement commands so that our turtle moves to the goalpoint fetched from Navigation Node.
In our task, we will move the turtle through a series of goalpoints to form different shapes. The turtle will spawn at the center by default, and then go to our first waypoint, followed by the others. When the last waypoint is reached, the turtle will head back home. In detail, we can list its responsibilities as below:

- Fetch required rosparams (**delta, distance, max_linear_velocity, max_angular_velocity**) if they exist. If one or more than one param does not exist, please refer to the list below:
  - distance: It is used to calculate if the drone arrived at the waypoint (default: 0.1)
  - delta: It is used for calculating the float equality check (default: 0.001)
  - max_linear_velocity: Maximum linear velocity (default: 2.0)
  - max_angular_velocity: Maximum angular velocity (default: 1.0)
- Constantly read pose data from /turtle1pose Topic of the turtlesim, (i.e. Subscribe to that Topic)
- Request next goalpoint from Navigation Node by using a custom ROS Service, (i.e., you are required to build a new ROS Service)
- Calculate the heading given your current position and goalpoint positon,
- Change your heading to match the required heading by publishing into /turtle1cmd_vel (Please experiment by hand beforehand using the following command:

  ```
  $ rostopic pub /turtle1/cmd_vel geometry_msgs/Twist "linear:
    x: 0.0
    y: 0.0
    z: 0.0
  angular:
    x: 0.0
    y: 0.0
    z: 0.0"
  ```

  However, you need to discover what the above parameters does by changing them manually
- When aligned, you need to send a velocity command to move the turtle (Hint: Only one linear parameter is enough to move the turtle, please determine which one it is)
- Check if the turtle has reached to the point (this will be determined by comparing the distance between the current position and goalpoint to a DISTANCE value, it is visualized below)

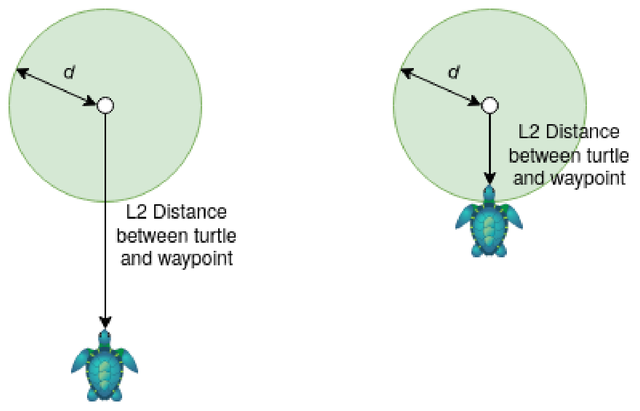
d
L2 Distance between turtle and waypoint
d
L2 Distance between turtle and waypoint

Figure 2: The method to determine whether the turtle has reached the waypoint. It has not reached on the left figure, whereas it has reached on the right figure.

- If reached, request the new waypoint, and iterate the algorithm described above.
- If the list of waypoints is consumed, return home.

3.4 Task 3: Navigation Node (15 points)

This is a fairly simple Node compared to the Controller node. It is responsible for providing the Controller Node waypoints. The reason for giving this responsibility to Navigation Node instead of Controller Node is to achieve Single-responsibility principle in code (i.e. each piece of code must responsible for only one thing) and also to force you into writing your own Services and Topics.
The Node should provide the client with target points when requested(i.e. when the Service is called). The details are also simple:

- The system should implement a Service
- It should fetch the waypoints from rosparam (waypoints).
- When some client asks for waypoint, it should return it in response.

3.5 Task 4: Launch File (5 points)

You need to write a Launch file, so that all the Nodes can be run easily. Please do not forget to implement rosparam logic in your launch file.

3.6 Task 5: Write a Report (30 points)

You need to write a README.md file, explaining your design choices and implementations. You are required to explain your Services and Topics, high-level overview of your algorithms and **the required commands for me to run your code**. Codes that are not clearly explaining the listed things will be penalized. You are also required to show a rqt_graph of your system.

3.7 Code Clarity and Comments (5 points)

Your code will be evaluated in terms of clarity, therefore it should be written clearly and explained thoroughly with comments. Apart from this assignment, nice written and commented code allows others to understand your way of thinking on the given problem easily.

4 Specifications

- This project should be implemented in Python 2.7 and ROS Melodic, and must run on an Ubuntu 18.04 LTS machine.
- If you need an external library, please ask before using. However, only rospy, some standard message libraries and numpy is enough for the scope of this project.
- You are allowed to implement your own Services and Topics apart from the ones described above.
- This is an individual assignment. High level discussions are encouraged, however, this assignment should solely be your own work.
- Your code will be evaluated with different params.yaml files. One params.yaml file is given to you as an example.

5 Submission

You will submit the following files:

- .py files you implemented
- Any Service file you implemented
- Any Message file you implemented
- README.md file explaining your logic
- rqt_graph of your system

6 Hints

- Float equality is a tricky business. To check if two floating numbers are nearly equal, please follow this link.

7 Extra Resources

- Python Refresher
- Markdown Guide
- How to write a Service-Client
- How to write Publisher/Subscriber