SOFTWARE ARCHITECTURES FOR ROBOTICS

SOFAR Project Report
(Mobile Robot Navigation & Mapping)

Group: NAV-05
Repository Link

https://github.com/mohammed-eldin/SOFAR_NAV_05

Students:
Mohamed Qaoud - S4729321
Mohamed Mahmoud - S4844271
Naresh Arthimalla - S4927772

Table of Contents:

Aim of The Project

The aim of the project is localize and path planning for a mobile robot with an unknown environment for the robot in UNITY platform.

Also, using LIDAR sensor on the robot to create a map in Rviz simulator.

1 - Introduction

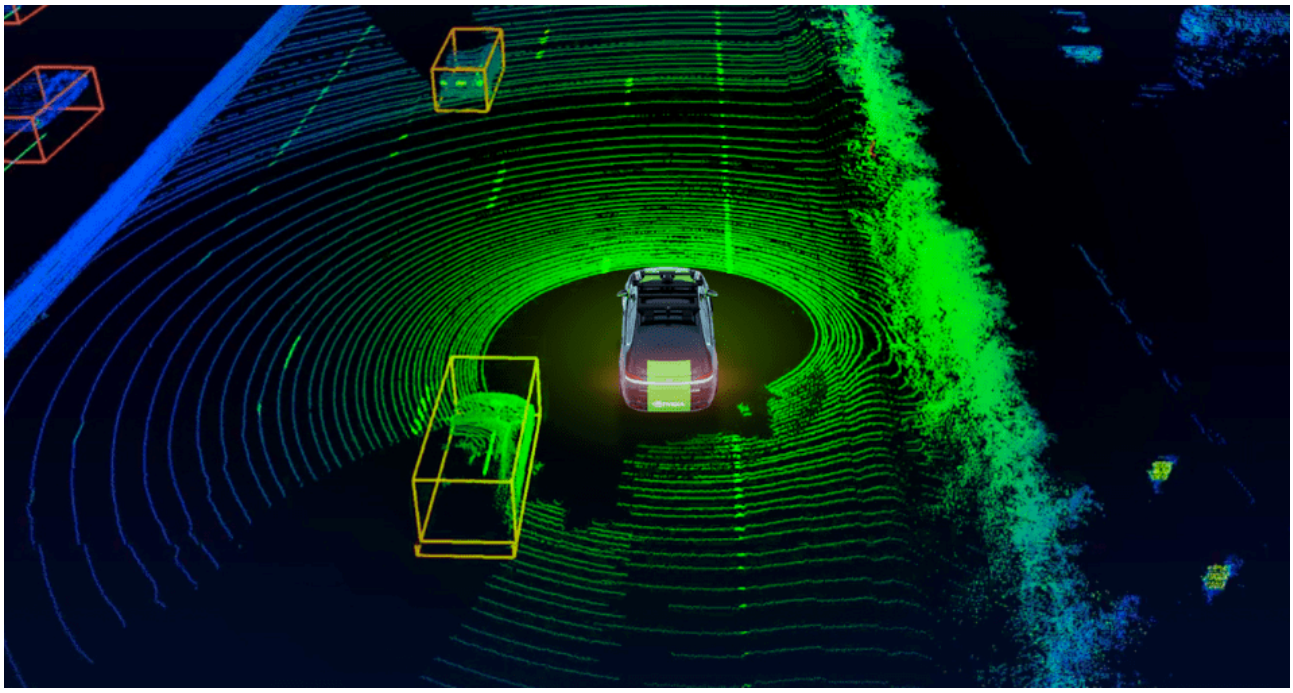
In order to do the aim of the project we have to consider the following:

- Unity editor, which is an engine that can be used to create three-dimensional (3D) and two-dimensional (2D) environments, as well as interactive simulations and other experiences.

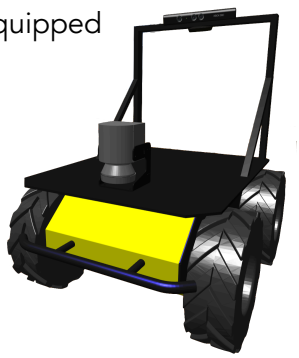
- ROS Environment: Where we use it to control the robot movement and get the created map from the Unity environment to Rviz.

- Rviz is the ROS simulation that will simulate the scanned environment for the LIDAR sensor that attached to the mobile robot.

- LiDAR Sensor is used to determining ranges by targeting an object with a laser and measuring the time for the reflected light to return to the receiver.



- The robot that is used is the four wheels Husky autonomous robot equipped with a LIDAR sensor.



With all he above mentioned, we could then connect them all together to get the aim of the project. In the following sections, you would find how could this project done.

2 - Software Architecture

The software architecture mainly describes the overall flow of the data, and as you can see in the following components diagram:

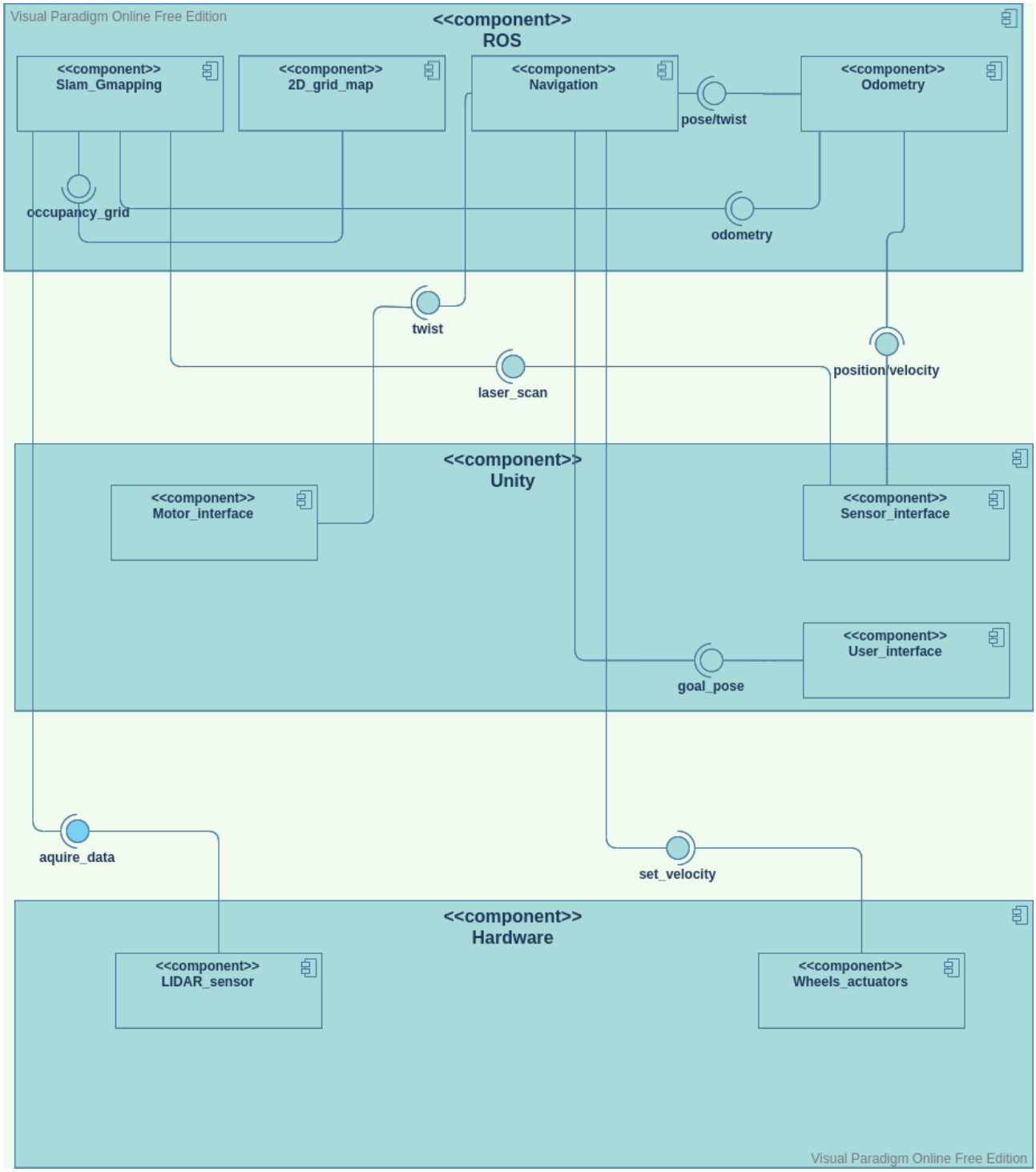


Figure 2.1: *component diagram of the software architecture.*

Where:

- Data are acquired from LIDAR sensor (Unity) and sent to SLAM_GMAPPING via /laser_scan (ROS) topic via publisher subscribe design pattern.

- 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 is published via a transformation 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 take, 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.

In the Rviz scanned map, you could move the robot in the Unity environment by just select a 2d Nav position and click on the desired location on the Rviz map. On the other hand, in Unity, the robot goes to the desired selected point from Rviz.

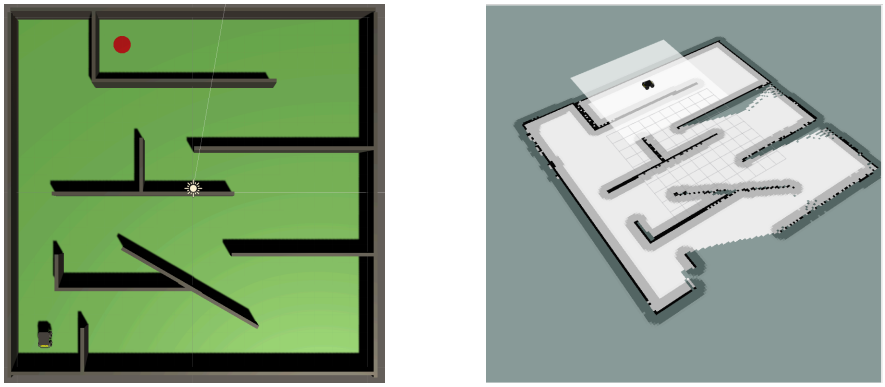


Figure 2.2: *Left figure is the Unity environment, while the right figure is the Rviz scanned map.*

The following figure2.3 shows the flow of the Temporal UML diagram.

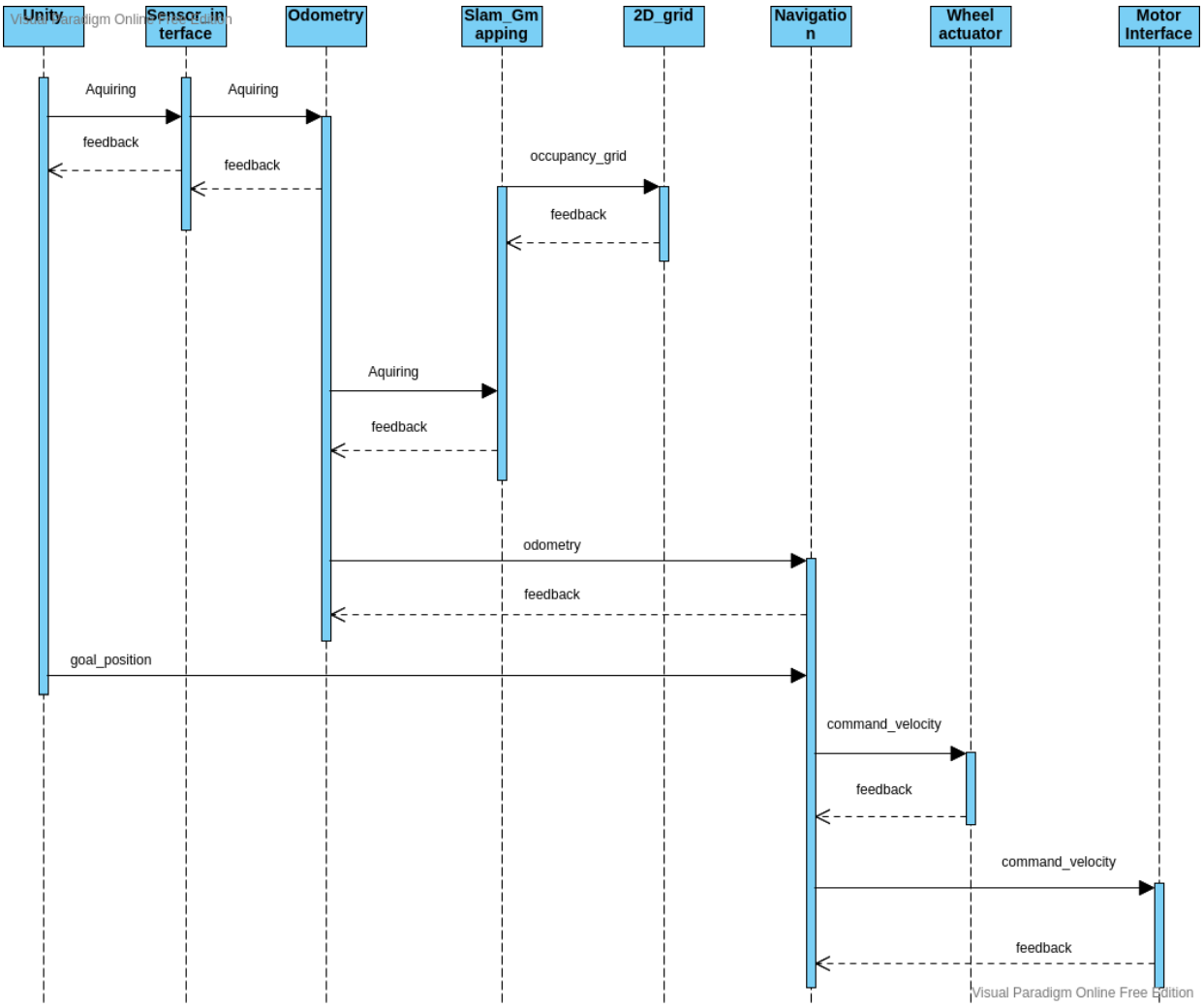


Figure 2.3: *The Temporal UML diagram*

3 - System Components

3.1 - Slam_gmapping

Slam_gmapping is a simulation for Localization And Mapping in ROS environment. This node has the power of creating a map on the /map topic of type nav_msgs OccupancyGrid. It is possible, since it takes the position of the robot via /tf topic which contains the transforms necessary, to relate frames for base and odometry. Furthermore it receives the LiDAR sensor as an input from /scan.

3.2 - Navigation

The navigation 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.

3.3 - Odometry

Thistake 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.

3.4 - User_interface

This manages to publish the goal position at the start of the simulation. In Unity scene the goal is marked with a red circle.

3.5 - Sensor_interface

This is provided by Unity scene to ROS. It is constituted by two publish subscribe design pattern.

- 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.

3.6 - Motor_interface

This controls 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 with avoiding the walls.

3.7 - LiDAR_sensor

This is the LiDAR sensor that scans the environment.

3.8 - Wheels_actuators

This is controls 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 - Preparations & Testing

4.1 - Installing

We tested the project in two different methods.

1 - Using the same machine (Linux) to implement the project.
2 - Using to different machines (Linux for ROS) (MacOS for Unity).

In order to install the system in the first method and run the code perfectly, we have to do the following:

1 - Install ROS-Noetic:

For all the instructions related the installation of ROS-Noetic, click **HERE.**

2 - Install Unity 2020.x.x

We used both Unity 2020.2.2/2020.3.13. You can download the desire Unity version from **HERE**.

3 - Clone our Repo into your ROS Workspace

By running this command in the Linux Terminal window " ***git clone https://github.com/mohammed-eldin/SOFAR_NAV_05*** "

After cloned the repo, navigate to config folder and open the params.yawl file to adjust the IP address of the ROS/Unity

4 - Installing the Navigation and gmapping libraries by using the following commands:

Navigation library " ***sudo apt-get install ros-noetic-navigation*** "
Mapping library " ***sudo apt-get install ros-noetic-openslam-gmapping*** "

5 - As the Husky robot is the robot that would scan the environment, we have to include it to our workspace:

In order to do so we have to clone the repo to our ROS workspace by using the following command " ***git clone https://github.com/husky/husky.git*** "

6 - Also, we have to install the LiDAR sensor by running the following command:

" ***sudo apt-get install ros-noetic-lms1xx*** "

The second method to prepare the system to run the project is as follows:

We follow the same procedures as the first method, but instead of using a same machine, we used two different machines "remotely" by using a middle ware to create a virtual network.

We used an application called LogMeIn Hamachi. After installing the application on both machines, one machine create a network and let the other machine join the created network. After joining the network, the application gives an IP address to each machine, so we use these IP addresses instead of the real ones in order to be on the same network “virtually”.

4.2 Running The Code

After finishing all the required to install, we can run the project as follows:

In the ROS side:

1 - Launch the slam_gmapping node by the command:

“ ***roslaunch mobile_robot_navigation_project gmapping.launch*** ”

2 - Launch the move_base node by the command:

“ ***roslaunch mobile_robot_navigation_project move_base2.launch*** ”

3 - Launch the Rviz node by the command:

“ ***roslaunch mobile_robot_navigation_project view_model.launch*** **“**

Also, uploading the model of the robot in the Rviz but opening it from Rviz from the repository config folder.

4 - Launch the Navigation node to start the communications between ROS and Unity by the following command line:

“ ***roslaunch mobile_robot_navigation_project navigation.launch*** ”

In the Unity side, we have to open the already implemented environment and add the IP address of ROS machine into the Scene section in Unity. If w use one machine, the IP address would be the same, but if we used two different machine with a virtual network, we would use the virtual IP address. To avoid any errors, wait until launching all the nodes on the ROS side then press the play button in the Unity in the middle top window. If the connection is running correct, we would have a window that give us the date received from the ROS and the data sent by the Unity on a window on the left corner.

4.3 - Testing & Result

As we mentioned before that we tested the project in two methods, on is using the same machine to run the ROS and Unity and the other is to use a virtual network to use two machines and to connect both machines to the same network in order to run the project.

The advantages of having a remote control is that you can use different machines in totally different places, but the disadvantages is that the time to scan the map takes a little bit of time because it depends on the speed of the internet.

The advantage to have them installed on one machine is that you scan in less time that if you use two different machines, but the disadvantage is that it's not practical, especially if you use it in the real robot.

The time that takes to scan the map in the same machine method might take up to 20 minutes, while using two machines might take double the time of the same machine. The following figure shows the result of scaling the map of Unity.

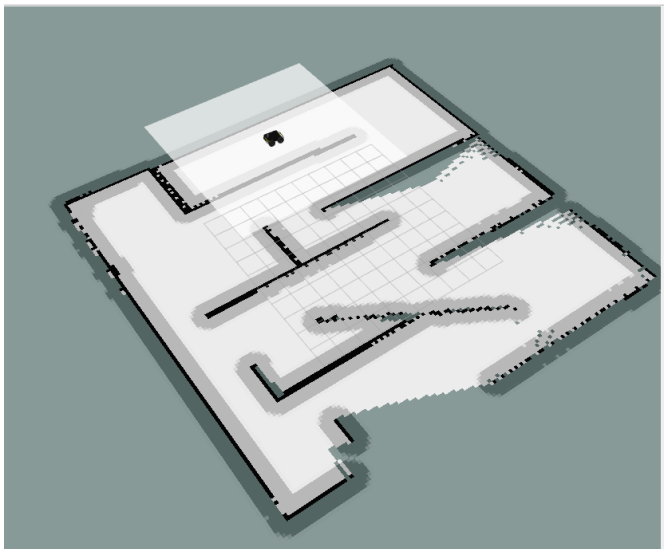


Figure 4.1: *Scanned map after reaching the desired end point in the Unity environment.*

While figure 4.2 shows the rat graph of the simulation.

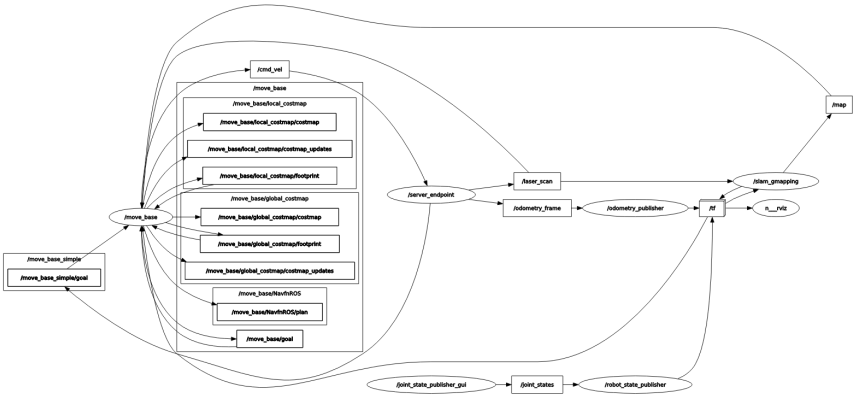


Figure 4.2: *The rqt_graph*