• Software Architecture Project

Group: NAV-03
Student(s): Ermanno Girardo, Alessio Roda, Enzo Ubaldo Petrocco
Tutor(s): Simone Macciò
Year: 2020 - 2021

Mobile Navigation and Mapping

Table of Contents

.

Abstract

The aim of this project is to allow a localization and path planning of a non holonomic robot in a unknown environment mapped while moving.
At the end of the day the code will be implemented in an autonomous vehicle made by Husqvarna company.
Here you can find the GitHub repository with the source code:
https://github.com/AlessioRoda/mobile_robot_navigation_project

1. Introduction

The scope of this project is to reach a defined position with a Husky autonomous vehicle equipped with a LIDAR sensor and four wheels in an unknown environment. The vehicle must move without colliding against the obstacles and create a map of the environment in which it moves.

The project is divided in two parts, one for the code to implement the algorithm and the other to make a simulation. The code is developed for a ROS environment (in particular this code is developed for ROS noetic), while the simulation is performed by Unity. Then to run the simulation please take care of what follows:

In order to realize the project you need to have two machines.
Once runs Unity (please see the link in order to download the 2020.2.2. version https://unity3d.com/get-unity/download/archive).
In the second machine you must have a ROS workspace with all source for launch the node required.
In Unity you can run the simulation that you can download at this link: https://github.com/TheEngineRoom-UniGe/SofAR-Mobile-Robot-Navigation.
In the link above there are also file for ROS in order to establish the connection with Unity.
In order to localize, move the robot and create the map, move_base and slam_gmapping packages are required.

2. Architecture of the System

Data are acquired from LIDAR sensor and sent to SLAM_GMAPPING via /laser_scan topic via publisher subscribe design pattern. The data acquired into the Unity simulation are sent to ROS.

The position of the robot is acquire and sent to ROS via publish subscribe design pattern via /odometry_frame topic.
The base_link frame is published from Unity to ROS via /tf topic.
In ROS the position, via a transformation is published on /odom topic.
SLAM_GMAPPING package uses data from /tf and from /laser_scan in order to publish a map on /map topic.
MOVE_BASE package takes as input the map generated by slam_gmapping and the information from Unity simulation: the position of the robot and the goal position, that is published on the topic /move_base_simple/goal once the user presses the button “Move!”
You can see all the UML diagram of the components and the temporal diagram in the following figures:

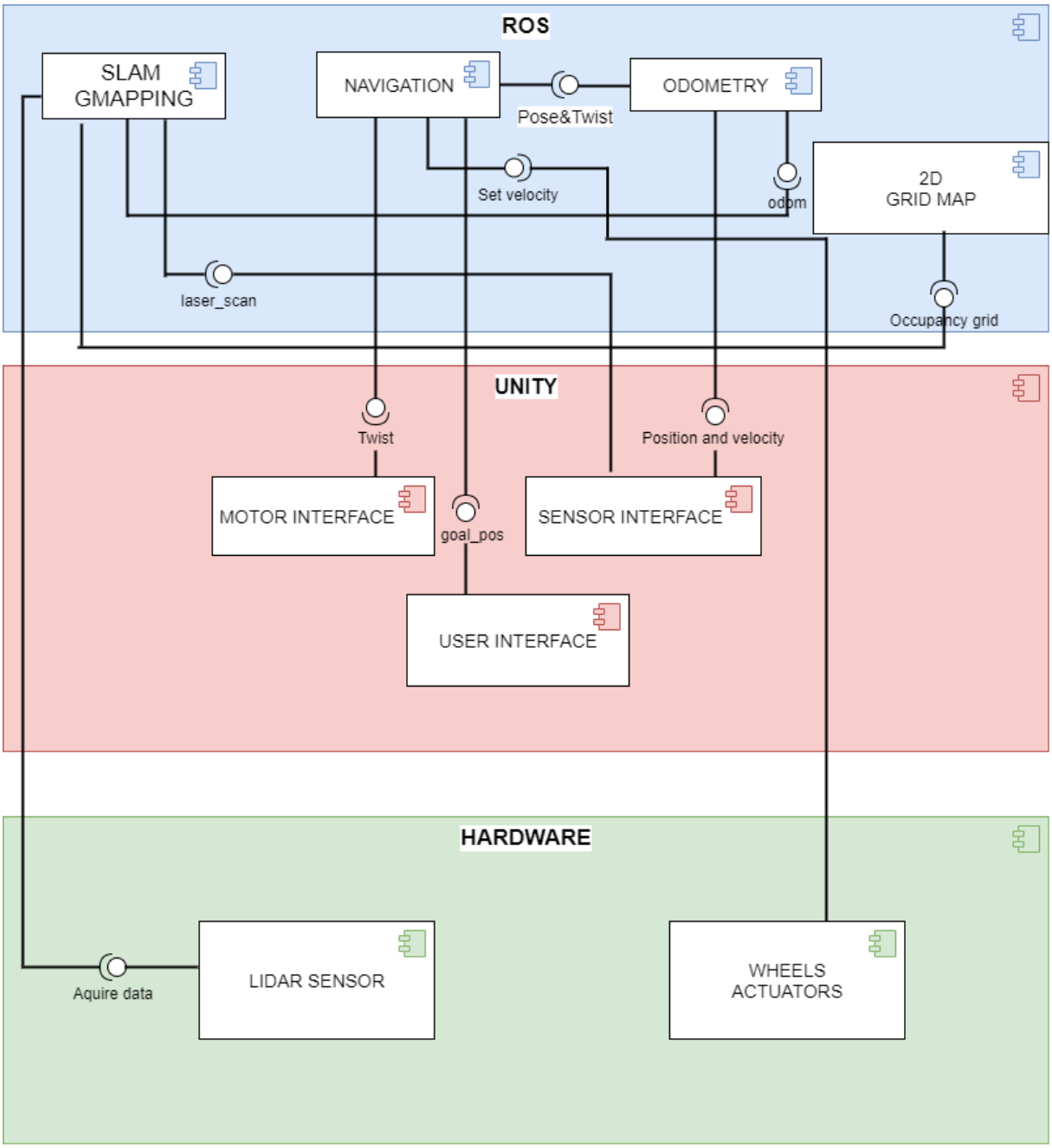

Figure 1: UML components diagram

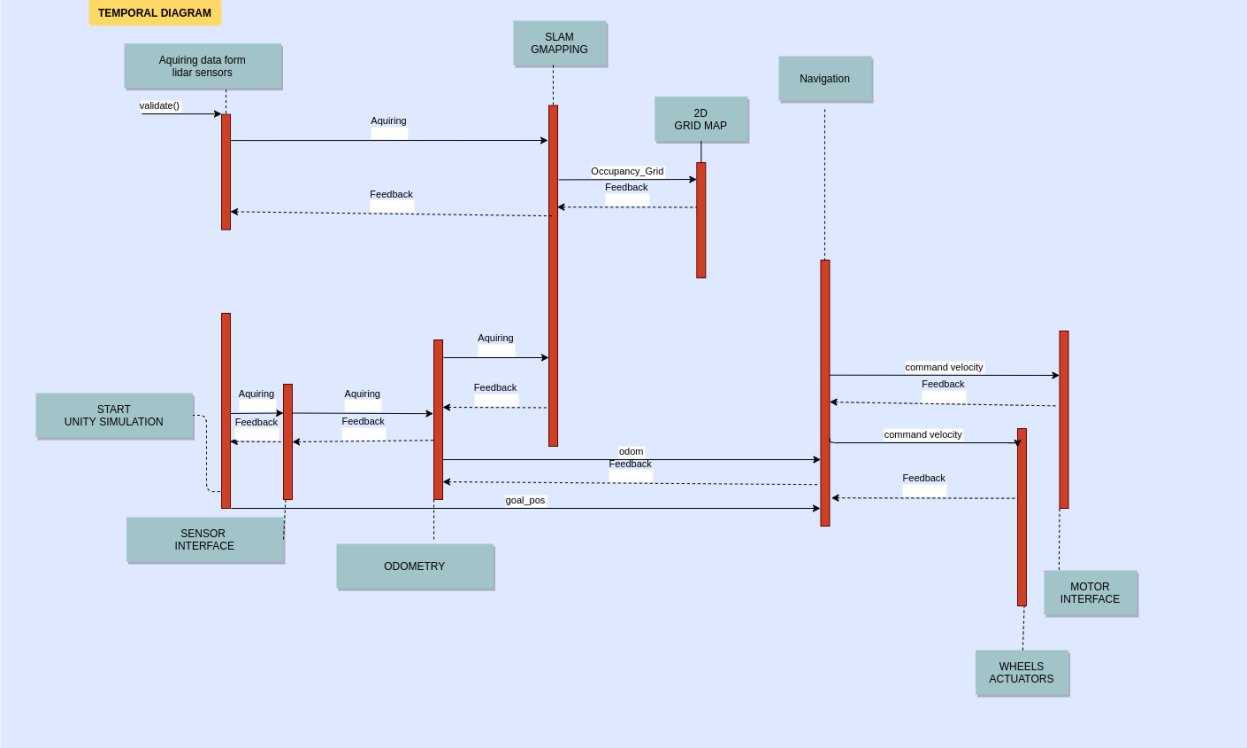

Figure 2: UML temporal diagram

3. Description of the System's Architecture

In this chapter, a list of all modules is described.

3.1. Module 1: sensor interface

This module is provided by Unity scene to ROS.
It is constituted by two publish/subscribe design pattern.
In particular, the first is a simulated implementation of a real LIDAR sensor.
Laser scan does not take any input but publishes on the /laser_scan topic.
It acquires 220 samples in a 220 angular degree. The maximum distance that the sensor can see is 20 meters.
The second via a sensor (like for instance GPS or others), publishes the position of the robot in the Unity environment to ROS via /odometry_frame topic.

Figure 3: LIDAR sensor

3.2. Module 2: user interface

This module has the scope of publish the goal position at the start of the simulation.
In the Unity graphical interface, when the user pushes the button “Move!”, the position of the goal is published on the /move_base_simple/goal topic.
In Unity scene the goal is marked with a red circle.

.

3.3. Module 3: slam_gmapping

SLAM stays for Simultaneous Localization And Mapping.
The node implemented in ROS is called slam_gmapping.
This module has the scope of creating a map on the /map topic of type nav_msgs/OccupancyGrid. It is possible since it takes as input the position of the robot, via /tf topic which contains the transforms necessary to relate frames for base and odometry. Furthermore, it receives as input the LIDAR sensor data from /scan.
Gmapping is based on a particle filter, approach to estimate a probability density.
In order to model the package provided by ROS, a certain number of parameters must be setted. For more detail about gmapping you can find the WIKI:
http://wiki.ros.org/gmapping

3.4. Module 4: navigation

The navigation module is simply the core of the architecture.
It has the task of allow the robot reaching the goal position using navigation stack. It is implemented with the move_base ROS package.
The move_base node links together a global and local planner to accomplish its global navigation task.
The local planner works only with the information it currently gets from the sensors and plans a path that is limited in space. When the next set of information come in it plans a new piece of the path.

The global planner, on the other hand, build a map of the environment. It gathers all the information ever received and then plans a path that reaches to the goal, considering the whole map.
By default, the move_base planner uses as global planner, the NavfnROS plugin, which plans a path based on the Dijkstra's algorithm.
It is a graph search algorithm that solves the shortest path problem for a graph with non-negative edge path costs, producing a shortest path from a starting vertex to an ending vertex. This implementation can be used with a directed graph and an undirected graph.
There are many parameters that you can set for the planners and the coastmap. You can find all these parameters in the param folder of our repository.
In the figure above you can find all the inputs, the outputs and the behavior of the move_base package.
For more detail about move_base kindly see the WIKI:
http://wiki.ros.org/move_base

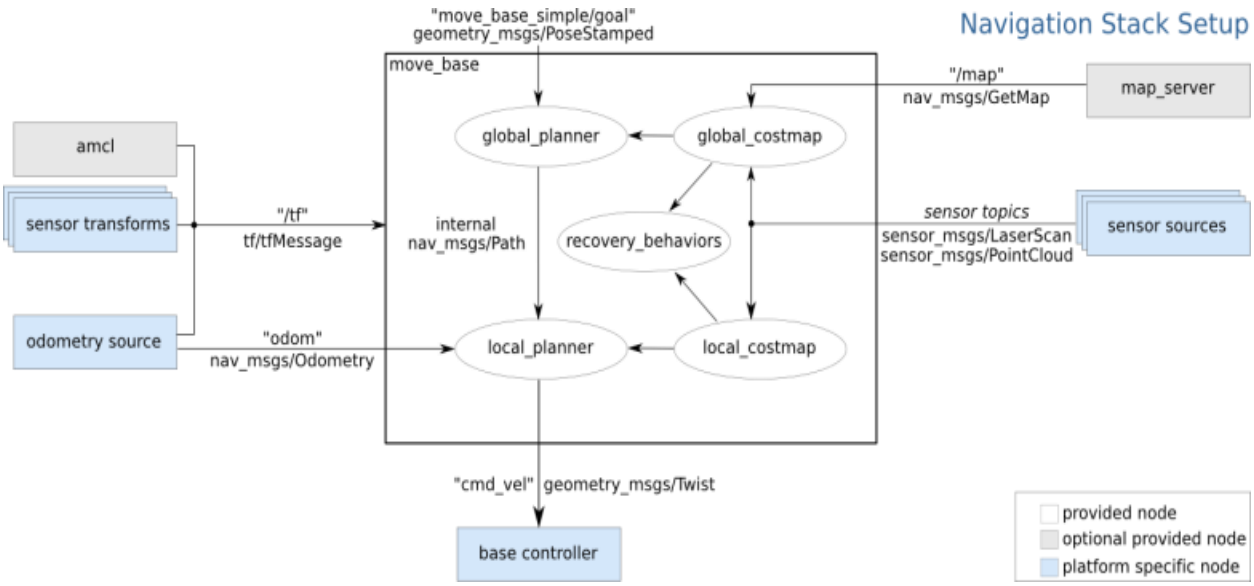

Figure 4: Move base behavior.

3.5. Module 5: odometry

The script publish_odometry.py has the scope of an adapter design pattern.
This is simply a transformation of the odometry of the robot. In fact, this node take the data related to the position of the robot on the /odometry_frame topic and republish it on /odom topic.
This is necessary since the slam_gmapping module and move_base need the information related to the position of the robot on /odom topic.

3.6. Module 6: motor interface

The motor interface module has the scope of manipulate the data that move_base publish on topic /cmd_vel, in order to set properly the rotation of the wheels, so that the robot could reaches the goal and avoids the walls. This is fully implemented into Unity side, you can see all the materials related to Unity here: https://github.com/TheEngineRoom-UniGe/SofAR-Mobile-Robot-Navigation

3.7. Module 7: Hardware Lidar sensor

This module is real LIDAR sensor of the autonomous Husqvarna vehicle. This kind of sensors must be compatible with the LIDAR sensor of the simulation, in this case it is a laser, but it can be other as, for instance, a camera.

3.8. Module 8: Wheels' actuators

This module has the scope of transduce the command velocity that move_base publish on topic /cmd_vel into a current signal, in order to set the wheels' rotation velocity accordingly with the data published on /cmd_vel.

4. Installation

First, in order to install all the module required for the simulation, verify to have a ROS noetic version installed into your machine.
Here you can find the WIKI for the noetic installation: http://wiki.ros.org/noetic
Since the simulation starts on Unity you can use another machine with Unity 2020.2.2 version. Please click on the following link in order to download the correct version for you OS: https://unity3d.com/get-unity/download/archive
In order to establish a TCP/IP connection, the two machines should stay under the same LAN or you can create a virtual LAN, in order to establish a remote connection.
If you do not have two machines at your disposal, you can use a virtual machine or a docker image.
If you want to use a docker image, what I suggest you to do is to launch the container in this way: docker run -dit -p 6080:80 -p 5901:5900 -p 22:22 -p 10000:10000/tcp -p 10000:10000/udp --name <container_name> <image_name>.
Now I want to investigate about how you can establish the connection between ROS and Unity.

First of all download the package that you can find at this GitHub repository:
https://github.com/TheEngineRoom-UniGe/SofAR-Mobile-Robot-Navigation
Here you can find the Unity scene and the package for ROS in order to establish the connection.
If you are using two machines under the same LAN, you can set into Unity the IP of the ROS machine, and into the file /config/params.yaml, you must the ROS and Unity IPs machines.
If you are using a docker image in order to establish a communication you can look at this link: hypothe/sofar_ros (docker.com)
Then you should verify to have the libraries already installed into your Ubuntu system.
Please run these commands in order to install the navigation and gmapping libraries for ROS noetic version:

- **sudo apt-get install ros-noetic-navigation** to install the navigation library
- **sudo apt-get install ros-noetic-openslam-gmapping** to install the gmapping library

Please clone our GitHub repository into your ROS workspace:
https://github.com/AlessioRoda/SofarProject
Once done these steps you are ready to install the modules.

4.1. Module: Husky

As mentioned below, the robot used in the Unity simulation is a Husky autonomous vehicle (see Figure 5).
Then you can clone the package that you can find here, in your ROS ws:
https://github.com/husky/husky
This step is required in order to have the correct URDF to spawn the robot into RViz simulator.
Once done this step you must add into your machine a library to use correctly this package, then please run:

sudo apt-get install ros-noetic-lms1xx

Figure 5: Husky Robot

4.2. Module: slam_gmapping

You can find into our GitHub repository the folder launch. Inside it you can find all the launch file required for the simulation.
In order to launch the slam_gmapping node and all its parameters please run:

roslaunch mobile_robot_navigation_project gmapping.launch

Then you also have to download the gmapping package for ROS noetic (here the link: https://github.com/ros-perception/slam_gmapping)
Please note that in order to allow a correct subscription from the laser sensor data, the /scan topic is remapped as /laser_scan topic.

4.3. Module: move_base

In order to launch the move_base node and all the parameters required for move_base package you can run the following command:

roslaunch mobile_robot_navigation_project move_base2.launch

All the parameters are executed via move_base2.launch. You can find all the parameters for the local, the global planner, the general parameters for move_base and the coastmap into the param folder.

4.4. Module: view

The view model has the purpose of load the URDF and the transformations of the robot, in order to see the robot and the map created by slam_gmapping into RViz simulator. Also a file needed for upload the maps (the first is reffered to the topic /map, the second to the topic /local_coastmap and the last one to the /global_coastmap), the path and the robot description into the RViz simulator.
You can find this file (model.rviz) into the folder config of our repository.
In order to launch this launch file please run the following command:

roslaunch mobile_robot_navigation_project view_model.launch

If everything went well RViz simulator should open.
Do not worry if you do not see the robot well, once the communication will start between ROS and Unity, everything will be fine.

4.5. Module: navigation

Last but not least, the navigation module in order to establish the communication between ROS and Unity. It runs also the odometry_publisher.py script in order to publish the data received from /odometry_frame into /odom topic.
In order to start the communication please run the following command:

roslaunch mobile_robot_navigation_project navigation.launch

Only here please click the button to enable the start of the simulation in Unity.
If the connection was successful, the HandShake message will appear into your Ubuntu shell.

5. System Testing and Results

The overall simulation is tested via two machines with a remote connection.
In order to do this, we have used Hamachi software in order to create a VLAN, here you can find the link to download it: https://www.vpn.net/
Drawback: normally the ping is low in our case, but if you do not have an efficient connection this method is not recommended.
You can do also in the other ways that I listed you in this guide.
In order to make a correct simulation, I also suggest to you to launch all the module on the ROS side and only then launch the simulation on Unity.
We have launched the simulation more and more in order to set the parameter to the best, both the parameter for move_base and also for gmapping.

The final result that we have obtained is represented in the following figure 6 related to the map build by gmapping,
In the figure 7 instead, you can see a rqt_graph of the nodes running in the simulation.

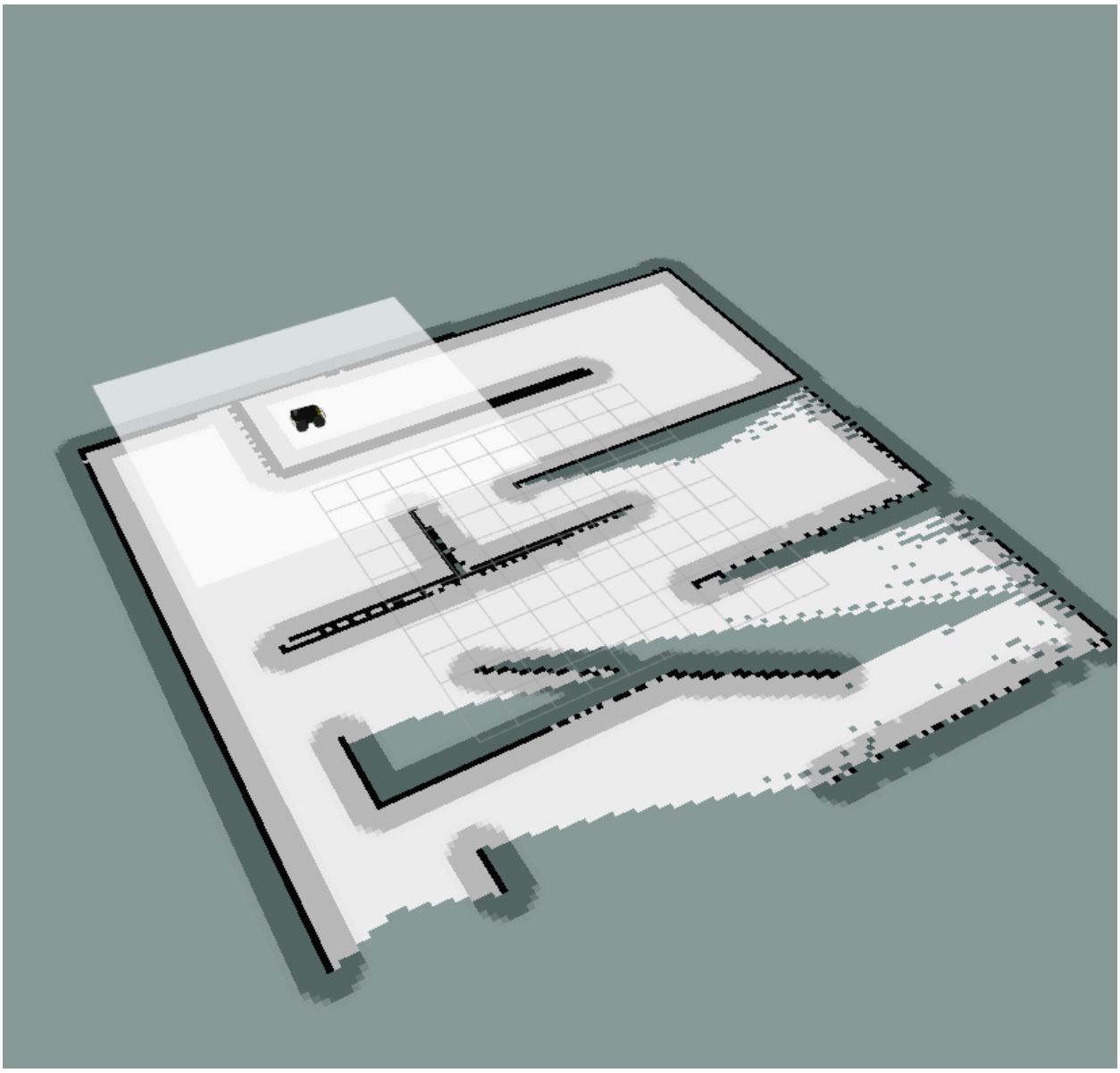

Figure 1: Map build in RViz via gmapping module

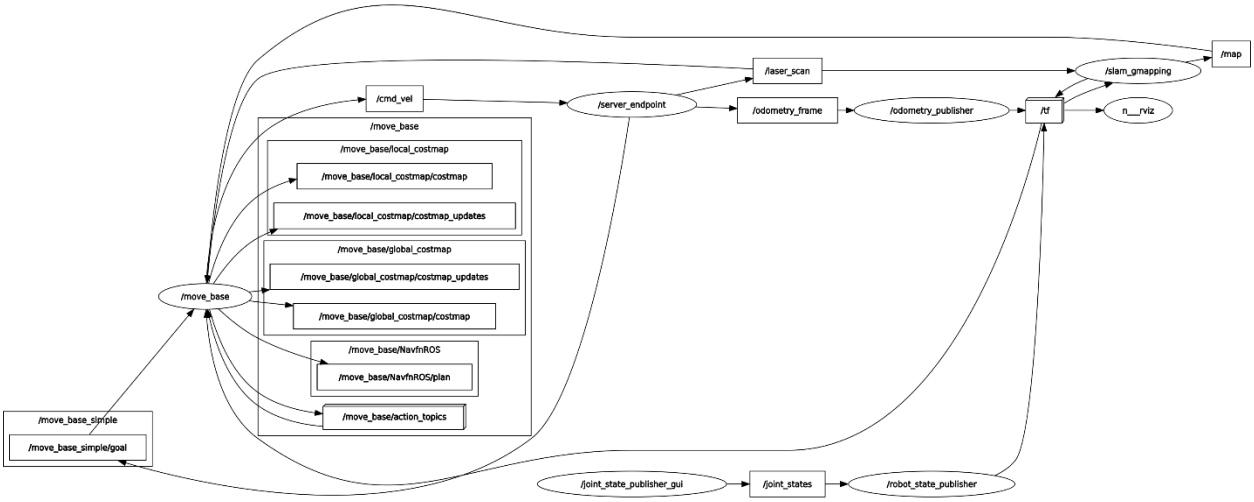

Figure 2: rqt_graph of the simulation

In out GitHub repository you will find also the video of the simulation.

6. Recommendations

Here some recommendations in order to perform optimally the simulation:

1. The environment in which the robot has to pass mustn't be too arrow, since the husky maintains a certain distance between the obstacles and so it won't be able to move inside.
2. During the simulations performed, it has been discovered that in order to have a good reconstruction of the map, the robot can't rotate too fast (so be careful if you want to change the angular velocity value)
3. If the robot tries to move too fast it will overturn itself (so be careful if you want to change the linear velocity value).