

# ROS EXERCISES

## Sarim Mehdi & Zara Torabi

### MOTION PLANNING

Given a sequence of points and locations of obstacles, we use the potential field algorithm to move from one point to the next. The obstacles provide repulsive potential while points on the trajectory acts as a source of attraction for the robot.  $K_a$ ,  $K_r$  and  $q_{star}$  are kept as 0.2, 0.05 and 0.5 respectively.

#### Attractive potential

$$U_{att}(q)$$

- It has the minimum in  $q_{goal}$
- Its job is to attract the robot to the goal

#### Repulsive potential

$$U_{rep} = \sum_{i=1}^{N_{obst}} U_{rep_i}(q)$$

- Sum of the repulsive potentials for each obstacle
- It takes the robot away from obstacles

#### Control law

$$\dot{q}(t) = -\nabla U(q), \quad \text{dove: } U(q) = U_{att}(q) + U_{rep}(q)$$

$$U_{att} = \frac{1}{2} K_a \Delta(q, q_{goal}) \quad \nabla U_{att} = \frac{1}{2} K_a \nabla \Delta^2(q, q_{goal}) = K_a (q - q_{goal})$$

$$U_{rep_i}(q) = \begin{cases} \frac{1}{2} K_{r_i} \left( \frac{1}{d_i(q)} - \frac{1}{q^*} \right)^2 & d_i(q) \leq q^* \\ 0 & d_i(q) > q^* \end{cases}$$

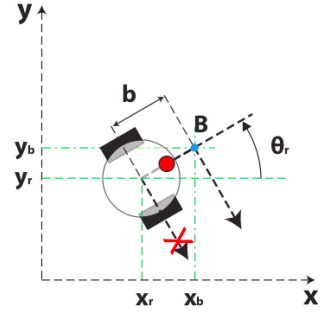
$$\nabla U_{rep_i}(q) = \begin{cases} K_{r_i} \left( \frac{1}{q^*} - \frac{1}{d_i(q)} \right)^2 \nabla d_i(q) & d_i(q) \leq q^* \\ 0 & d_i(q) > q^* \end{cases}$$

$$d_i(q) = \sqrt{(q - q_{obs_i})^2}$$

$$q = [x_b, y_b]^T$$

From above, we obtain the rate of change along the x and y direction. The relationship between the robot linear speed and angular speed is given as:

$$\begin{bmatrix} \dot{x}_b \\ \dot{y}_b \end{bmatrix} = \begin{bmatrix} \cos \theta_r & -b \sin \theta_r \\ \sin \theta_r & b \cos \theta_r \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} = T(b, \theta_r) \begin{bmatrix} v \\ \omega \end{bmatrix}$$



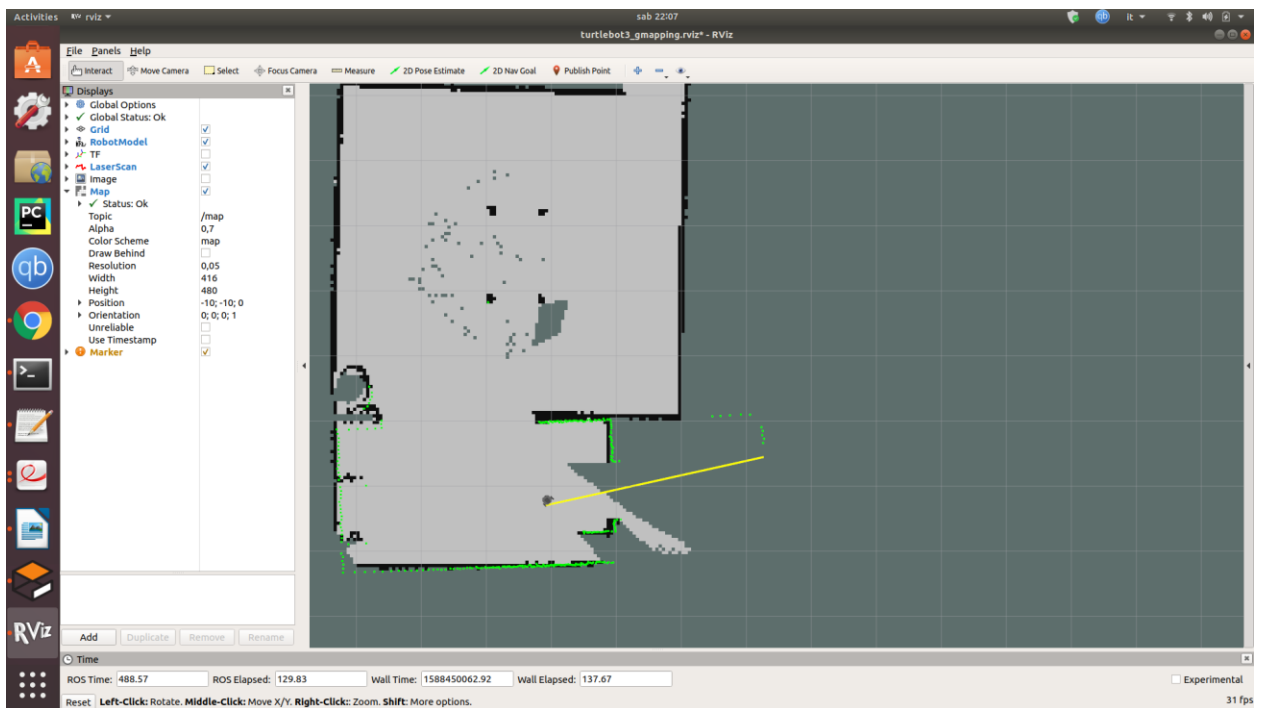
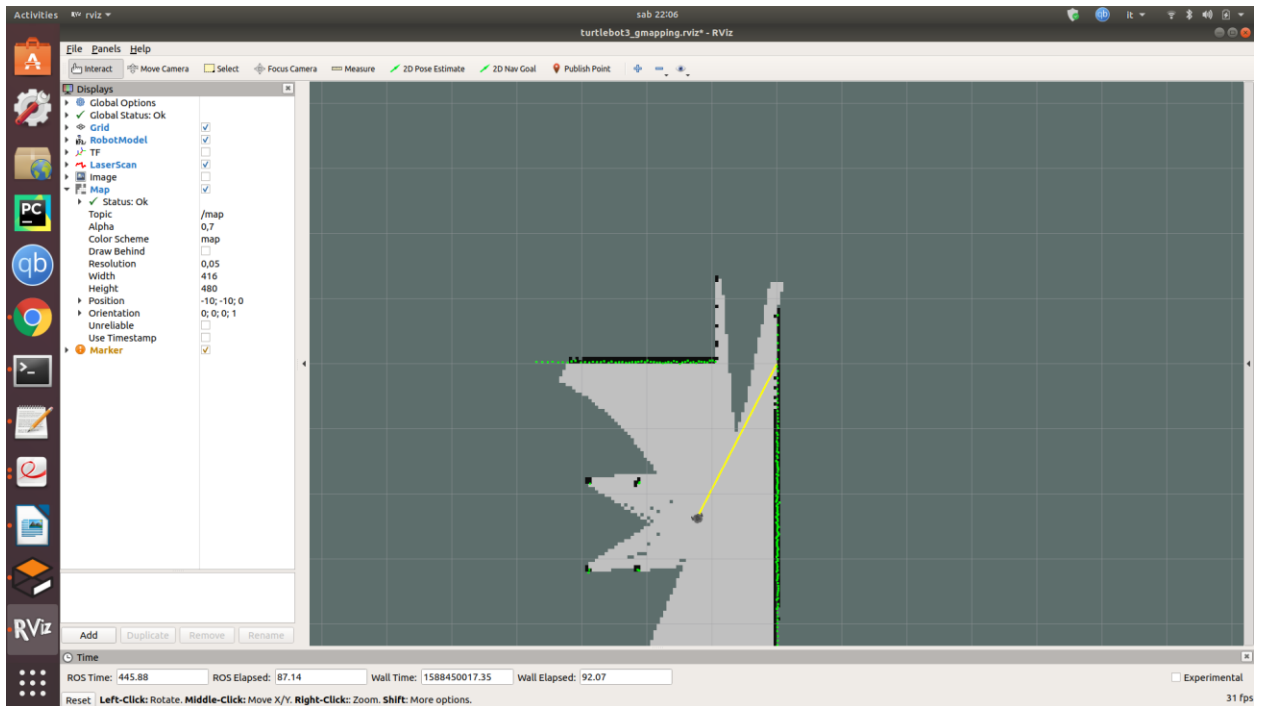
$$\begin{bmatrix} v \\ \omega \end{bmatrix} = T(b, \theta_r)^{-1} \begin{bmatrix} \dot{x}_b \\ \dot{y}_b \end{bmatrix} = \begin{bmatrix} \cos \theta_r & \sin \theta_r \\ -\frac{1}{b} \sin \theta_r & \frac{1}{b} \cos \theta_r \end{bmatrix} \begin{bmatrix} \dot{x}_b \\ \dot{y}_b \end{bmatrix}$$

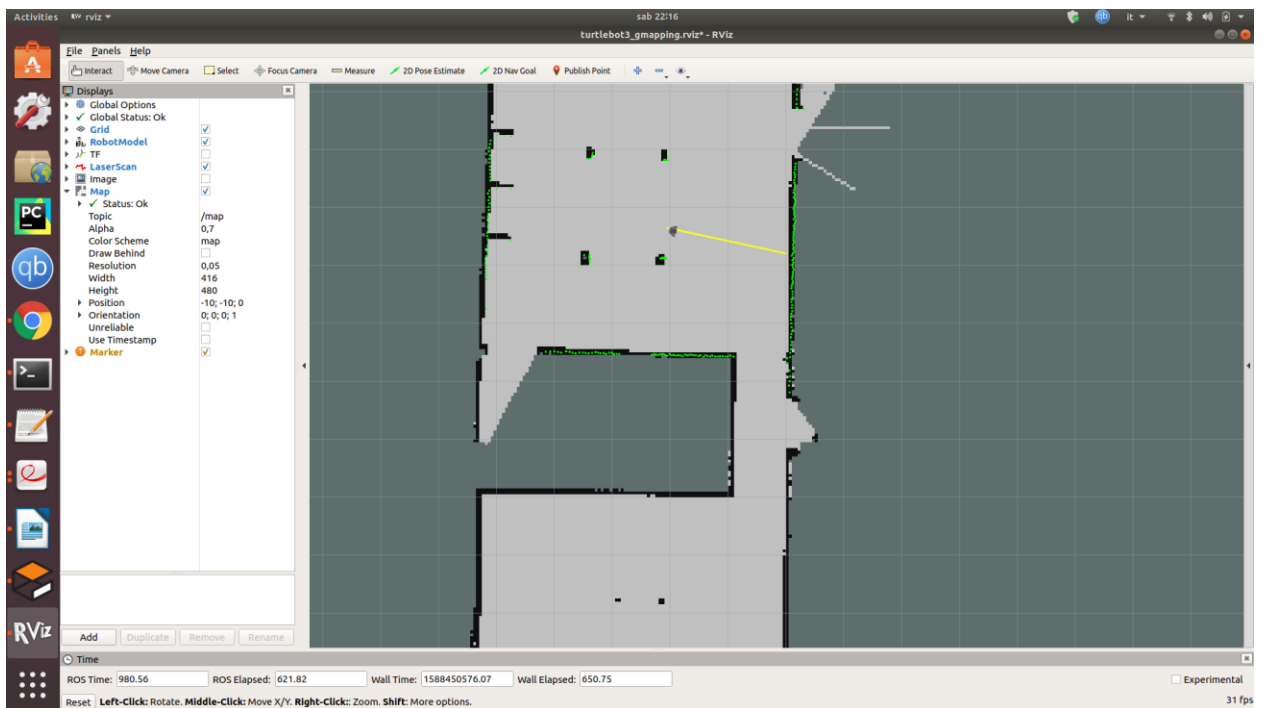
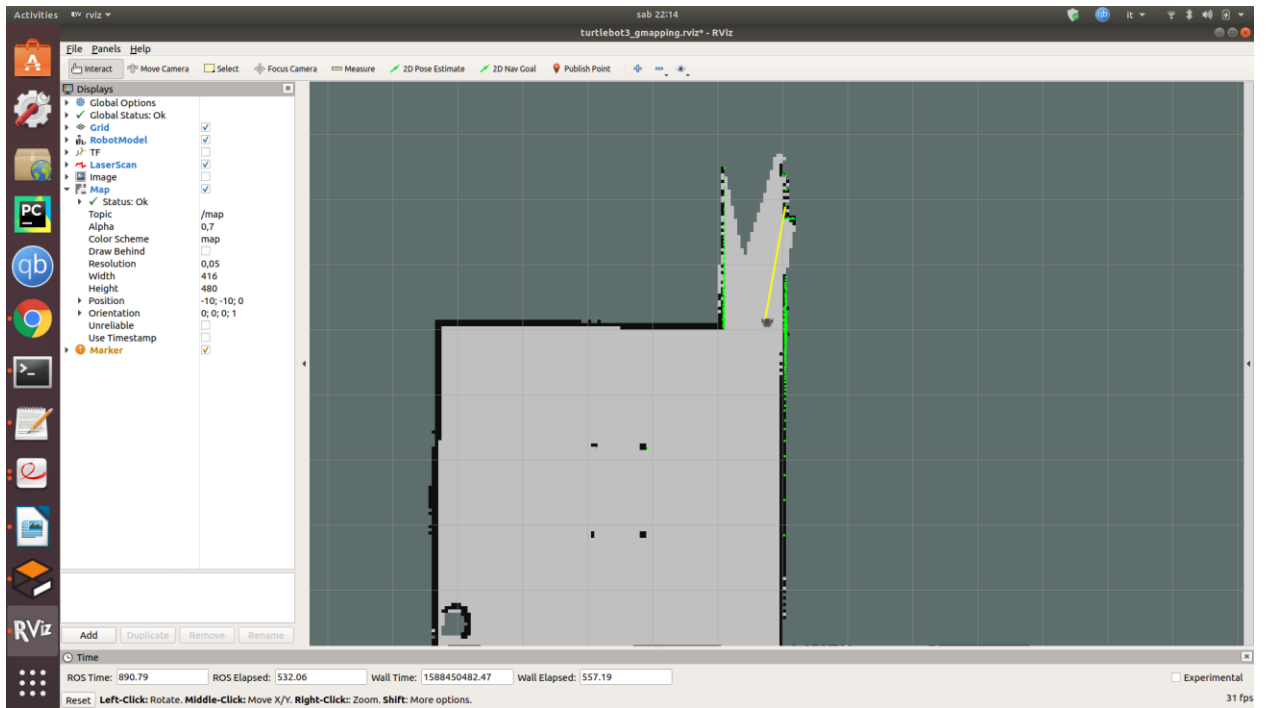
Where B is an imaginary point that is always ahead of the robot center and this imaginary point is used to define the position of the robot. In the simulation, we assumed it to be 0.1 meters. v and w are directly fed to the robot as twist.linear.x and twist.angular.z

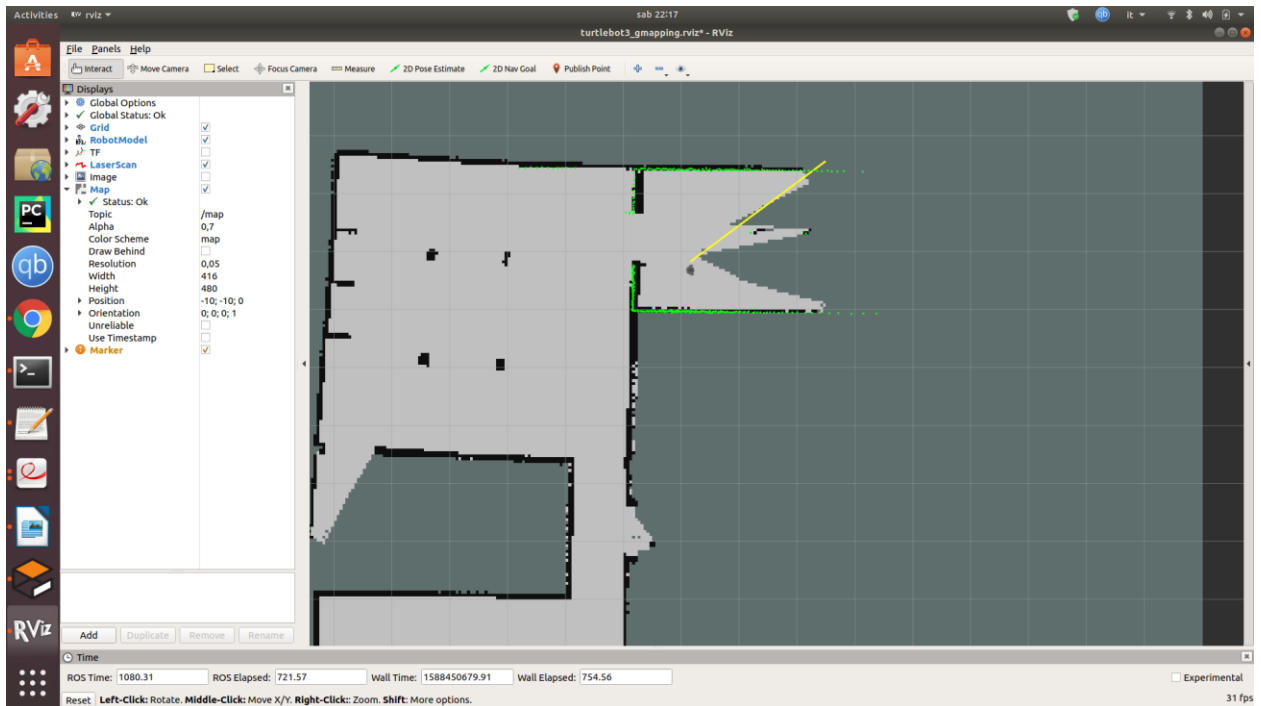
# MAPPING

We make use of the odometry topic to know the position of the robot during mapping. Laser Scanner data can then be converted into position values in the reference frame of the map. This is important as we follow along obstacles in the house. Our method is a slightly modified version of the bug planning algorithm:

- **STAGE 1:** The robot looks for the shortest obstacle right in front of it and moves towards it. Once it reaches the obstacle, it looks for the next one and, in this way, it moves along walls like in a typical bug planning algorithm.
- **STAGE 2:** To prevent the robot from revisiting recent points, we keep a buffer of 60 points. The robot is not allowed to travel to any point that is closer to a point in this buffer. However, as we observed during simulations, after robot goes around a room it might get stuck as it would be unable to find any suitable point that fits the mentioned criteria. To deal with this, we let the robot look for the farthest obstacle instead of the closest one based on the laser scanner data. This allows the robot to move into previously unexplored territory and even allows it to enter a new room.
- **STAGE 3:** If the robot is unable to find a point in Stage 2 and Stage 3 (the farthest point in Stage 3 must also fit the criteria of not being too close to any previously explored points), then we tell the robot to pick a random point from the last 5 visited points. We observed that usually we encounter this situation when the robot is stuck in a corner of a small room and having it do a simple backtrack would allow it to get out easily.

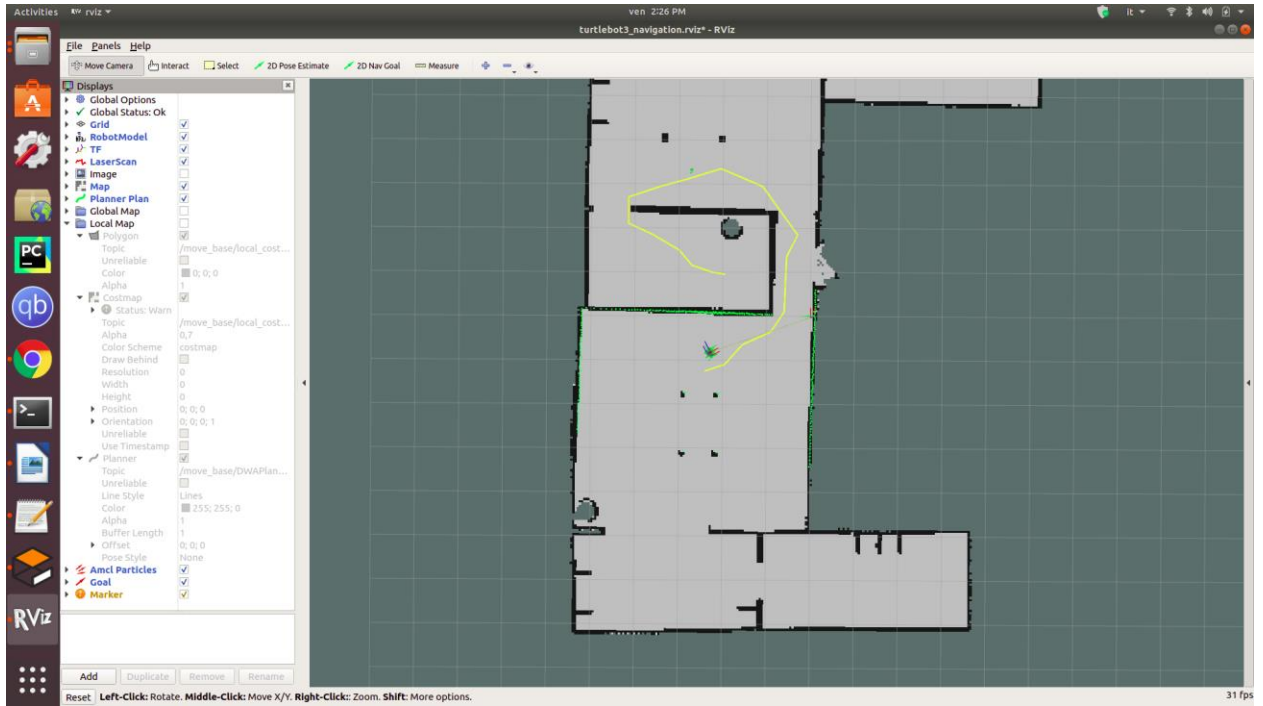
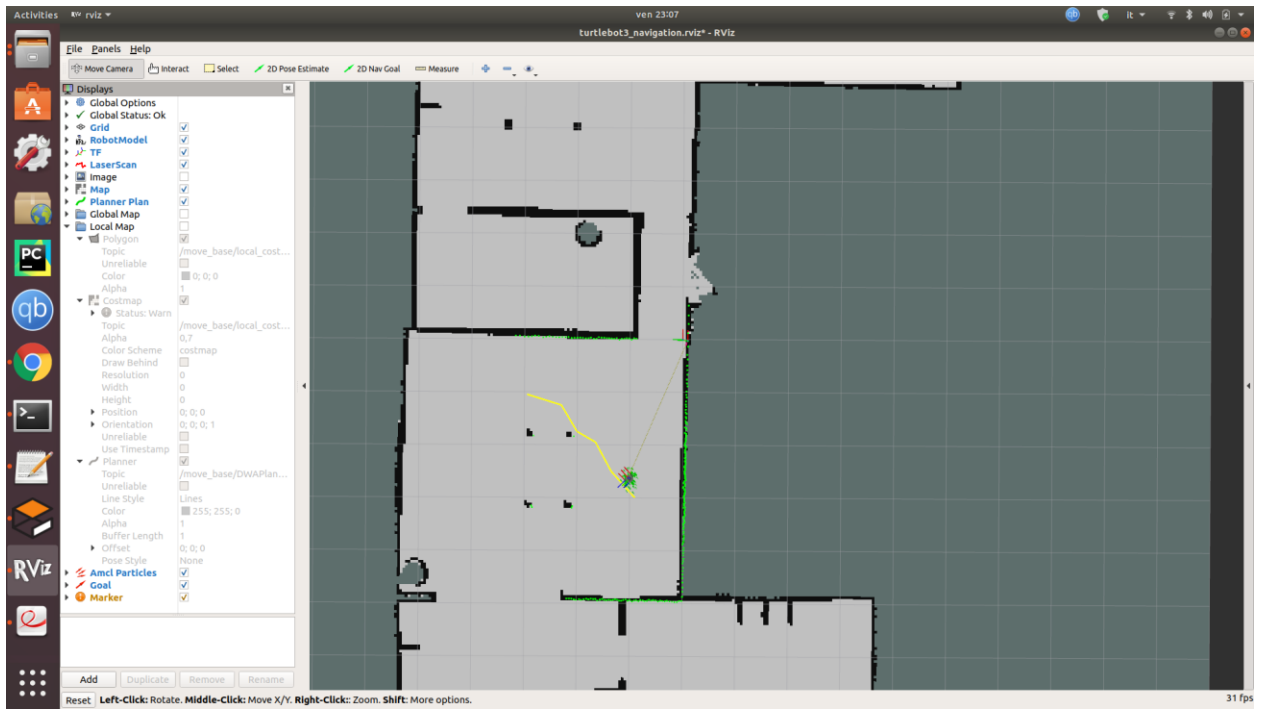


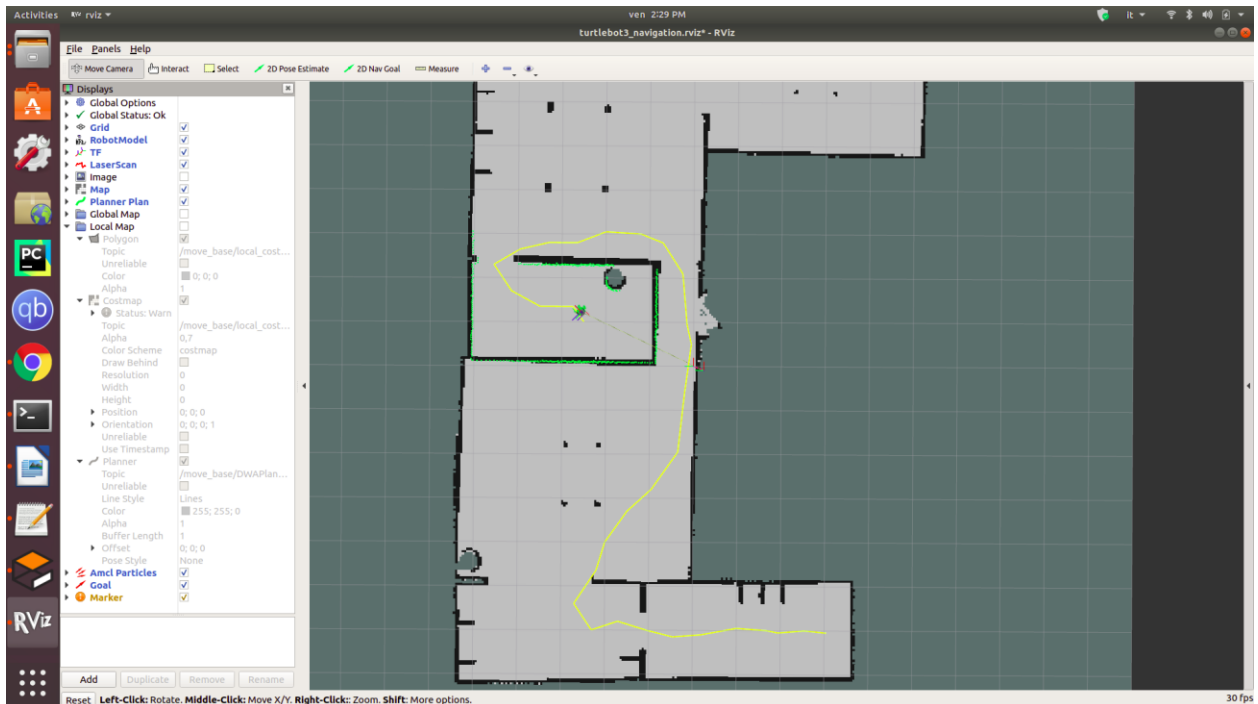




## NAVIGATION

For path planning, we use the Probabilistic Road Map technique. An open source implementation of PRM was already available in Python. We simply adjusted some parameters to get the desirable results. Up to 500 points are initially sampled. For each sampled point, up to a maximum of 10 neighbors are selected. Neighbors of a sampled point are at 0.3 cm or less. If there are more than 10 eligible neighbors, the 10 best neighbors are chosen. Among the neighbors, a sampled point is chosen at random and the same criteria is used for choosing its neighbors. Edges between two nodes must lie in a free space else the two nodes are not neighbors. The sampling phase ends if a node close to the goal is discovered or no node close to the goal is discovered. Dijkstra's Algorithm is applied on the resulting graph to find the shortest path from the starting position of the robot till the goal. Each point can have at most 10 edges leading to other points and the maximum allowable length is 30 cm. In case of failure, the algorithm has a maximum of 10 tries before it gives up. In our experiments, we never had more than 5 failed attempts.

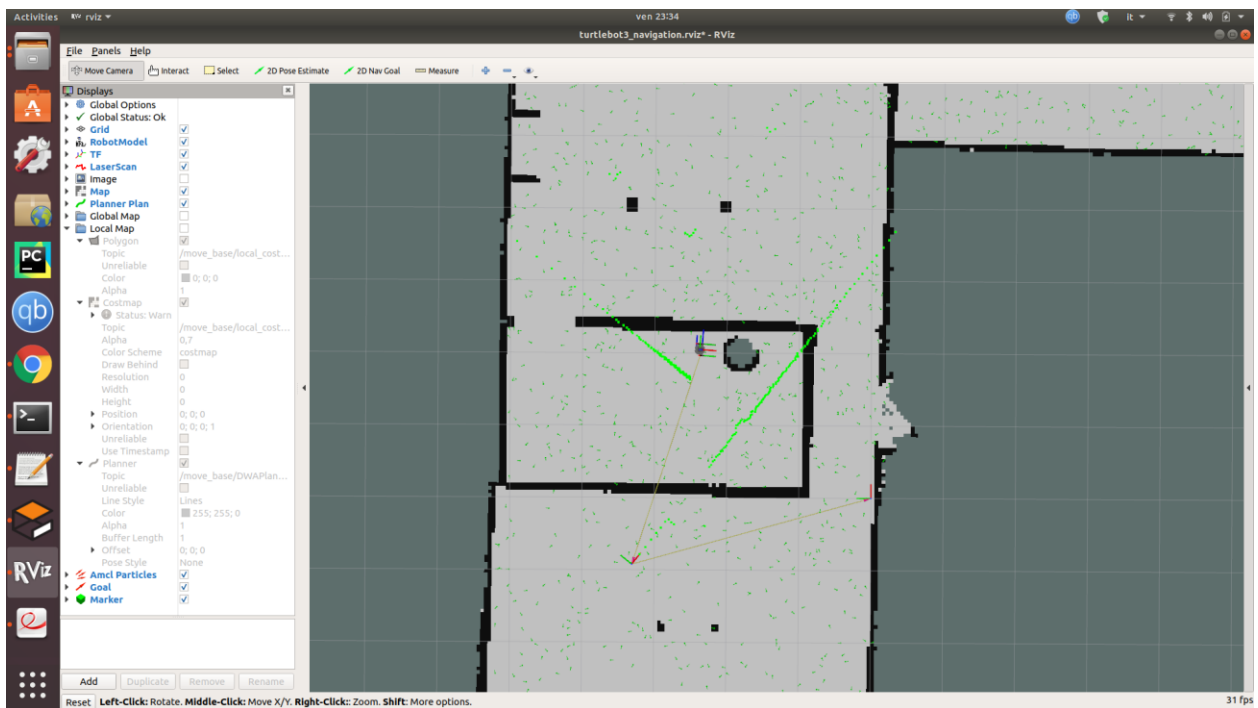
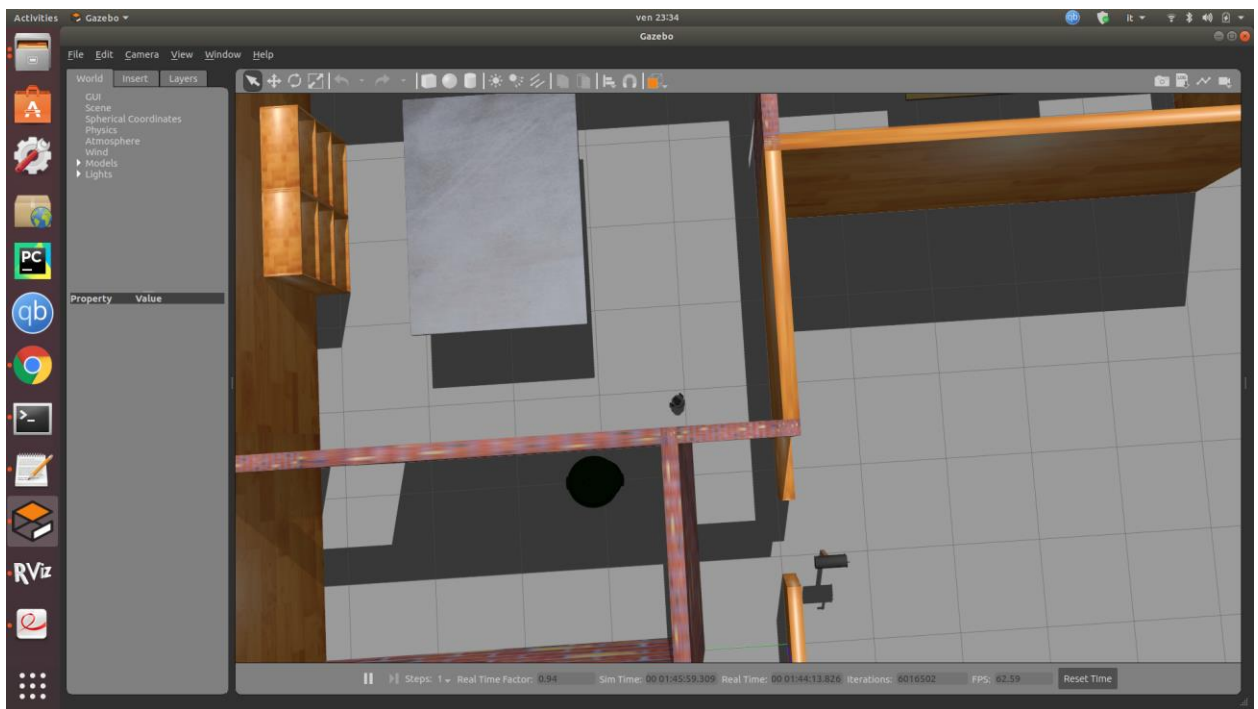


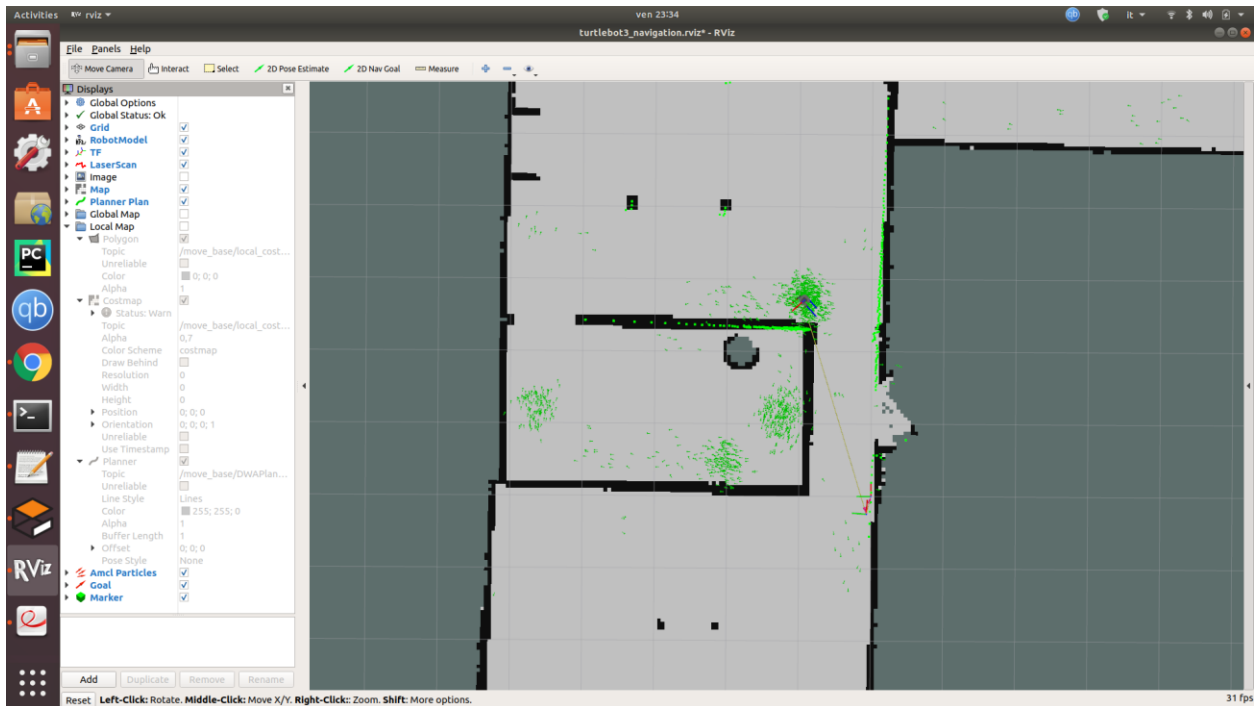


## LOCALIZATION & OBSTACLE AVOIDANCE

We use AMCL to do initial position localization. Particles are spread all over the map and the robot is spun around for 60 seconds. During this time, the laser scanner data is used to compute the innovation as the algorithm checks which particles correspond more accurately to the type of laser scanner data that is received by the robot. In almost every case, the robot correctly localizes itself to the correct initial position.







For obstacle avoidance, we simply tell the robot to turn slightly in the opposite direction. This works for both mapped obstacles and dynamic obstacles which were not present during mapping but are introduced during navigation.

Mapping: <https://youtu.be/IbWyAXOiKx0>

Navigation:

<https://www.youtube.com/watch?v=DPQBtdBvOTo&t=2m7s>

## HOW TO RUN MAPPING

```
roslaunch turtlebot3_gazebo turtlebot3_house.launch
```

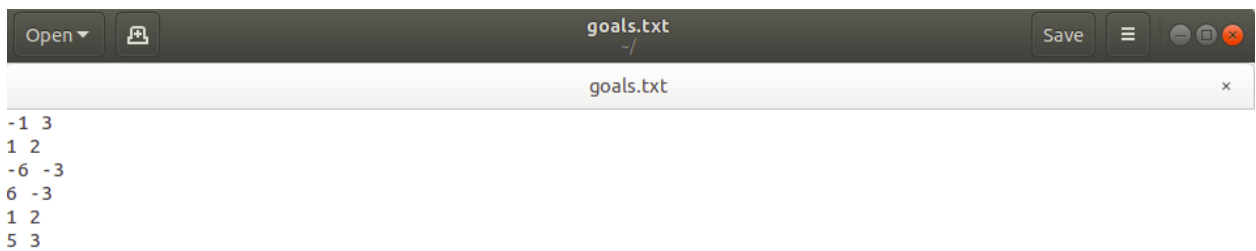
```
roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
```

```
roslaunch turtlebot3_slam turtlebot3_slam.launch slam_methods:=gmapping
```

```
roslaunch ros_project mapping.py
```

## HOW TO RUN NAVIGATION

Your goal file (goals.txt) must be in the following format:



```
Open [icon] goals.txt Save [icon] [icon] [icon] [icon]
goals.txt x
-1 3
1 2
-6 -3
6 -3
1 2
5 3
```

Plain Text ▾ Tab Width: 4 ▾ Ln 1, Col 1 ▾ INS

goals.txt must be in the catkin\_ws folder, map.pgm and map.yaml must be in \$HOME folder

```
roslaunch turtlebot3_gazebo turtlebot3_house.launch
```

```
roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
```

```
roslaunch turtlebot3_navigation turtlebot3_navigation.launch
```

```
map_file:=$HOME/map.yaml
```

```
rosservice call /global_localization "{}"
```

```
roslaunch ros_project navigation.py
```