UNIVERSITÀ DI BOLOGNA

School of Engineering

Master Degree in Automation Engineering

Industrial Robotics

Report assignment ROS

Professors:
Claudio Melchiorri,
Gianluca Palli

Students:
Stefano Mulargia

Academic year 2020/2021

Contents

Chapter 1

Setup

The student is asked to setup the simulation of the TurtleBot3 burger robot using the Gazebo simulation environment and the TurtleBot3 House scenario. Note that during the exam evaluation, the robot can be spawned on the simulation scenario in a random position and additional obstacles can be present.

1.1 Workspace environment

Since there are many ditributions of ROS it worth to point it out that this project has been developed under ROS Melodic and Ubuntu 18.04.6 LTS (Bionic Beaver).

In order to setup the workspace environment, it is necessary to run in the shell the bash file *create_sw.sh*. This file provides the creation of the necessary folder.

1.2 Simulation environment

In order to set up the simulation environment, it is necessary to run in the shell the bash file *start_simulation.sh*

In this bash file, there are two instructions of particularly interesting, the first one define which type of TurtleBot (*burger, waffle, waffle_pi*), we will use *burger* :

```
export TURTLEBOT3_MODEL=burger
```

Now in order to launch the simulation environment **Gazebo** we use the command

```
roslaunch turtlebot3_big_house turtlebot3_big_house.launch
```

once *Gazebo* and the robot are loaded inside the *big_house*, our robot by default will be placed in position (-3,1). This position can be changed by setting

```
roslaunch turtlebot3_big_house
```

```
turtlebot3_big_house.launch x_pos:=x y_pos:=y
```

where x and y are the coordinates of the desired starting point.

Important note: I have not implemented any policy to check if the coordinates entered by the user are meaningful. You can set a coordinate that correspond to a wall or outside the house, that does not make any sense in our application.

Finally, it is requested to allow the modification of the Gazebo simulation environment by picking and placing additional obstacles. It is possible to choose among the predefined obstacles, which correspond to simple geometric solids such as cubes, spheres, etc., but also to upload more complex ones by the insert window. In particular, the Gazebo model database is a repository of several types of models, including robots tables, buildings, etc. In addition, the user can rotate, translate and re-scale objects.

Chapter 2

Map creation

The student is asked to write a code that makes the robot able to move autonomously in the environment to create a map covering the larger possible part of the environment. Note that no previous knowledge of the map will be considered during the exam.

2.1 SLAM

In order to perform the mapping of an unknown environment, we need to implement a SLAM algorithm. "The SLAM (Simultaneous Localization and Mapping) is a technique to draw a map by estimating the current location in an arbitrary space. The SLAM is a well-known feature of TurtleBot from its predecessors."

ROS allow us to implement an easily some kind of algorithm in the SLAM node, which can be activated with:

```
roslaunch turtlebot3_slam turtlebot3_slam.launch

slam_methods:=gmapping
```

In particular for this project, it has been used the gmapping algorithm already provided by the ros package.

2.2 Explore_lite

In order to perform the map I used the well known package exploration lite [1] which implements a frontier exploration following the idea that we can find in Yamauchi's paper [2].

Since, according to the documentation, this package does not create its own costmap, but instead it simply subscribes to "nav_msgs/OccupancyGrid"

message we need to launch also the **Movebase** launch file.

So, in conclusion, in order to implement it in an easy way was written a bash file that launches, Roscore to start up the master, Gazebo, SLAM algorithm with gmapping and the Move_base in order to send the command to the turtlebot and finally the Explore_lite package.

```
roslaunch explore_lite explore.launch
```

It will subscribe to the topic "map" and update the costmap to "max_update".

2.3 Plot

Here is reported the image obtained by the mapping process

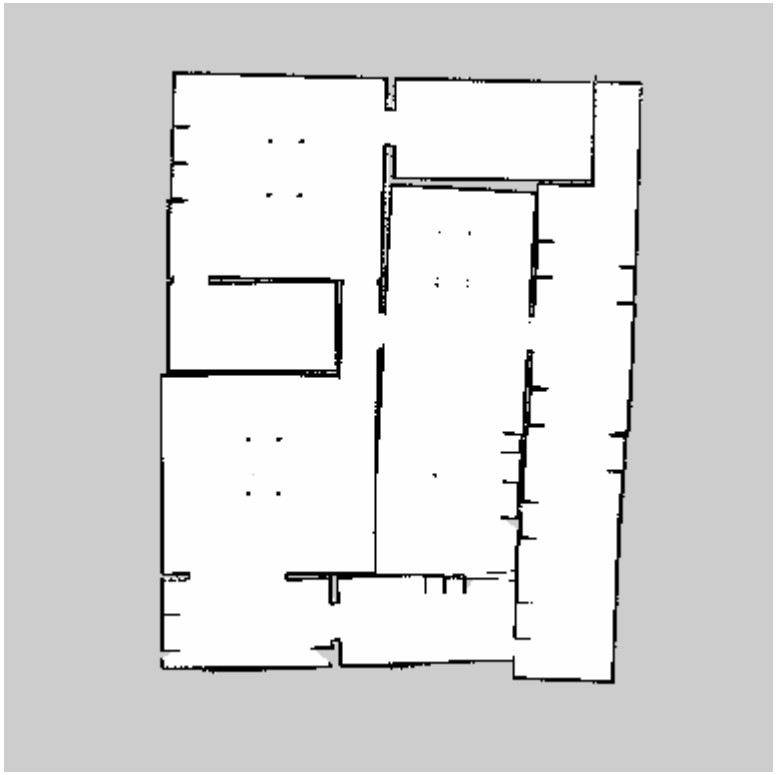

Figure 2.1: map big_house

As we can see the walls are not perfectly straight and there is a small angle on the bottom that is not perfectly mapped. However we can consider this map a good result for a lot of cases.

After some times the map_server node is called and it shows up in the first terminal and you can answer "y" when you want to save the map. Note

that I have put a sleep of 60 second, but the mapping process requires more time, so,you are allowed to save the map before the explore_lite has finished its job, so just wait until it completes the map.

2.4 Note

It seems that sometimes the explore_lite package sees that the map is full but it is not true is you look ad Rviz. If you stop the roslaunch with "ctrl+C" and start the launch file in a now terminal to the map. The good old turn off and turn on works also in this case.

2.5 plot

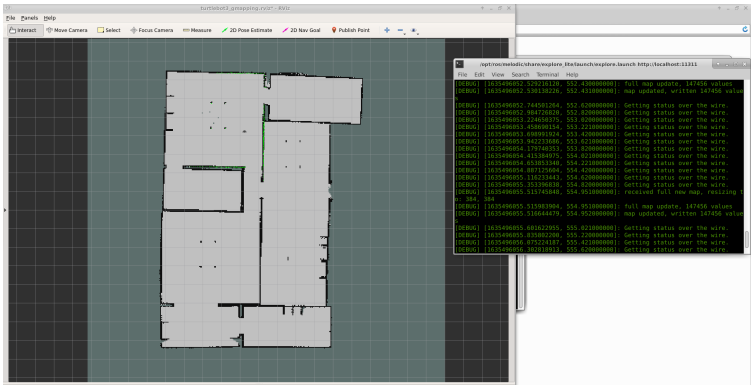

Figure 2.2: map big_house

Note full map update which is not true in practice, the robot has not covered the room on the right.

Chapter 3

Navigation

The student is asked to write a code that after the mapping of the scenario, makes the robot able to navigate in the environment reaching a set of goals contained in a text file. The alignment between the map and the navigation system must be performed. SUGGESTION: use the initpose topic to to set the pose of the robot (the base_footprint TF) with respect to the map frame (the map TF) into the robot localizer (amcl, Adaptive Monte Carlo Localization) immediately after the navigation stack is launched. ALTERNATIVE: send an empty request to the global_localization service to let amcl to spread the particle around to try to find the actual robot position (from the console, you can use the command rosservice call global_localization "{}").

3.1 Problem formulation

As before, to simplify the startup of all the nodes a script "navigation.sh" is provided. It will execute Roscore in case it is not already running, open the big house with Gazebo (note that I added a command gui:=True/False, in case you do not need the gui. The default configuration is False). Then:

init_pose: the init pose python script allow us to handle with possible misalignment between the map and the navigation system. Once our map is in a correct position the robot start to localize itself through the "amcl_init".

amcl_init: the amcl is used to recover and publish the current position of the robot.

load map: through the navigation package we load the map in the Rviz and make it running all the nodes that we will use later.

simple_navigation_goal: this python script is basically composed of two functions "movebase_client" which is in charge to create an action called

"move_base" and wait for a server response from the node. Since the set of goal is written in a txt file called "navigation_goal, the method open is used to access to this information. A second function "reach_goal is used to extract the coordinates to provide the correct destination to the MoveBaseGoal constructor. The goal is sent to the move_base client and the robot start moving. Once it has reached the goal the "wait_for_result become true.

Chapter 4

Optional exercise

Suppose that

- in order to kill the coronavirus, the robot is equipped with a set of UV lamps able to spread all around the robot a light power $P_l = 100\mu W m^2$;
- the UV energy E at point (x, y) and time t can be computed as

$$E(x, y, t) = \int_0^t \frac{P_l}{(x - p_x(\tau))^2 + (y - p_y(\tau)^2} d\tau$$

where $p_x(t)$ and $p_y(t)$ represent the robot position along the x and the y axis respectively at time t;

- any obstacle completely stop the UV power propagation;
- the light power emitted at a distance lower that 0.1 meters from the robot is zero due to the robot encumbrance;
- the room of interest can be discretized as a grid with resolution 0.2 m.

The student is asked to implement a strategy to define a path able to cover a selected room in the map in order to guarantee a minimum value of 10 mJ of the UV energy over the room to ensure the coronavirus is inactivated.

4.1 Problem formulation

As before a script is provided allowing to launch all necessary file.

The code is basically the same as the navigation problem except that instead of the simple_navigation_goals.py two now code are provided.

4.2 create_matrix_map:

this file is used to generate a sequence of point and sample it from the costmap in order to create a txt file with all the coordinates of the map and a cost value that is basically the probability that a given point is occupied (100), empty (0) or unknown (-1).

4.3 Cover_all_map

This file is used to manipulate the txt as an array table and send the goals to the robot.

4.3.1 Table

The Table has the following structure

x	y	cost	Energy	distance	dst_x	dst_y
-3.0	1.0	0.0	0.0	0.0	0.0	0.0

Table 4.1: Example of the Table

Then it will be manipulated in the code as an array while can be saved with the numpy method

```
np.savetxt()
```

in order to provide a text file.

This trick allows us to use the text file as done in the previous exercise to send the goal and move the robot.

Note: sometimes you see the robot stuck. It is completing a recovery behaviour. It need some times to find a new trajectory, be patient. If you look at the terminal of the turtlebot3_navigation you will see all the plan and messages that you need to understand this issue.I noticed that the problem arises usually when the robot is close to some wall.

4.3.2 Update Energy Table

This function is used to update the table and keep track of Energy "stored", distance from the turtlebot of the other point.

The distance is a simple euclidean distance computed with

```
np.linalg.norm()
```

An improvement can be to consider the distance in base of the length of the trajectory that the robot

4.3.3 Remove from the Table

This function is used to remove from the table the coordinates that have received a quantity of energy grater than 10mJ.

The code is basically a while loop in the array Table the use the numpy method

```
np.delete()
```

in order to eliminate the right rows.

4.3.4 Integral

The integral it's implemented as a sum. I'm assuming that an iteration of my code is an infinitesimal $d\tau$. If this assumption holds we can implement the integral just as a sum so:

```
Table[i][3] += (100*10^(-6))/(dst_x^2+dst_y^2)
```

where dst_x is the distance from the coordinate that I am considering and the robot position on the x-axis. Same for dst_y considering now the y-axis.

In order to compute the distance I need to know the position of the robot. To have this information I used the command

```
rospy.Subscriber('/odom',Odometry,get_position)
```

where the callback get_position writes over a global variable the actual position of the robot so that I can use later in the update energy function.

4.3.5 Movebase_client

similar to the previous case it is used to pick from the table the destination converted from the text file. It performs a while loop until it has completed all the map.

Bibliography

[1] Jiří Hörner. *Map-merging for multi-robot system*. Bachelor's thesis. Prague, 2016. URL: https://wiki.ros.org/explore_lite.

[2] B. Yamauchi. "A frontier-based approach for autonomous exploration". In: *Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'97. 'Towards New Computational Principles for Robotics and Automation'*. 1997, pp. 146–151. DOI: 10.1109/CIRA.1997.613851.