University of Burgundy

Master of Science in Computer Vision-2nd Year

Robotics Project Module

Mastering in ROS: Turtlebot3

Team Members

Zhiqiang Pan

Qingqing NIE

Under the supervision of

Dr. Ralph Seulin

Dr. Daniel Braun

Dr. Raphael Duverne

CONTENTS

1. Introduction

1.1 Scene

This first part of the project will be based on a new environment, which is a map of a real Costa Coffe in Barcelona.

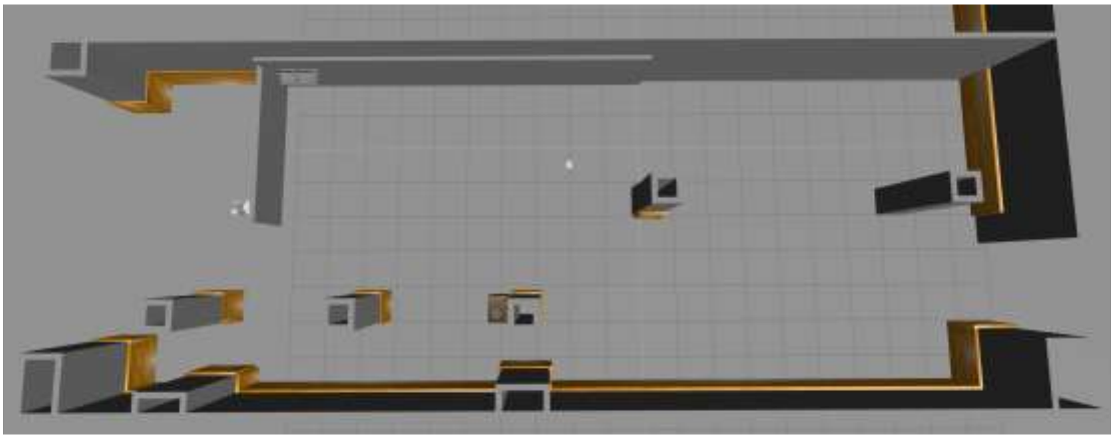


Figure 1. Costa Coffee Room

We use **keyboard_teleop.launch** launch to move Turtlebot3 Model. Take a stroll around to have a closer look at all the terrain and elements.

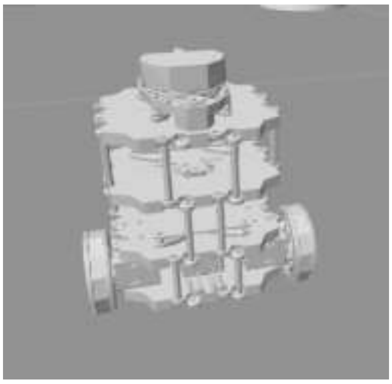


Figure 2. Turtlobot3

1.2: Task

1. Create a script that moves the robot around with simple /cmd_vel publishing. See the range of movement of this new robot model.
2. Create the mapping launches and map the whole environment. You must finish with a clean map of the full cafeteria.
3. Setup the launch to be able to localize the Turtlebot3 robot.
4. Set up the move base system so that you can publish a goal to move_base and Turtlebot3 can reach that goal without colliding with obstacles.
5. Create a program that allows the Turtlebot3 to navigate within the environment following a set of waypoints. The 3 following spots are mandatory:

2. Strategy

2.1 Moving robot

We will use keyboard teleop to move around the turtlebot in gazebo

roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch

After we launched, the following instruction will be appeared to the terminal window.

Figure 3

2.2 Creating a Map

The first thing to perform ROS Navigation, is to create a Map of the environment you want to navigate.

roslaunch micro_project start_mapping.launch

For that, you are going to need the slam_gmapping node that the Navigation Stack provides.

Figure 4

As we move the turtlebot around the built environment, now we can save the map which will generate the .png and .yaml files.

rosrun map_server map_saver -f ~/micro_project/maps/my_map

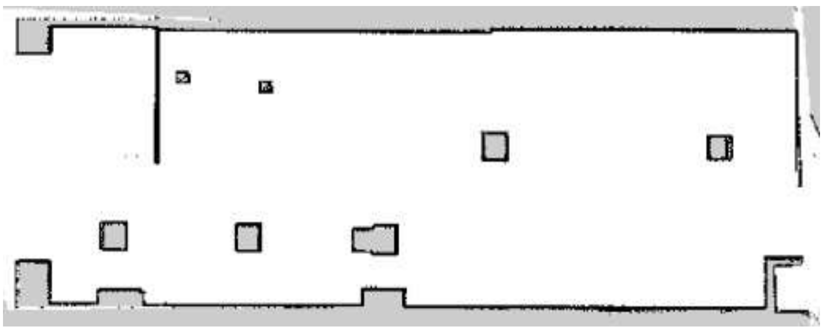


Figure 5.my_map

2.3 Localize the robot

Initial Pose Estimation must be performed before running the Navigation as this process initializes the AMCL parameters that are critical in Navigation. TurtleBot3 must be correctly located on the map with the LDS sensor data that neatly overlaps the displayed map.

roslaunch micro_project start_location.launch

And launch RViz to be able to visualize the localization process. We can use the same setup we used for the mapping process, adding 1 more display: **Pose Array**.

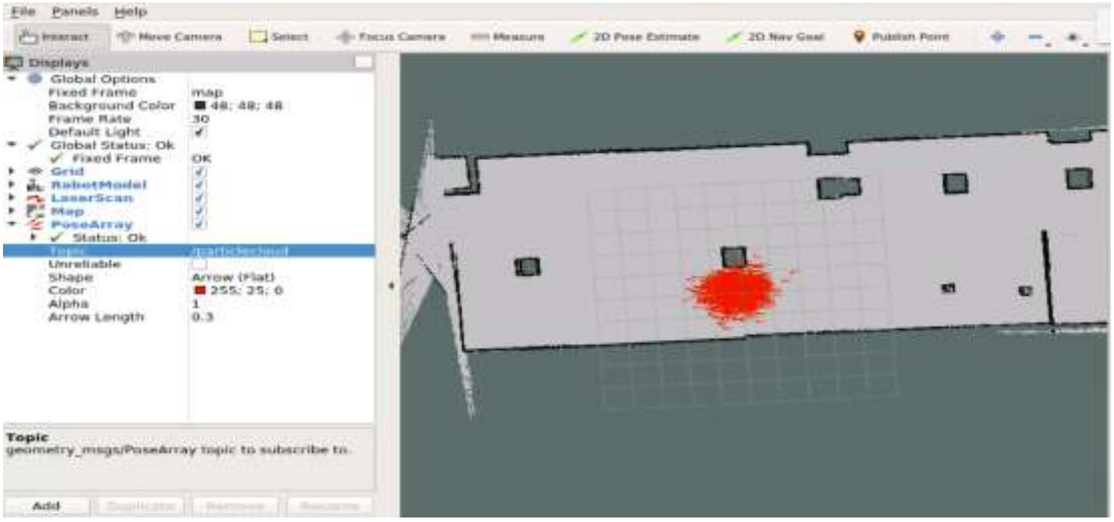


Figure 6

Then start moving the robot around the environment to localize the robot. As you move the robot, you will see in RViz how the particles keep getting closer, which means that the estimated poses of the robot are getting closer to the real place. This is a test on how good your localization system is working.

2.4 Path Planning and Obstacle Avoidance

We use the **move_base** node from the Navigation Stack, which will manage all the Path Planning system. To execute the launch file to start the navigation system.

roslaunch micro_project start_navigation.launch

Click the 2D Nav Goal button in Rviz menu to send a Goal(desired pose) to the Robot. Before we send a goal, the first thing we need to do is localizing the robot to provide an initial pose for the robot. Then we can see the Turtlebot3 robot going to that position in the simulation. In Rviz, you can also visualize the planned path that it follows.

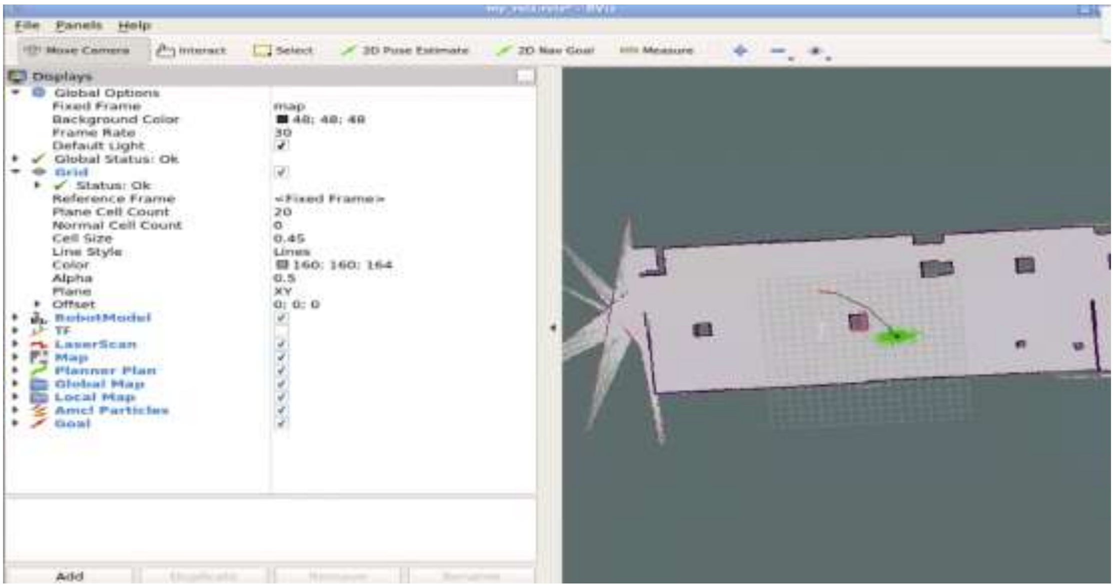


Figure 7

cp /home/simulations/public_sim_ws/src/all/turtlebot/turtlebot_navigation_gazebo/urdf/object.urdf /home/user/catkin_ws/src

rosrun gazebo_ros spawn_model -file /home/user/catkin_ws/src/object.urdf -urdf -x 0 -y 0 -z 1 -model my_object

Adding obstacles to Gazebo for Navigation Simulation

First, we copy the file *object_box.urdf* which is located in the folder /home/user/catkin_ws/src/model.

With the following command we will spawn an obstacle in the environment at the x, y, z position with the model name obstacle_01:

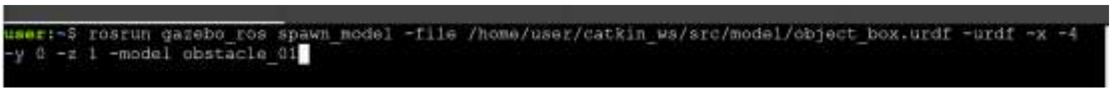


Figure 8

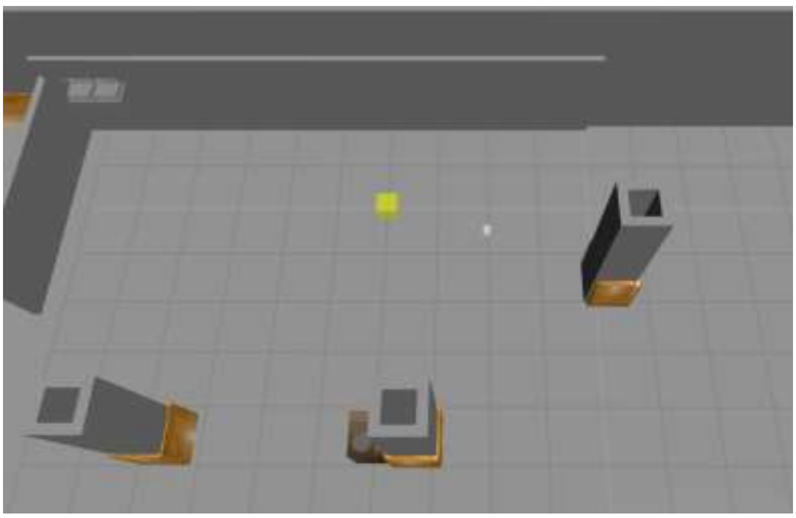


Figure 9

2.5 Add Waypoint

We use a ROS package called follow_waypoints, This package basically tracks the Estimate pose that we place in RVIZ and stores it. Then when we have finished our just have to publish in a topic that starts sending those positions to the movebase system.

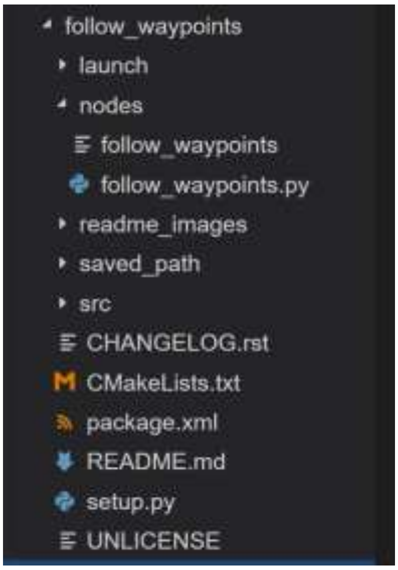


Figure 10

Now we have to set the waypoints. For this we have to select the PoseEstimate and set it where we want it. Because this will move the estimated pose of the robot around it is important that the last waypoint is where the robot is right now. Otherwise, it will be harder for the robot to get to the place

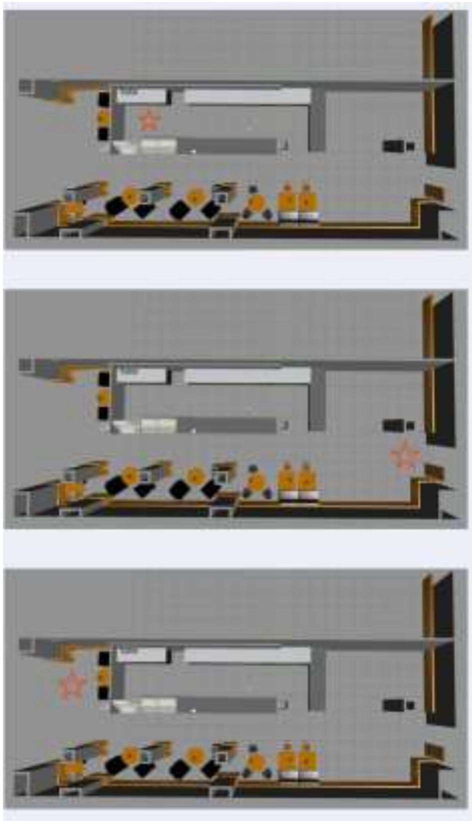
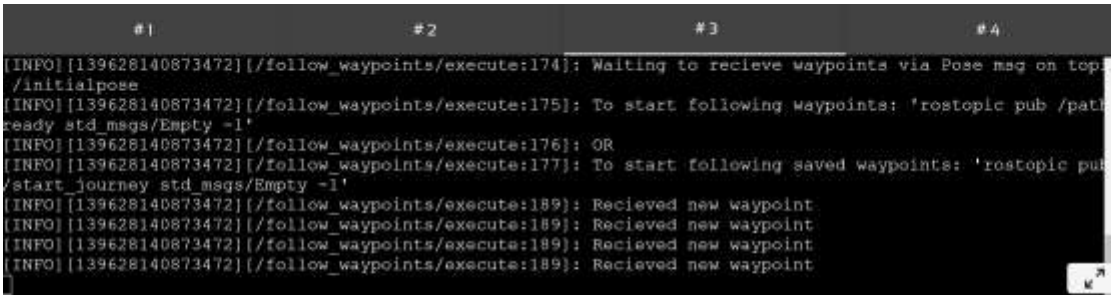


Figure 11

We should get something like this:

Figure 12

Then we get in webshell where we launched the waypoint server a message stating that it received the waypoint:



```
#1 #2 #3 #4
[INFO][139628140873472][/follow_waypoints/execute:174]: Waiting to recieve waypoints via Pose msg on top
/initialpose
[INFO][139628140873472][/follow_waypoints/execute:175]: To start following waypoints: 'rostopic pub /path
ready std_msgs/Empty -1'
[INFO][139628140873472][/follow_waypoints/execute:176]: OR
[INFO][139628140873472][/follow_waypoints/execute:177]: To start following saved waypoints: 'rostopic pub
/start_journey std_msgs/Empty -1'
[INFO][139628140873472][/follow_waypoints/execute:189]: Recieved new waypoint
[INFO][139628140873472][/follow_waypoints/execute:189]: Recieved new waypoint
[INFO][139628140873472][/follow_waypoints/execute:189]: Recieved new waypoint
[INFO][139628140873472][/follow_waypoints/execute:189]: Recieved new waypoint
```

Figure 13

The end we must publish in the topic /path_ready to start sending waypoints to the movebase.

rostopic pub /path_ready std_msgs/Empty -1

And we got the result in video.

3. Questions and Solve

3.1 Refers to the "old" Turtlebot2 :$ roslaunch turtlebot_teleop keyboard_teleop.launch. But for the Turtlebot3 the keyboard teleop is:
$ roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch

3.2 For the follow_waypoints package. We got it from the:
git clone https://github.com/danielsnider/follow_waypoints.git
but the code in ~/catkin_ws/src/follow_waypoints/nodes/follow_waypoints can't be able to locate the module follow_waypoints.
Solution: copy the
~/catkin_ws/src/follow_waypoints/src/follow_waypoint/follow_waypoint.py to
~/catkin_ws/src/follow_waypoints/nodes. And change the code "from follow_waypoints import follow_waypoints" to "import follow_waypoints"

3.3 For the mapping, the fist map we got is

Figure 14

For the start_mapping.launch . We change the most important parameters in these files are:
1.maxUrange(This parameter sets how far the laser will be considered to create the map. Greater range will create maps faster and its less probable that the robot gets lost. The downside it’s that consumes more resources).
2.map_update_interval(This parameter defines time between updating the map.
The smaller the value, the more frequent the map is updated. However, setting this too small will be require more processing power for the map calculation. Set this parameter depending on the map environment)
3.linearUpdata(When the robot translates longer distance than this value, it will run the scan process)
4.angularUpdata(When the robot rotates more than this value, it will run the scan process. It is recommended to set this value less than linearUpdate.)
Solution:
Set the "maxUrange" value="16.0"; “map_update_interval" value="5.0”.
“linearUpdate" value="1.0"; "angularUpdate" value="0.5"

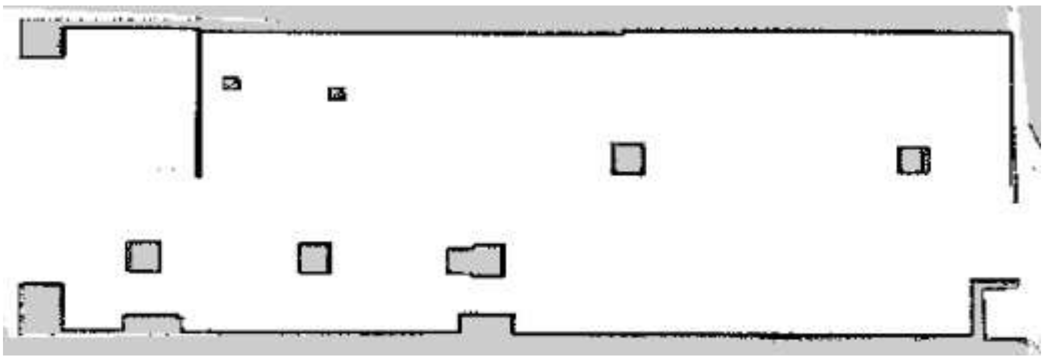


Figure 5

4. Conclusion

In this project, we had created a map of the environment using 'gmapping' package and used 'amcl' package to navigate autonomously in the map. We had developed various nodes to achieve our task of detection and localization of markers with respect to map and make the turtlebot to go to localized positions of markers to full fill the task of pick and place of objects.