Autonomous Robotics: ROS Project 1

Arwed Paul Meinert

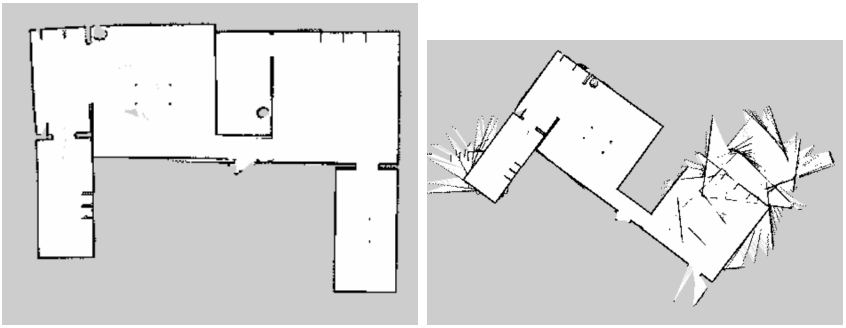
December 22, 2024

1. Task 1

To navigate the robot, the core needs to be started. After that, the environment is initialized using the button on the desktop, and the control can be opened using the rqt command followed by opening the remote control panel. Using the control panel, both the velocity and angular velocity can be adjusted, allowing the robot to be navigated within the house environment.

2. Task 2

The house can be mapped using the appropriate package from Robot Operating System (ROS). To do this, the robot needs to navigate through the house, automatically generating a map of the environment. Either the GMapping algorithm or the Hector Mapping algorithm can be used. The resulting maps are shown in Fig. 2.1.



(a) GMapping algorithm (b) Hectormapping algorithm

Figure 2.1: The different mapping algorithms

The results from both the GMapping algorithm (Fig. 2.1a) and the Hector Mapping algorithm (Fig. 2.1b) are very similar. It should be noted, however, that the GMapping algorithm was used to map the entire house, whereas the Hector Mapping algorithm was only applied to parts of the house. The results from the Hector Mapping algorithm are also less accurate when the robot moved quickly. This is evident from the artifacts in the map in Fig. 2.1b, particularly at the edges where the robot had to make sharp turns.

The artifacts in the map created by the GMapping algorithm in Fig. 2.1a are smaller, and the rooms are mapped correctly even when the robot was moving fast. Therefore, the map generated by the GMapping algorithm has been used for all other parts of this project.

Both algorithms subscribe to similar topics from the robot. The sensor data is obtained from the `/scan` topic, where data from the Light Detection and Ranging (LiDAR) sensor is stored. Additionally, the `/tf` topic is read, which contains the frame transformations. The output for both algorithms results in updates to the `/map` and `/map_metadata` topics. The GMapping algorithm also writes to the `/odom` topic. This topic contains data related to the robot's movement, such as wheel revolutions. This is not required by the Hector Mapping algorithm, as it relies solely on scan matching. The topics are summarized in Tab. 2.1.

The differences in the map shown in Fig. 2.1 can be explained by the missing odometer data in the Hector Mapping algorithm. Since the Hector Mapping algorithm does not have access to the expected velocity values, fast

SLAM Method	Input	Output
Gmapping	• `/scan`: Laser scan data from LIDAR • `/tf`: Transformations between frames (e.g., `base_link` and `map`)	• `/map`: Occupancy grid • `/map_metadata`: Metadata for the map • `/odom`: Odometry data
Hector Mapping	• `/scan`: Laser scan data from LIDAR • `/tf`: Transformations between frames	• `/map`: Occupancy grid • `/map_metadata`: Metadata for the map • No odometry required, relies solely on scan matching

Table 2.1: Comparison of GMapping and Hector Mapping Inputs and Outputs

movements of the robot in corners create artifacts. For slow movements, however, it works very well. In contrast, the GMapping algorithm, which has the ability to read movement data, can compare it to the sensor data and generate a more accurate map.

3. Task 3

The map created by the GMapping algorithm still contains some artifacts. Additionally, the door outside the house is open, but the robot is not supposed to move outside the house. Therefore, the map needs to be modified.

To achieve this, a simple graphics program such as GIMP can be used. This allows for removing pixels or adding black lines to block specific areas and prevent the robot from moving into them. The resulting map is shown in Fig. 3.1.

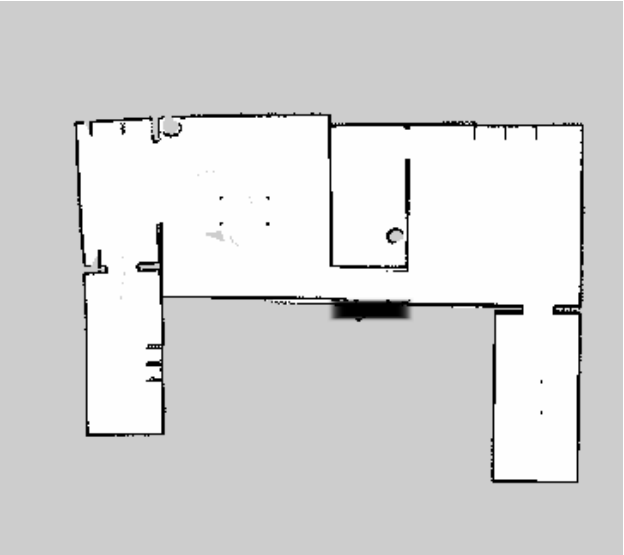


Figure 3.1: Modified map with blocked door and reduced noise.

4. Task 4

To perform navigation, a global planner is required. This needs to be installed and ensures that the robot finds a rough path to the target based on the saved map. In the local area, a local planner is used to circumvent objects detected by the LiDAR. The planned route is shown in the graphical window, where the parameters of the navigation can also be observed. All objects have an aura around them. The closer the robot is to an object, the higher the penalty for moving near it. The rate at which the penalty decreases can also be adjusted. This ensures that the robot only moves close to objects when it is absolutely necessary to reach the target. The navigation can be seen in Fig. 4.1 on the left side, with the simulated environment on the right side. The red areas indicate a higher cost for the robot to move through, while blue areas represent lower costs, as they are farther away from any object.

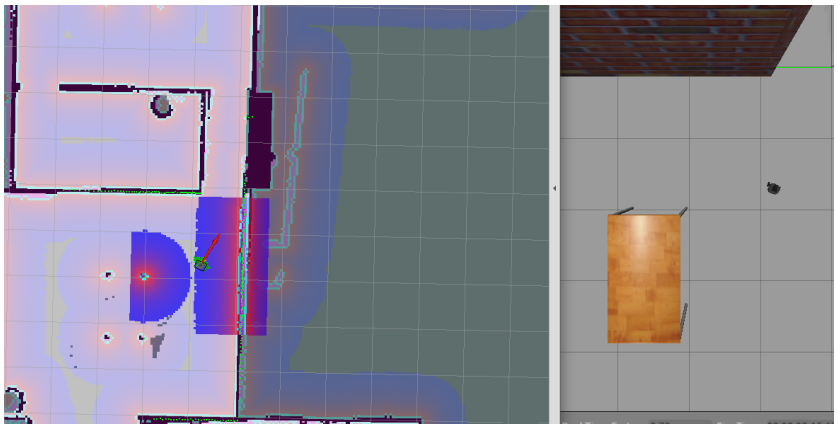


Figure 4.1: Robot navigating using ROS.

For the simulation to work, different topics are accessed by the navigation node. The structure is shown in Fig. 4.2. Since the map has already been created, it remains static and unchanged. It is published to the `/move_base` node. The parameters for the navigation are stored in the `/move_base` node. As this node also contains the navigation algorithm, the control commands are sent to the `/cmd_vel` topic. The simulation of the robot runs in the Gazebo node, which subscribes to the `/cmd_vel` topic. In the Gazebo simulation, the LiDAR sensor data, position data, and odometer data are generated and evaluated. Since the navigation algorithm also requires access to the LiDAR data, the `move_base` node subscribes to the `/scan` topic.

The initial pose and the data from the `/scan` topic are used to create the `/amcl` node, which is responsible for localizing the robot. All the data is combined and published to the `/tf` topic, which describes the transformation of the robot with respect to different frames.

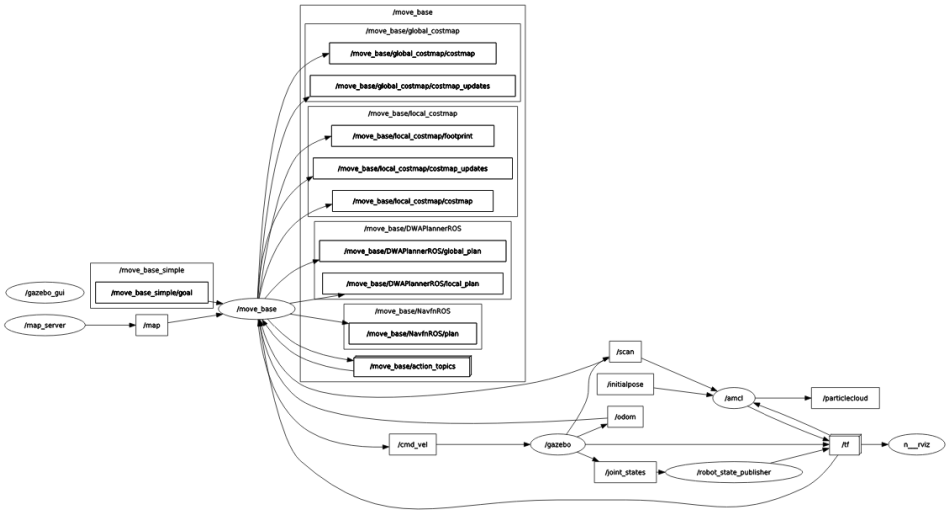


Figure 4.2: Node graph of the GMapping algorithm.

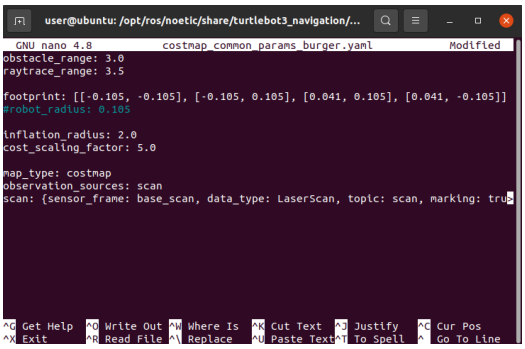
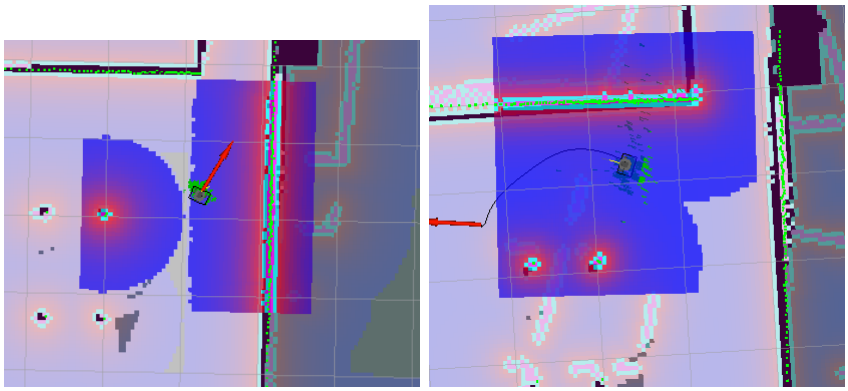


Figure 4.3: Parameter set of the navigation.

The Transformation (TF) tree is updated continuously. This is necessary because the robot could be moving at all times, meaning that the transformation of the robot can change constantly. An accurate transformation is essential, as it describes the position of the robot.

The parameters for navigation are stored in a file accessed by the navigation node. To change the parameters in the `/move_base` topic shown in Fig. 4.2, this file can be modified. It can be used to optimize the robot's path planning. The different parameters are displayed in Fig. 4.3. The inflation radius defines the size of the area around an obstacle where a cost penalty is applied to the path. The cost scaling factor determines the rate at which the cost penalty increases as the robot gets closer to the obstacle. A higher cost scaling factor results in a steeper increase in the penalty near the obstacle, making paths closer to obstacles less desirable.

The change in the navigation can be seen in Fig. 4.4. In Fig. 4.4a, the default parameters for the cost map were used. In Fig. 4.4b, the parameters shown in Fig. 4.3 were applied. The objects have a larger area where the cost is penalized, but the area closer to the objects is slightly less penalized. This results in the robot moving closer to the objects, potentially leading to a shorter path.



(a) Default parameters. (b) Changed parameters.

Figure 4.4: Different parameters for the navigation algorithm.

5. Acronyms

LiDAR Light Detection and Ranging. 2, 5

ROS Robot Operating System. 2, 5

TF Transformation. 6

5.1 Use of Generative AI

AI was used for spelling and grammar checks as well as for the creation of tables and debugging in the LaTeXsyntax.