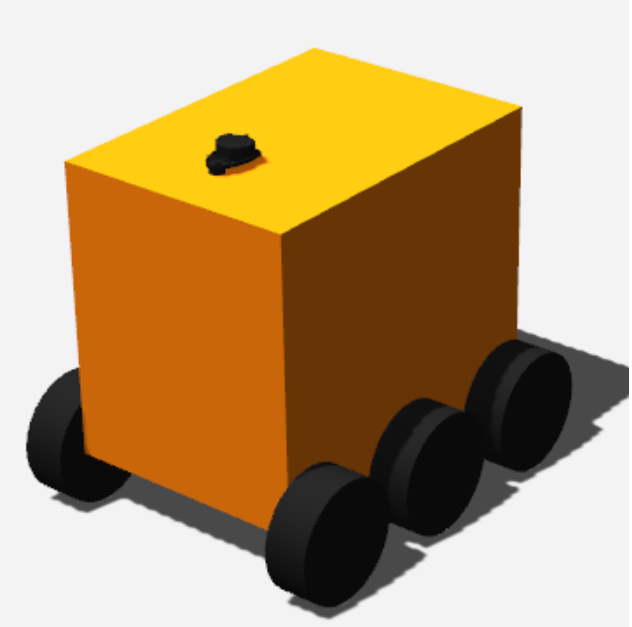
POLYTECHNIC OF BARI
Department of Electrical and Information Engineering

Automation Engineering Master's Degree

Course on Mobile Robotics

DelyBot



Professors:
Engr. Luca De Cicco
Engr. Carlo Croce

Students:
Alessandro Quatela
Giuseppe Roberto

Academic Year 2021/2022

ABSTRACT

The DelyBot project is presented in this report. The process that led to the simulation of the robot will be described in detail. At first, using the urdf language, a 3D prototype of the robot was modeled and simulated in Gazebo. Then, it was chosen to implement a differential drive control to the robot. Next, laser and imu sensors were added to allow the robot to localize itself. Several static 3D worlds were made in Gazebo and mapped to a 2D map using the gmapping algorithm. The last part of the work focused on the implementation of navigation and the automation of waypoint handling by a specific spawner node. The code files are available on GitHub at the following link: https://github.com/gmeidk/DelyBot .

Contents

1 Introduction

The goal of this project is to simulate a delivery robot, i.e., a robot capable of making deliveries in an urban environment.

The DelyBot is composed of an upper structure designed for transportation and six motorized wheels that allow it to move with differential drive control. The robot can locate itself through knowledge of the global static map and is able to detect and avoid moving obstacles.

The project is presented explaining the various packages implemented:

- delybot_description
- delybot_control
- delybot_slam
- delybot_navigation

2 DelyBot Description

The DelyBot structure is inspired by delivery robot models available on the market. The robot's 3D model is designed using the urdf and xacro languages. The structure of the robot consists of a box and six cylindrical wheels sized according to real models on the market.

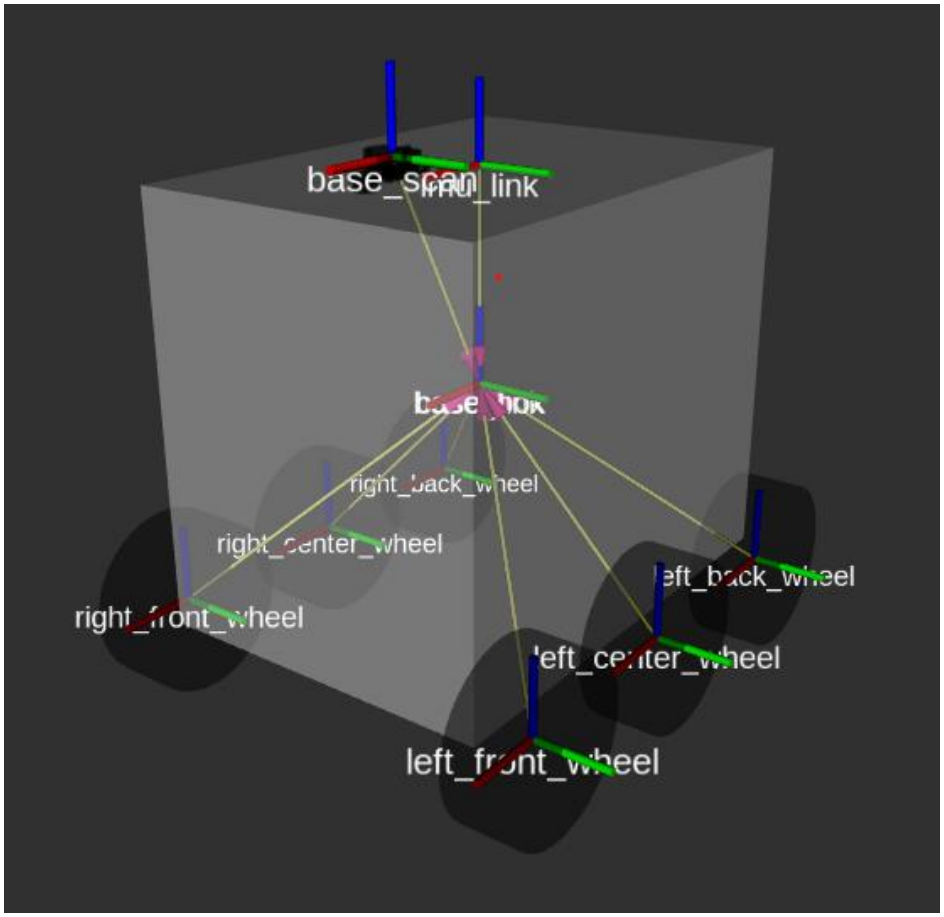
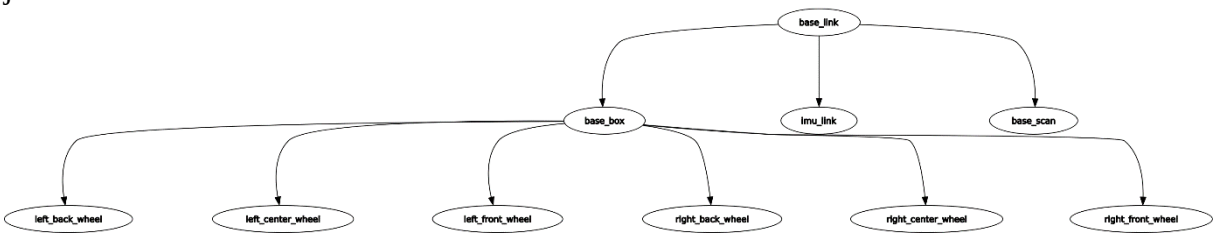


Figure 2.1 DelyBot 3D model (Rviz).

In the following figure it is possible to see how the robot model is structured through links and joints.



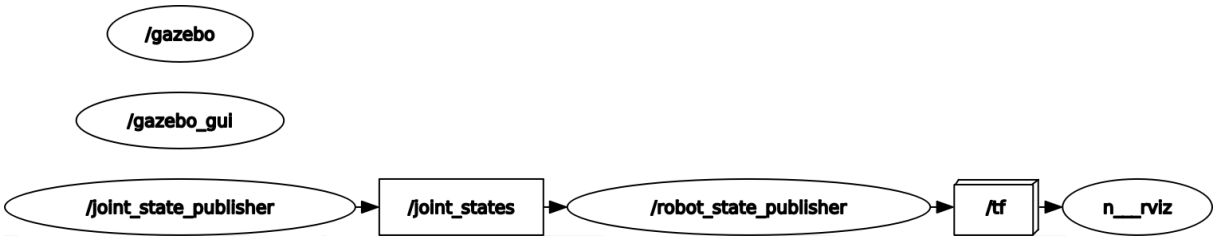
It is defined a base link to which are connected through fixed joints the link of the box and the sensors. To the link of the box are connected the links of the wheels through continuous joints.

2.1 Gazebo Plugin

To enable wheel movement in *Gazebo* a simple transmission is defined and the *libgazebo_ros_control* plugin is used.

To enable sensor operation in *Gazebo* the *libgazebo_ros_gpu_laser* plugin is used for lidar and *libgazebo_ros_imu* for imu.

The nodes and topics used in **delybot_description** are shown in the following image.

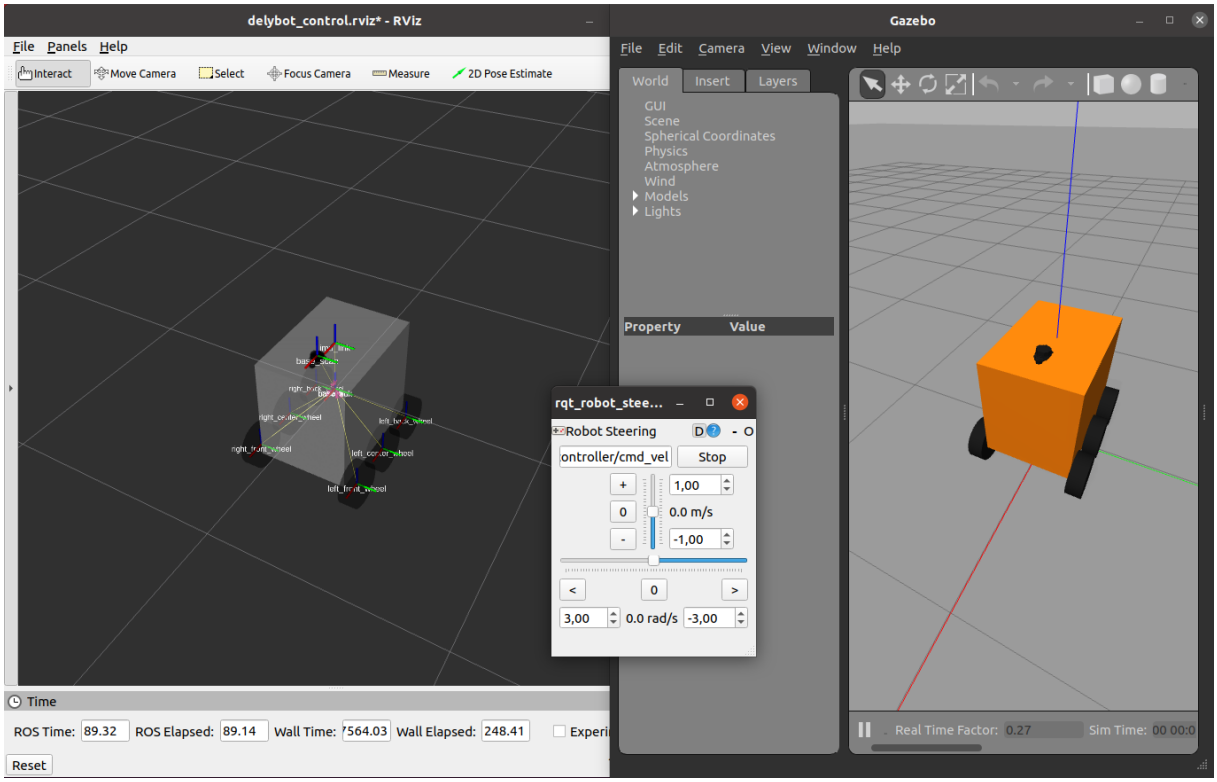


3 DelyBot Control

In the **delybot_control** package the differential drive control of the robot is implemented.

In the `delybot_control/config/` folder there are the following files:

- `joint_state.yaml`, where the *JointStateController* that publishes the state of all joints is defined.
- `delybot_control.yaml`, where the *DiffDriveController* parameters are defined.
- `ekf.yaml`, where the parameters of the *KalmanFilter* are set.



⚠ Localization Warning!

Working on the project, initially the robot only worked with the odometry provided by the *DiffDriveController*. The robot worked poorly, in fact the movements it made on Rviz were different from those that happened in *Gazebo*.

Solution: This problem has been solved by implementing the *imu* sensor whose measurements together with the odometry of the differential drive controller are combined through the Kalman Filter. In this way the robot is able to localize itself correctly.

⚠ Configuration Warning!

Working on the project, initially the inertia of the box and wheels were both set using identity matrices. This resulted in equal reaction to forces along the different directions of the robot, and an unrealistic resulting behavior to accelerations.

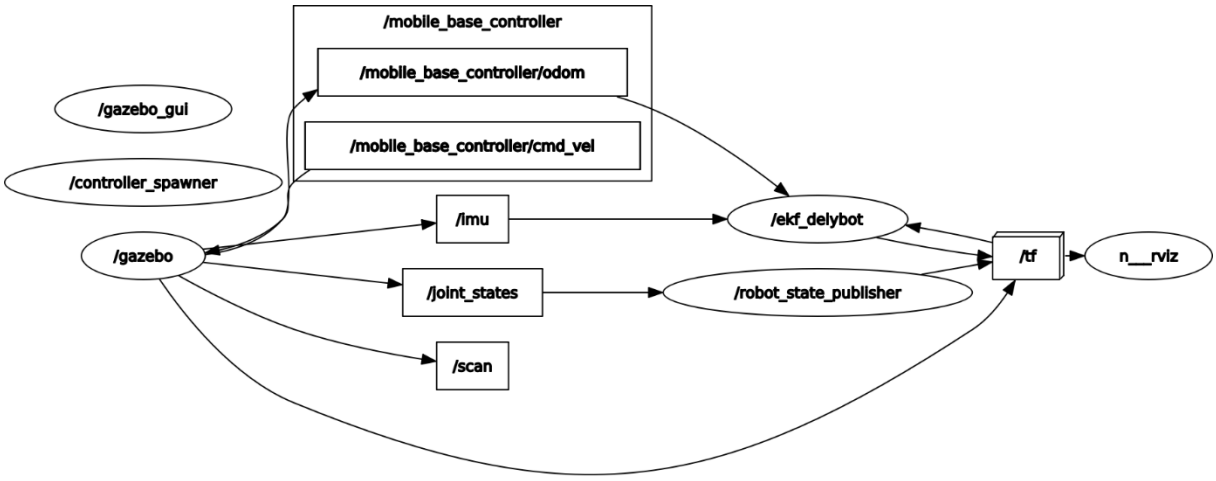
Solution 1: The inertia matrices were set correctly using a cylinder inertia matrix for the wheels and a parallelepiped inertia matrix for the box.

This allowed the robot to move correctly along a straight trajectory; however, the robot's rotation was causing a strong vibration.

Solution 2: To solve the vibration of the robot due to rotation, parameters *kp* and *kd* were introduced set to a high and low value respectively.

The combination of these two solutions resulted in the correct configuration and thus simulation of the robot.

The nodes and topics used in **delybot_control** are shown in the following image.



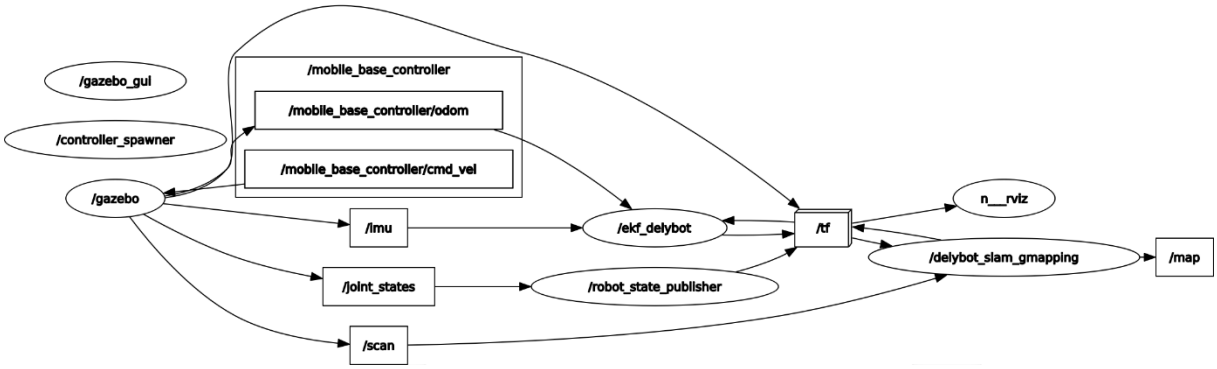
4 DelyBot Slam

In the **delybot_slam** package a technique is implemented to build a map by estimating the current position of the robot in an unknown environment.

The *SLAM* method that is implemented is *gmapping* that processes the sensor data published in the Topics building a 2D map of the world.

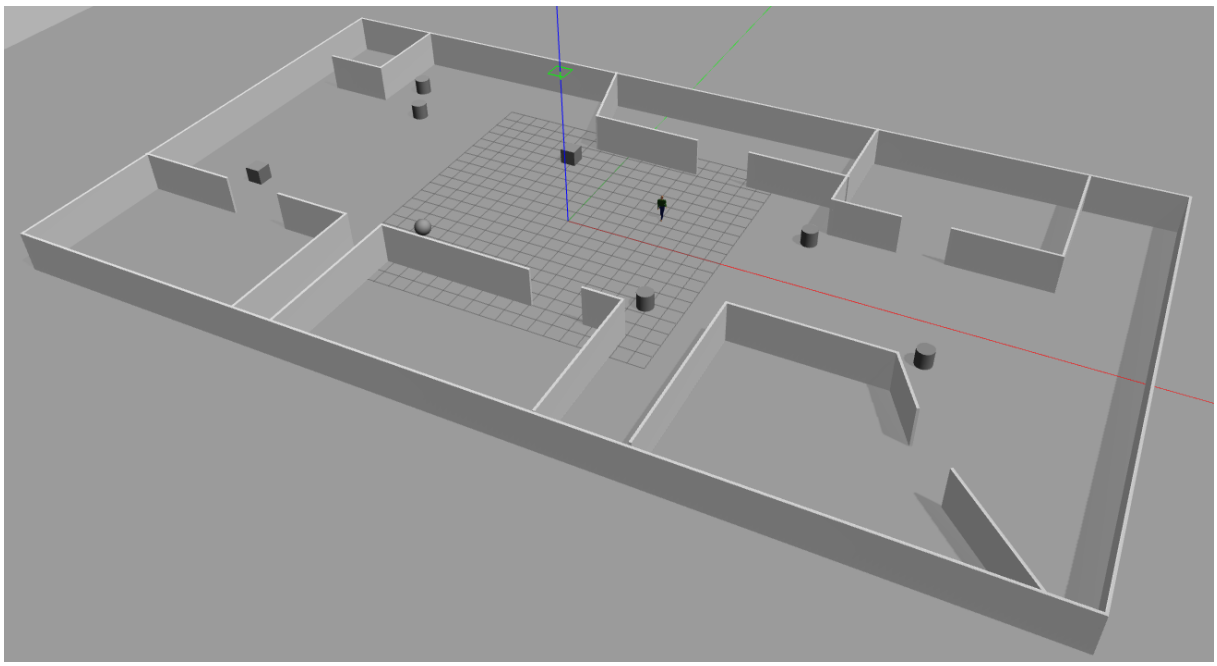
To test the robot in a realistic urban environment with the presence of fixed and mobile obstacles there are two different worlds, which are then mapped using the **delybot_slam** package.

The nodes and topics used in **delybot_slam** are shown in the following image.

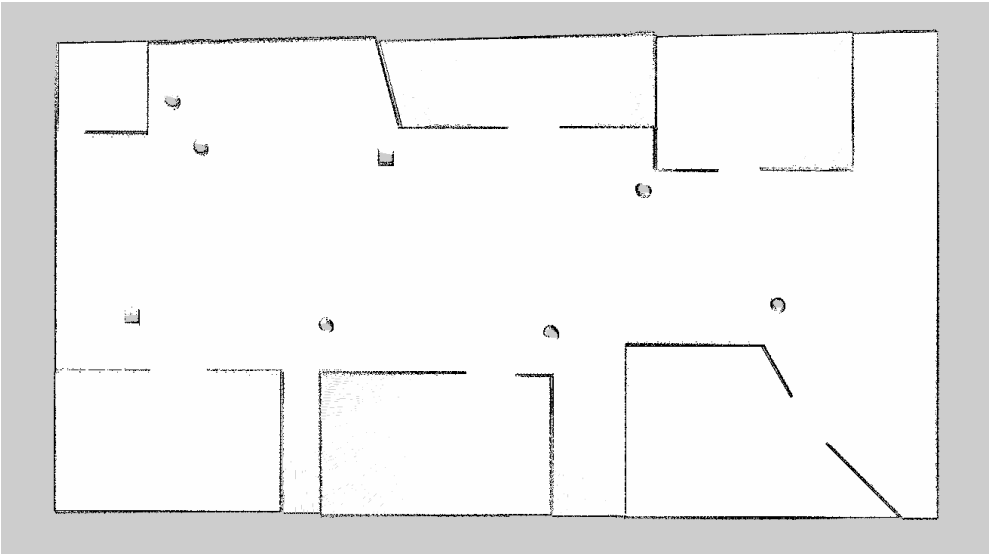


4.1 DelyBot Test World

A simple test world with basic street infrastructure and a single moving agent used to test the robot in the initial phase of the project.

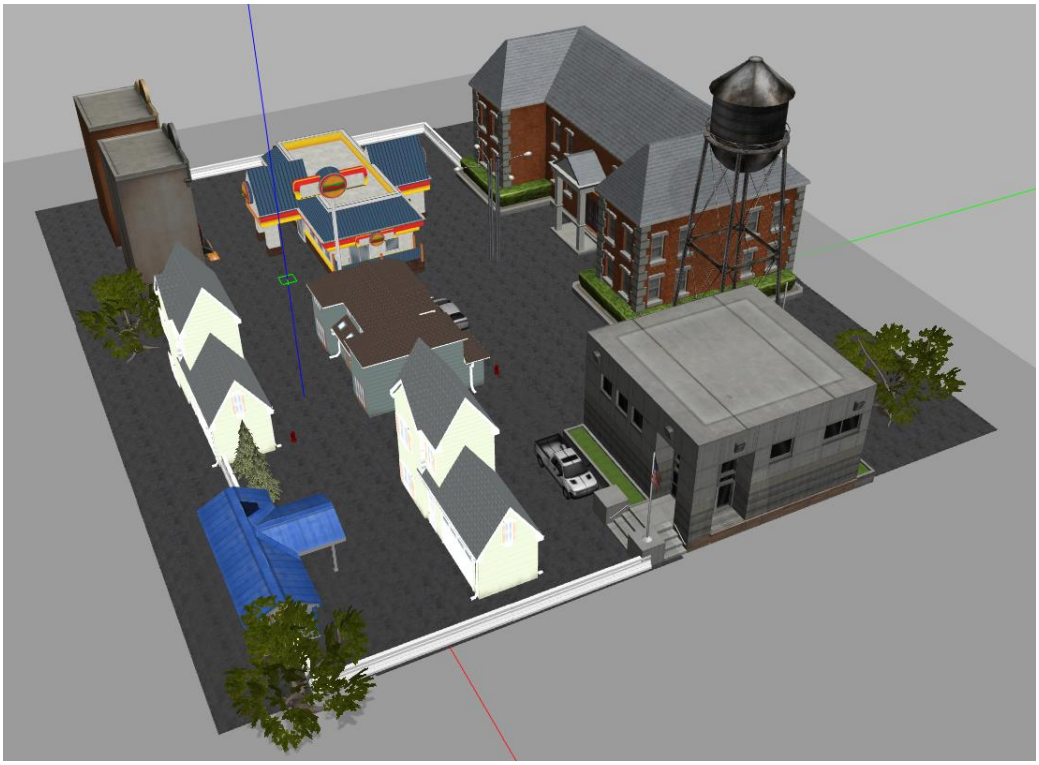


The corresponding 2D generated map is shown below.

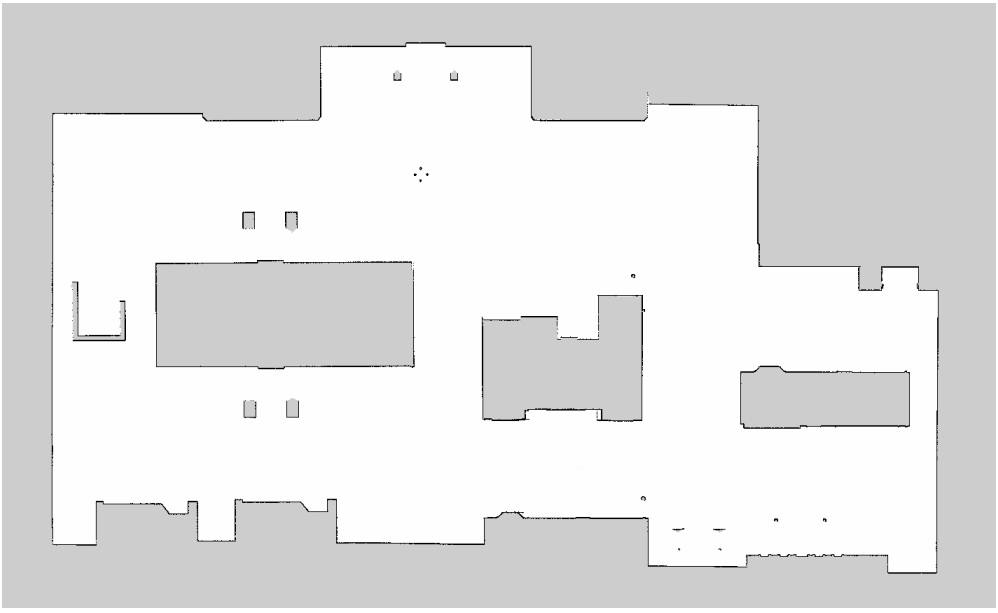


4.2 District World

A more realistic urban environment with a complex infrastructure and multiple moving agents built composing pre-existing *Gazebo* models.



The corresponding 2D generated map is shown below.



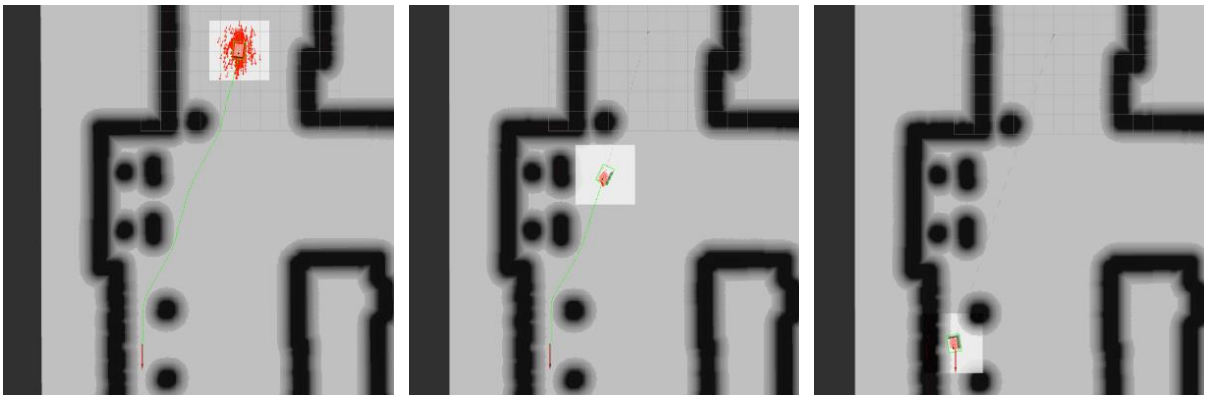
5 DelyBot Navigation

Robot navigation is implemented in the **delybot_navigation** package.

In the `amcl.launch` file, Adaptive Monte Carlo Localization (*AMCL*) is implemented, which is a method for localizing the robot and tracking its pose in a map.

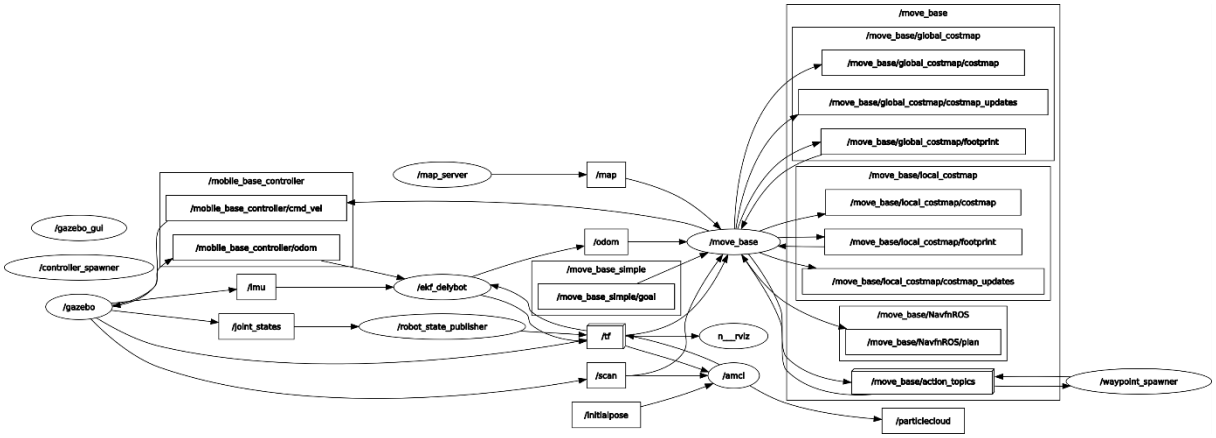
The optimal path planning from the robot pose to the target pose with prior knowledge of the environment and static obstacles is done by the basic *Global Planner.*

On the other hand, the *Local Planner* computes in real-time the new path to avoid dynamic obstacles. It is possible to choose between two implemented Local Planners: *DWA* and *Trajectory Rollout*, both of which work efficiently.



The addition and removal of actors, moving obstacles in the costmap is programmed by correctly setting the laser parameters within the obstacles layer.

The nodes and topics used in **delybot_navigation** are shown in the following image.



6 Usage

6.1 Prerequisites

At first it is necessary to install several external packages:

- navigation

```
sudo apt install ros-noetic-navigation
```

- slam-gmapping

```
sudo apt install ros-noetic-slam-gmapping
```

- map-server

```
sudo apt install ros-noetic-map-server
```

- robot-localization

```
sudo apt-get install ros-noetic-robot-localization
```

6.2 Installation

1. Clone the repo inside the catkin workspace

```
cd ~/catkin_ws/src
git clone https://github.com/gmeidk/DelyBot.git
```

2. Build packages

```
cd ~/catkin_ws
catkin_make
```

6.3 Packages Usage

The possible optional parameter values are listed in the table.

Optional Parameter	Values	Default
`open_rviz`	true, false	true
`world`	empty, delybot_test_map, delybot_test, district_map, district	district
`dwa_local_planner`	true, false	true

The **world** parameter can be used to load a specific world map (the `.world` file must be located in the `delybot_description/world/` folder).

To test the different world files in *Gazebo* it is possible to run the following command:

```
# Usage example to open district.world in Gazebo

roslaunch gazebo_ros empty_world
world_name:=/home/your_username/catkin_ws/src/DelyBot/delybot_description/
world/district.world
```

6.3.1 delybot_description

```
roslaunch delybot_description display.launch
```

Open a pre-configured *Rviz* session to display the robot.

```
roslaunch delybot_description gazebo.launch open_rviz:=true
```

Spawn the robot into the *Gazebo* simulation environment. If the **open_rviz** optional parameter is true a pre-configured *Rviz* session is also opened.

6.3.2 delybot_control

```
roslaunch delybot_control ddr_control.launch world:=district
```

Spawn the robot with a differential drive control and a teleoperation node in *Gazebo*.

6.3.3 delybot_slam

```
roslaunch delybot_slam delybot_slam.launch world:=district_map
```

This command can be useful to create a 2D map of a specific world using a *gmapping* algorithm.

6.3.4 delybot_navigation

```
roslaunch delybot_navigation delybot_navigation.launch world:=district
dwa_local_planner:=true
```

This command is used for the robot navigation, it's possible to give through *Rviz* a desired goal pose. If the **dwa_local_planner** parameter is *true* the *DWA* local planner is used, else if it is *false* the *Trajectory Rollout* local planner is used.

6.3.5 waypoint_spawner node

```
rosrun delybot_navigation waypoint_spawner.py
```

This command run the **waypoint_spawner** node, used to send a specific goal pose selected from a predefined list to the robot. The list is imported from the `waypoint.json` file inside the `delybot_navigation/scripts/` folder.

```
# Output example
user@user:~$ rosrun delybot_navigation waypoint_spawner.py
GOAL LIST:
0) Origin
1) Thrift Shop
2) Salon
3) Home
4) Post Office
5) Police
6) School
7) Fast Food

Insert goal index:
```

Appendix A

The table lists some important topics with their publishers and subscribers.

Topic	Message Type	Publisher	Subscriber
/tf	tf/tfMessage	robot_state_publisher ekf_delybot amcl gazebo	ekf_delybot move_base amcl rviz
/odom	nav_msgs/Odometry	ekf_delybot	move_base
../cmd_vel	geometry_msgs/Twist	move_base rqt_robot_steering	gazebo
/scan	sensor_msgs/ LaserScan	gazebo	amcl rviz move_base
../goal	move_base_msgs/ MoveBaseActionGoal	move_base waypoint_spawner	move_base
../status	actionlib_msgs/ GoalStatusArray	move_base	waypoint_spawner

References

[1] navigation package: http://wiki.ros.org/navigation
[2] slam-gmapping package: http://wiki.ros.org/gmapping
[3] map-server package: http://wiki.ros.org/map_server
[4] robot-localization package: http://wiki.ros.org/robot_localization
[5] actor: http://gazebosim.org/tutorials?tut=actor&cat=build_robot