* 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

[Table of Contents 1](#_Toc74261903)

[Abstract 1](#_Toc74261904)

[1. Introduction 2](#_Toc74261905)

[2. Architecture of the System 3](#_Toc74261906)

[3. Description of the System’s Architecture 5](#_Toc74261907)

[3.1. Module 1: sensor interface 5](#_Toc74261908)

[3.2. Module 2: user interface 5](#_Toc74261909)

[3.3. Module 3: slam\_gmapping 6](#_Toc74261910)

[4. Installation 8](#_Toc74261911)

[4.1. Module: Husky 9](#_Toc74261912)

[5. System Testing and Results 10](#_Toc74261913)

[6. Recommendations 12](#_Toc74261914)

[6.1. Module <name of the module> (can be more than one) 12](#_Toc74261915)

.

# 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

# Introduction

The scope of this project is to allow the achievement of a goal in a unknown environment.

In particular the robot investigated is a Husky autonomous vehicle [figure 1] equipped with LIDAR sensor [figure 2] and four wheels.

Immagine che contiene LEGO, giocattolo

Descrizione generata automaticamente

Figure 1: Husky Robot



Figure 2: LIDAR sensor

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.

# 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 press the button “Move!”

You can see all the UML diagram of the components and the temporal diagram in the following figures:

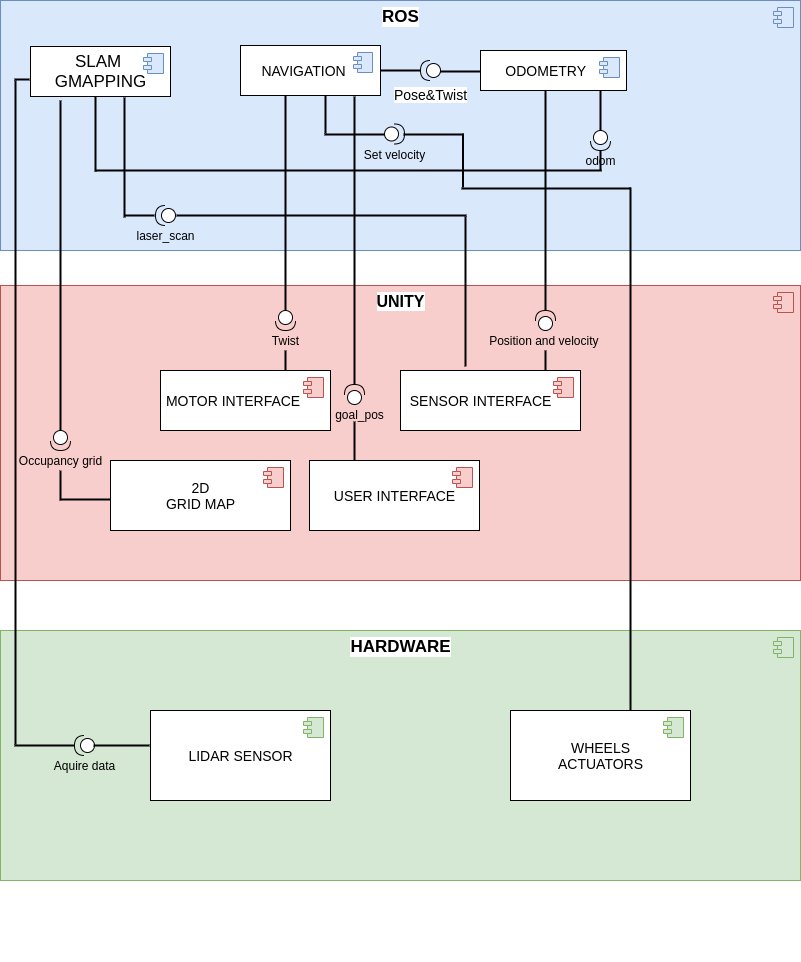


Figure 3: UML components diagram

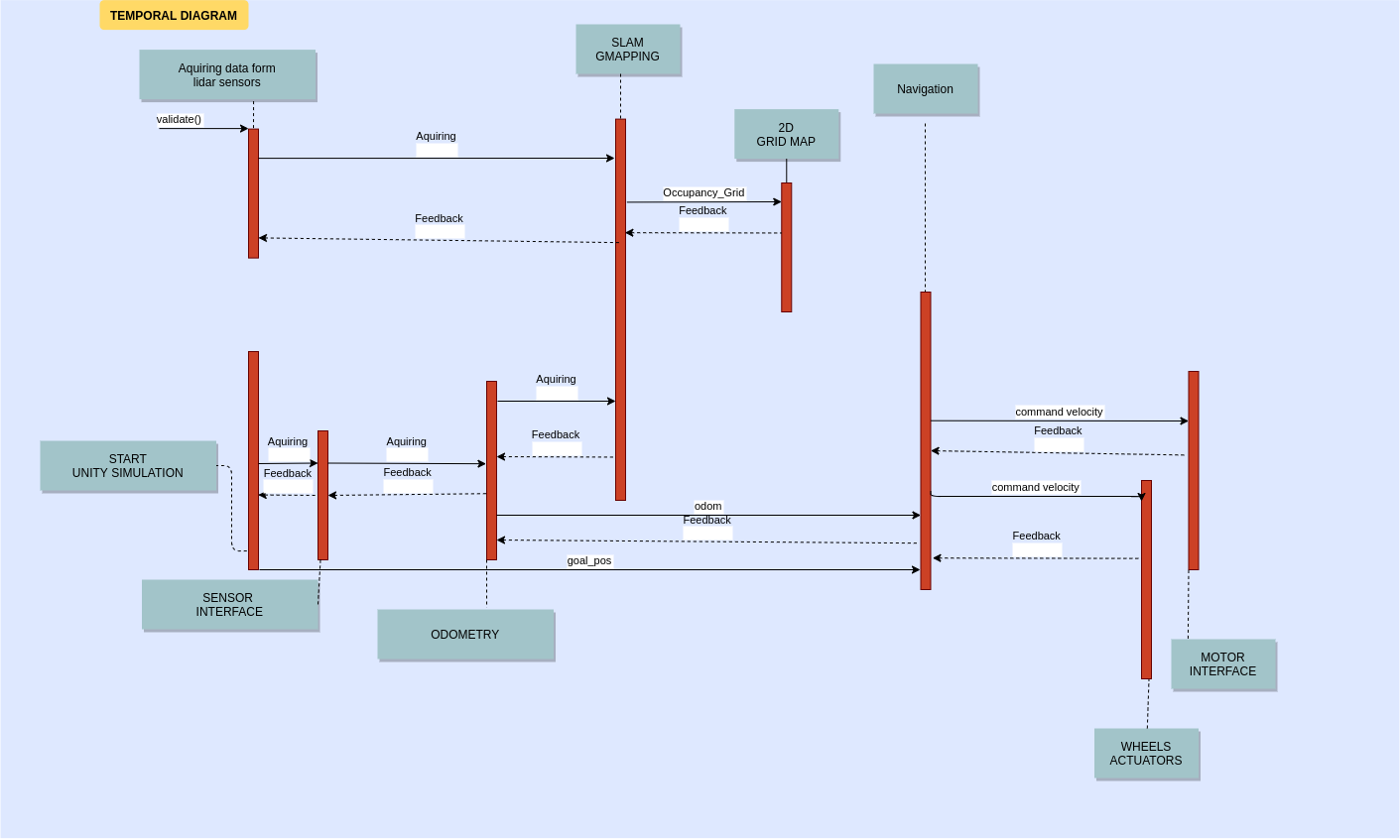


Figure 4: UML temporal diagram

# Description of the System’s Architecture

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

## 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 publish 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 GPS, publish the position of the robot in the Unity environment to ROS via /odometry\_frame topic.

## 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.

.

## 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>

* 1. 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>

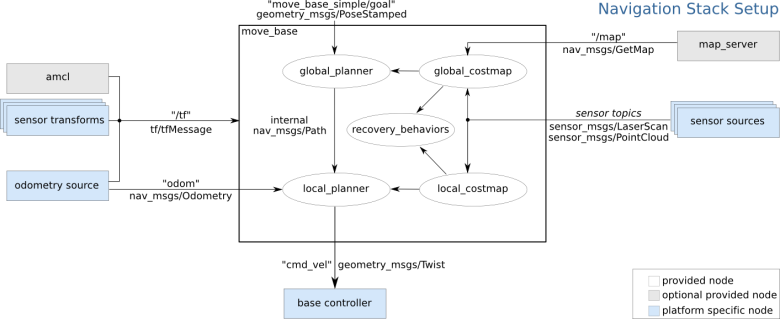


Figure 5: Move base behavior.

* 1. 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.

* 1. 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>

* 1. Module 7: Hardware Lidar sensor

This module is real LIDAR sensor of the autonomous Husqvarna vehicle.

* 1. 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.

# 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)](https://hub.docker.com/r/hypothe/sofar_ros)

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.

## Module: Husky

As mentioned above, the robot used in the Unity simulation is a Husky autonomous vehicle (see Figure 1).

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**

* 1. 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**

Please note that in order to allow a correct subscription from the laser sensor data, the /scan topic is remapped as /laser\_scan topic.

* 1. 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.

* 1. 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.

* 1. 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.

# 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.

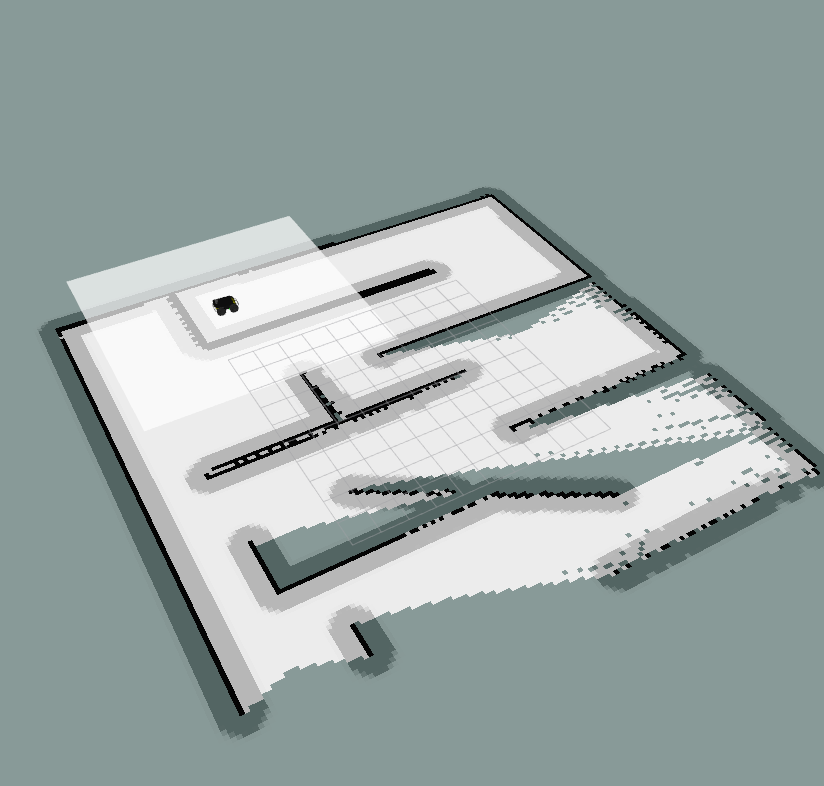


Figure 6: Map build in RViz via gmapping module

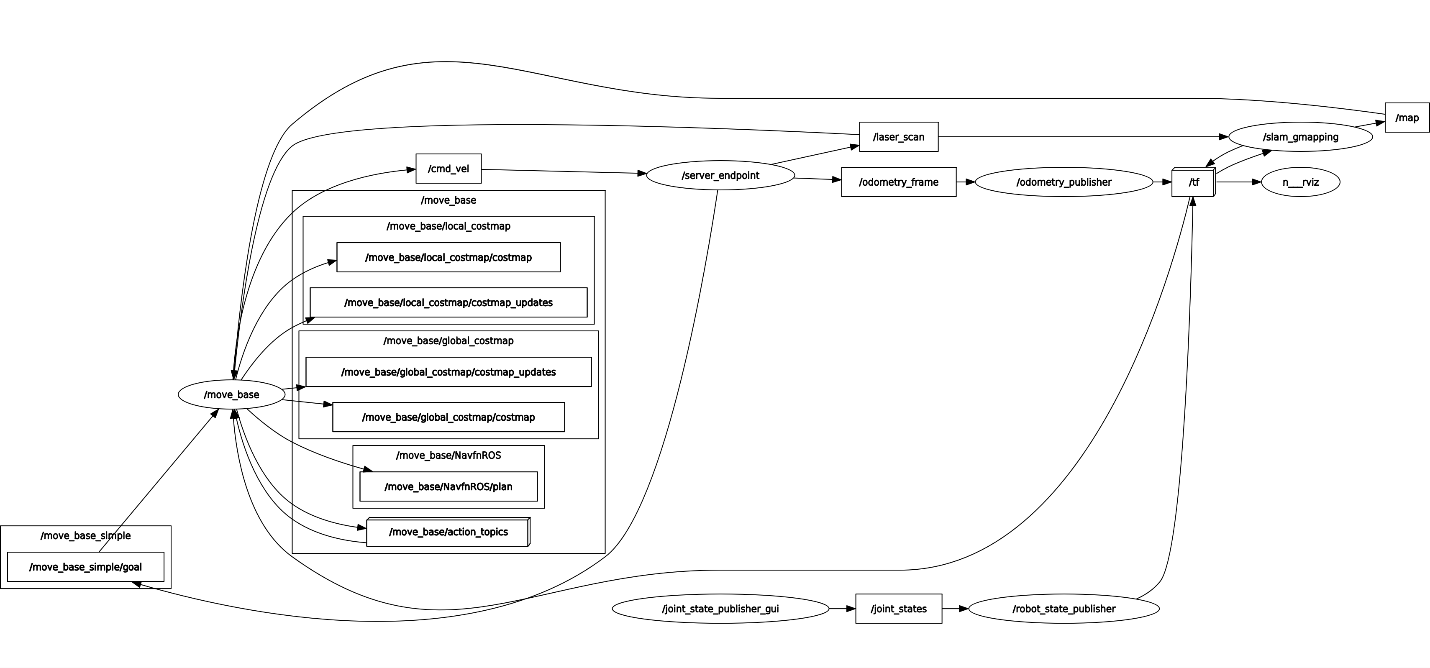


Figure 7: rqt\_graph of the simulation

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

# Recommendations

The recommendations follow naturally from the system testing and results. **Note that**: if all the modules have successfully completed their work and integrated everything together, then this section can present the overall recommendations for the “whole” system, instead of having a sub-section dedicated to the recommendations for each module.

## Module <name of the module> (can be more than one)

This subsection presents the recommendations for the module, i.e., **(i)** the assumptions made while building the module (and/or) the limitations of the working module, **(ii)** presenting possible ideas that could overcome the limitations or assumptions.