UNIVERSITAT DE GIRONA

AUTONOMOUS ROBOTICS

LABORATORY REPORT

Mapping, planning and controlling with the Turtlebot

Author
Mohit VAISHNAV
Malav BATERIWALA

Supervisor
Dr. Perez Marc CARRERAS
Tutor
Eduard Vidal GARCIA

Submission Last Date 15th, April

1 Introduction

The autonomous navigation of robots in uncontrolled environments is a one of the biggest challenge currently because it requires many systems work together to navigate. It requires building a map of the environment, localizing the robot in that map, making a motion plan according to the map, executing that plan with a controller, and other tasks and that also all at the same time. We know that this is one of the most important problem in the world of autonomous navigation, because all the position for further navigation dependent upon where the robot or AUV lies. There are many applications that require this problem to be solved, such as package delivery, cleaning, agriculture, surveillance, search & rescue, construction and transportation. In this lab, we applied to navigate a turtleBot and created a map of an unknown environment and then applied different algorithm to plan the path in that environment.

2 Mapping

2.1 Dependencies

We require particular dependencies for every different type of system to run a particular AUV and the backend system. So for that in this lab we install (*Octomap for Kinetic Version complete*) , which will install all the required packages for creating a map , which are missing in the system. So in linux we use the following line to install:

$$sudo \quad apt-get \quad install \quad ros-kinetic-turtlebot \quad octomap$$

Even we are given the link to pull it from the GIT folder online. We have to clone this repository into the catkin folder. We also need to install the same dependencies in the turtlebot , so we have to install the same version as we installed in the back-end system.

2.2 Checking Vehicle Odometry in Simulation and turtlebot

We do this to check if the system is running perfectly and we can move the turtlebot in the simulation and as well as the real model. So for running it in the simulation, we do that in gazebo and we use teleoperation of the turtlebot of simulation to run the turtlebot in the gazebo framework. For all these we use the command:

```
roslaunch control_turtlebot start_turtlebot_modules.launch
```

This launches all the required files that is required for movement of the tutlebot. Now we have to do that for the real model. For that we use the same command , but in the turtlebot framework also. So for this we have to connect to the same network as the turtlebot and make sure that we have started the above command in both the terminals. So we can check the motion in both the frameworks by using this command:

```
rostopic echo /odom
```

This provides the continuous print of the current odometry of the turtlebot and also if we running the same in gazebo.

2.3 Building the Map

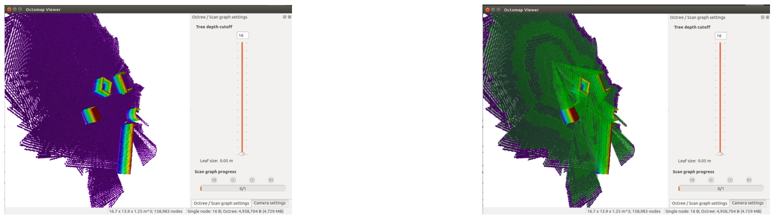
Now as we can navigate the turtlebot in an unknown terrain we will now create a map of that unknown terrain. For this we shall use the sensor that is present in the turtlebot of the Microsoft called as Kinect. This will give the data of all the the obstacles present in its field of view. Thus by using this input we can create a map of the surrounding. So we take this input and use RViz, which is a visualizer available for ROS and then use this information to create a database of all things we record. We use this commands to start mapping the system:

```
roslaunch control_turtlebot start_turtlebot_mapping_control.launch
```

In the new terminal we will start saving the map using this command:

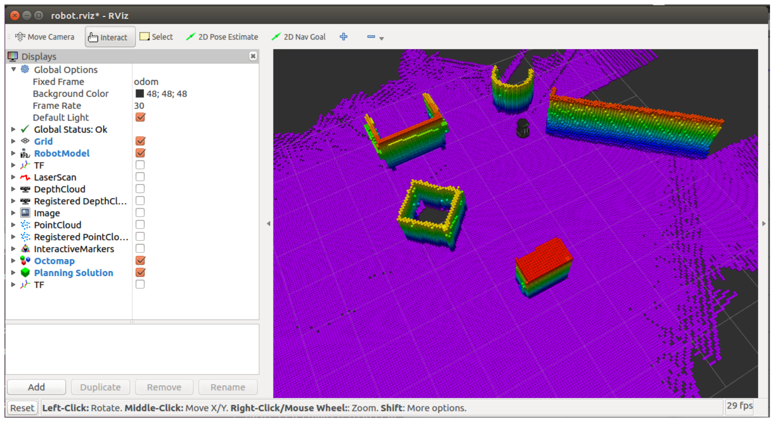
```
rosrun octomap_server octomap_saver map.bt
```

This will start the connection between the gazebo or the teloperation modules and RViz and it will later save the map we are creating. So when we move around the unknown terrain using the teleoperation commands from the system, we record all the obstacles present. Thus by doing this we create a map of the place and we know where we have seen an obstacle , this all will be mapped in the visualizer RViz as it is also running at the same time. Thus , by doing this motion around the given unknown place, we can create a map of that place. This is known as Mapping, after mapping we can then move it further for planning the path of the turtlebot or AUV.



(a) Gray Image 1 (b) Thresholded Image

Figure 1: Input Image and Output for Image 1

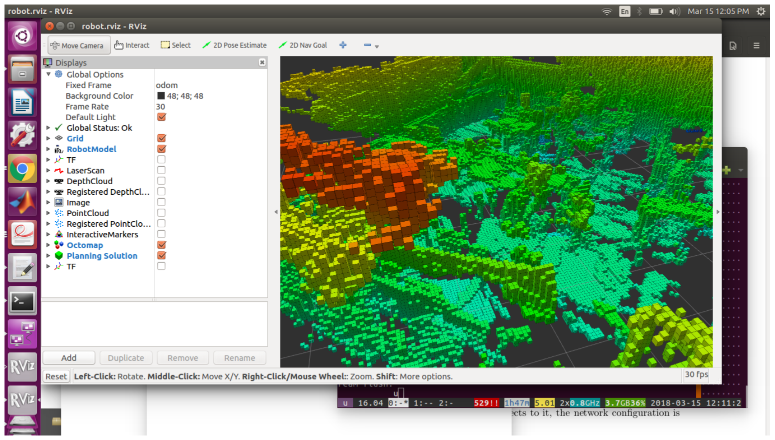


Global Options	
Fixed Frame	odom
Background Color	■ 48; 48; 48
Frame Rate	30
Default Light	✓
Global Status: Ok	
Grid	✓
RobotModel	✓
TF	☐
LaserScan	☐
DepthCloud	☐
Registered DepthCl...	☐
Image	☐
PointCloud	☐
Registered PointClo...	☐
InteractiveMarkers	☐
Octomap	✓
Planning Solution	✓
TF	☐

Figure 2: Rviz visualization of the environment

2.4 Understanding the * map

As you can see in the Figure 2 , that is the basic representation on the map we get in RViz. It shows the obstacle in different colors, and the ground as purple. As the size of the obstacle changes, also the color changes in their representation of the objects, size is less the color will only be blue. But if the size is bigger you can see in the figure that there will be many multilayes available. This is done by kinect , during the mapping of the environment.
We can even perform various operations on the build up map, so that we can perform planning better and also it can help us on deciding which type of algorithm to use for path planing.



Global Options	
Fixed Frame	odom
Background Color	■ 48; 48; 48
Frame Rate	30
Default Light	✓
Global Status: Ok	
Grid	✓
RobotModel	✓
TF	☐
LaserScan	☐
DepthCloud	☐
Registered DepthCl...	☐
Image	☐
PointCloud	☐
Registered PointClo...	☐
InteractiveMarkers	☐
Octomap	✓
Planning Solution	✓
TF	☐

Figure 3: Rviz visualization of the map building

3 Planning

3.1 Dependencies

To make this lab run smoothly we have to first install the dependencies with the *ROS* Kinetic version along with the ones we already have in *mapping* part (*Octamaps, octovis, etc.*). Following command line in *linux* terminal installs all such things:

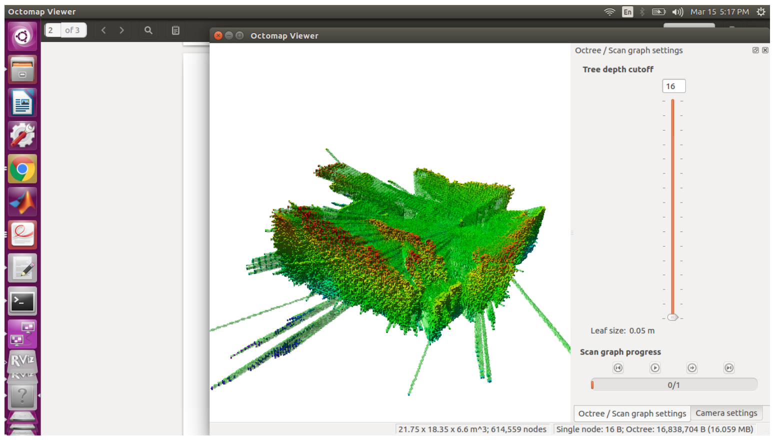
$$sudo \quad apt-get \quad install \quad ros-kinetic-ompl$$

Now, inside the source (*src*) folder of Catkin workspace (*/catkin_ws/*) we have to clone the mapping and planning tutorial (if already present, pull using *git pull*). Once we have all the files in the workspace next step is to compile it using *catkin_make*.

3.2 Checking vehicle capability for building simple map

We used a simplified version of the maps ie. we used 2D map instead of using a full 3-Dimensional (3D) maps. *Octomap-server* which takes use of laser scan of the Kinetic Camera to implement the desired thing instead complete point cloud. To move any further, we checked if the launch files (*start_turtlebot_modules.launch*) are available or not. In another terminal we have to start turtlebot simulator and teleoperation commands

$$roslaunch \quad control\_turtlebot \quad start\_turtlebot\_modules.launch$$



Tree depth cutoff 16

Leaf size: 0.05 m

```
21.75 x 18.35 x 6.6 m^3; 614,559 nodes
Single node: 16 B; Octree: 16,838,704 B (16.059 MB)
```

Figure 4: Rviz visualization of the map building with different view

In second terminal we have to execute the mapping, planning and control system which is done like this:

roslaunch control_turtlebot start_turtlebot_mapping_planning_control.launch

This ensures that we have all the environmental setup and dependencies ready. Our task was to then create a map by moving the turtlebot around the arena using keys *j*, *l* and other respective keys. While moving the turtlebot, observe the same in *Gazebo* simulated environment. As the turtlebot moves, this movement can be traced in it. Now comes into picture the role of *octomap_saver* which saves the map and *octovis* which is used to visualize and compare the 2D maps with 3D maps obtained earlier.

3.3 Planning a collision free path using Octomap

Open Motion Planning Library is used to plan a collision free path. It contains various sampling based algorithms like *RRT*, *RRT∗*, *PRM* and *EST*. To change the configuration space and tell the program to run the following algorithm we have to modify the planner parameters file present in control_turtlebot

∼ /catkin_ws/src/mapping_planning_tutorial/control_turtlebot/launch/planner_parameters.yaml

This planning node finds the collision free path from current position to the goal position conditioned that it should be within the bound of the previous files defined in offline planner as shown below.

$$\sim /catkin\_ws/src/mapping_{p}lanning_{t}utorial/ \\ control\_turtlebot/src/offline\_planner\_R2.cpp$$

Now we have to plan the path where robot moves with out collision which are defined in one of the file contained in the folder find path to goal.

$$rosservice \quad call \quad /controller\_turtlebot/find\_path\_to\_goal \\ \#then \quad press \quad tab \quad twice \quad to \quad complete$$

On double tapping the *tab* button we have an option to enter the goal satate in x and y direction.

3.4 Mapping the environment, planning a path and following the path

Once we have the path estimated, now the task remains is to traverse the path. In this we takes into consideration that we already have the environment fully mapped and the environment is static as it has already saved the path and obstacles which defined the path. Otherwise there will be collisions if the obstacles starts moving all around. To verify if the turtlebot is capable of moving in the path by using the file control turtlebot in folder with same name.

$$\sim /catkin\_ws/src/mapping\_planning\_tutorial/ \\ control\_turtlebot/src/controller\_turtlebot.py$$

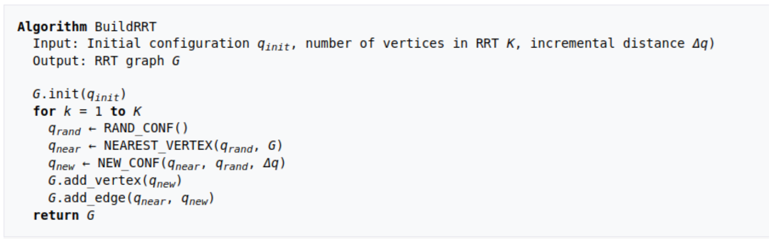
Again in a new terminal we have to give command to make the turtlebot move to the given path:

$$rosservice \quad call \quad /controller\_turtlebot/goto \\ \#then \quad press \quad tab \quad twice \quad to \quad complete$$

4 Results and Output

We have to do the planning and mapping of the bot in two types of environment, simulated version and the real scenario.

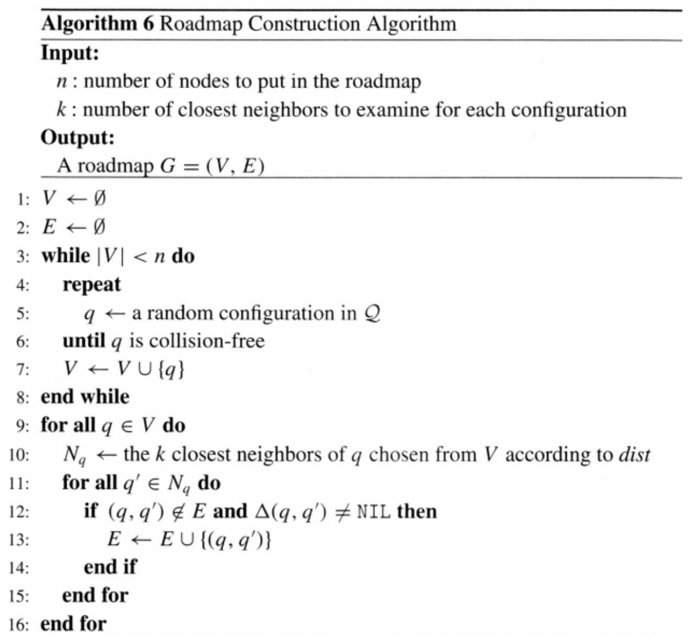
In Fig. 7, we have demonstrated the path planning using two different techniques like *RRT* and *PRM* amongst others. Their pseudo code has been elaborated below in Fig. 5 for RRT and in Fig. 6 for PRM. After generating the path their movement have been shown in Fig. 8 and it shows how the bot moves exactly in the same path which has been obtained.



```
Algorithm BuildRRT
  Input: Initial configuration $q_{init}$, number of vertices in RRT $K$, incremental distance $\Delta q$)
  Output: RRT graph $G$

  $G$.init($q_{init}$)
  for $k$ = 1 to $K$
    $q_{rand}$ ← RAND_CONF()
    $q_{near}$ ← NEAREST_VERTEX($q_{rand}$, $G$)
    $q_{new}$ ← NEW_CONF($q_{near}$, $q_{rand}$, $\Delta q$)
    $G$.add_vertex($q_{new}$)
    $G$.add_edge($q_{near}$, $q_{new}$)
  return $G$
```

Figure 5: RRT Algorithm

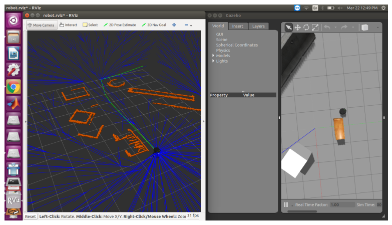


```
Algorithm 6 Roadmap Construction Algorithm
Input:
  $n$ : number of nodes to put in the roadmap
  $k$ : number of closest neighbors to examine for each configuration
Output:
  A roadmap $G = (V, E)$

$V \leftarrow \emptyset$
$E \leftarrow \emptyset$
while $|V| < n$ do
  repeat
    $q \leftarrow$ a random configuration in $\mathcal{Q}$
  until $q$ is collision-free
  $V \leftarrow V \cup \{q\}$
end while
for all $q \in V$ do
  $N_q \leftarrow$ the $k$ closest neighbors of $q$ chosen from $V$ according to $dist$
  for all $q' \in N_q$ do
    if $(q, q') \notin E$ and $\Delta(q, q') \neq$ NIL then
      $E \leftarrow E \cup \{(q, q')\}$
    end if
  end for
end for
```

Figure 6: PRM Algorithm

The same has also been implemented in the real world scenario in the lab environment where the arena has been designed for this purpose. Results have been demonstrated in Fig. 9.


(a) Path Finding using RRT

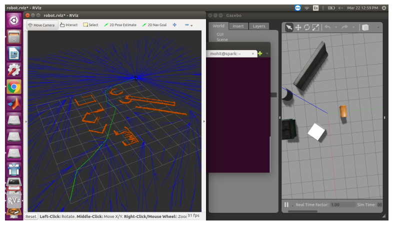
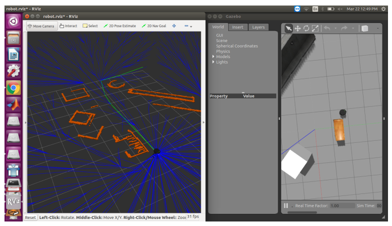

(b) Path Planning using PRM

Figure 7: Path planning using two different technique


(a) Turtlebot at Initial Position

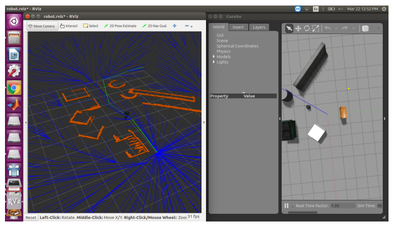
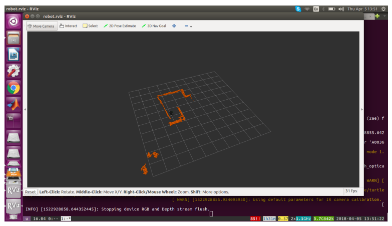

(b) Turtlebot moving along the path

Figure 8: Movement of path along the path


(a) Mapping in real scenario

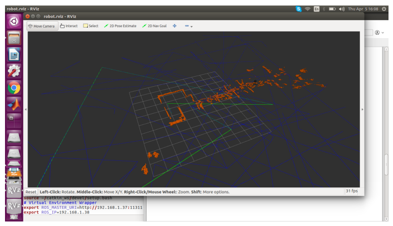

(b) Path planning for the desired location

Figure 9: Movement of path along the path in real scenario

5 Difficulties

We have faced a lot of difficulty in connecting the laptop to the turtlebot where we have tried all the combination like making a hotspot in host laptop, user laptop, connecting via wifi router and changing the laptop. Another problem was carrying on the same procedure as soon as the lab starts and it keeps on going till next two hours because of which we were always lagging behind in completing the lab in those stipulated session. Although after hours of efforts of instructor and lab mates that was made possible. Even during the last lab, we tried planning the path for the turtlebot which was taking the initial coordinate as the position where we first kept the bot instead the position in arena.

References

[1] ROS Website,
`http://www.ros.org/about-ros/`

[2] ROS Mapping and Planning,
`http://wiki.ros.org/Robots/TIAGo/Tutorials/Navigation/Mapping`

[3] ROS Package,
`http://wiki.ros.org/Packages`

[4] PDF link,
`https://web.wpi.edu/Pubs/E-project/Available/E-project-012115-103918/unrestricted/Motion_Planning_of_Intelligent_Robots.pdf`