SPC 418 – Control for Autonomous Vehicles
Zewail City for Science & Technology

Autonomous Mapping and Navigation Using ROS

Final Project Report

Mohammed El-Sayed | 201501508

Nadine Amr | 201501328

Omar Yasser | 201500831

Supervised by:

Dr. Mostafa El-Shafei

Table of Contents

Abstract: *In this project, the robot, Husky, is spawned in 5 different environments. In each of them, the robot uses the gmapping algorithm and autonomously maps the environments using its laser scanner. Next, the robot is prompted to navigate from its initial position to a final one across the map. The robot successfully executes those missions; building accurate maps and navigating using the shortest path.*

I. Introduction

A robot does not possess natural senses like human beings do. Human beings get information about their surroundings through vision and other sensing powers. A robot cannot explore an unknown environment unless it is provided with some sensing sources to get information about the environment. Different kinds of sensors such as sonars, odometers, laser range finders, inertial measurement units (IMU), global positioning system (GPS) and cameras are used to make a robot capable of sensing a wide range of environments [1]. The map of environment is a basic need for a robot to perform actions like moving room to room, picking an object from one place and taking it to another one. To perform such actions, the robot should not only know about the environment but while it is moving, but it should also be aware of its own location in that environment [2].

II. Problem Statement

The problem we are tackling can be summarized in the following points:

1. In 5 different simulated environments on Gazebo, let the robot map the environment while simultaneously localizing itself in it (SLAM – Simultaneous Localization and Mapping).
2. Assign a certain goal position for the robot to navigate to in the mapped environment, after localizing itself in it.

The robot used in the simulation is Husky and the tools used are Gazebo and Rviz on ROS.

III. Algorithms

A. Mapping Algorithms

1. GMapping

The robot needs an algorithm with which to map the environment it is in. GMapping is a highly efficient particle filter algorithm that learns grid maps from laser range data. Particle filters have been introduced as effective means to solve the SLAM problem [3].

In the GMapping algorithm, each particle in the particle filter carries an individual map of the environment and hence contributes to generating a collective output map. The GMapping algorithm estimates the map as well as the robot's position [4].

2. Autonomous Mapping

In order for the robot to build the map of the terrain it is in, it ought to have a mechanism with which to explore it. Instead of manually driving the robot through the terrain, we wrote a simple node to drive it autonomously.

This node, named "autonomous_mapping" subscribes to the laser scanner topic of our robot, "/scan", and publishes on the topic of the robot's velocity, "/cmd_vel", accordingly.

The algorithm is as follows:

1. From all the laser scanner returned beams, the middle 16% of the beams are considered.
2. If any of the right 8% of the beams detected an obstacle at a distance of less than 1.2 meters, a flag with an initial value of zero is set to 1.
3. If any of the left 8% of the beams detected an obstacle at a distance of less than 1.2 meters, the flag is set to 2.
4. The value of the flag is then checked; if it is 1, the robot is prompted to turn to the left. This is achieved by publishing on the "/cmd_vel" topic a twist message with a linear x velocity of 0.1 and an angular z velocity of 10.
5. If the value of the flag is 2, the robot is prompted to turn to the right. This is achieved by publishing on the "/cmd_vel" topic a twist message with a linear x velocity of 0.1 and an angular z velocity of -10.
6. If the value of the flag is still 0, the robot is prompted to keep moving forward. This is achieved by publishing on the "/cmd_vel" topic a twist message with a linear x velocity of 0.5 and an angular z velocity of 0.

Note that this algorithm lacks randomness, which means the robot might not be able to fully map the terrain as it might never reach some parts of it. Consequently, we further improved the algorithm by editing step 6 as follows:

If the value of the flag is still 0, a random number from 0 to 25 is generated. If the random number is anything but 0, the robot is prompted to keep moving forward by publishing on the "/cmd_vel" topic a twist message with a linear x velocity of 0.5 and an angular z velocity of 0. Conversely, if the random number is 0, the robot is prompted to turn either left or right by publishing on the "/cmd_vel" topic a twist message with a linear x velocity of 0.1 and an angular z velocity of 10 or -10. To choose whether the robot will turn left or right, another random number with the value of either 0 or 1 is generated and the decision is taken accordingly.

Note that the algorithm contains many parameters including the percentage of laser beams checked, the minimum allowed distance between the robot and the obstacles, as well as the linear and angular velocities of the robot in the different cases. All these parameters were tuned in order to achieve the best results in simulation.

B. Navigation Algorithms

1. <u>AMCL</u>

In order for the robot to navigate in a provided map, it must first be able to localize itself in it. This can be done using the AMCL algorithm. The AMCL algorithm, or the adaptive Monte Carlo localization algorithm, is a probabilistic localization algorithm which uses a particle filter to determine the robot position and orientation in a given map [5].

The AMCL algorithm takes as input the map, the robot's laser scans, the mean and covariance of the robot's initial position and the transform messages. The algorithm outputs the mean and covariance of the robot's estimated position, the estimated positions of the particles in the filter, and the transforms [5][6].

Note that the AMCL algorithm has a lot of parameters, like the minimum and maximum number of particles that we set to be their default values.

2. <u>Move Base</u>

After the robot has successfully localized itself in the map, we would like it to navigate from the start point A to a final point B at the other end of the map. The means by which this communication is achieved is through the move_base package. Given a goal position/orientation in the map, this package allows the robot to reach it. It links a global and a local planner, and maintains the costmaps of each of them in order to achieve its tasks [6][7].

The input is the desired goal position and the output is the velocity that drives the robot to reach its goal. This velocity is chosen based on a plan calculated such that the robot arrives at its goal by means of the shortest path. The move_base node can also carry out recovery behaviors in case the robot is thought to be stuck. These behaviors include performing in-place rotations or clearing the costmap. The node also provides the robot status or where the robot is in the process; whether it is generating a new plan, being stuck, or have arrived at the goal destination [7].

3. Simple Navigation Goals

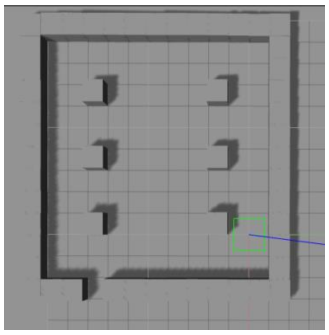
As mentioned, the input to the move_base package is a goal position.This goal position is assigned to the robot either using a tool like rviz, or through another node. Here, we use the simple_navigation_goals node to send the desired goal position (Point B).

The simple_navigation_goals node is a client to the action provided by the move_base node. It provides the move_base node with the position message with the type move_base_msgs::MoveBaseGoal. It eventually checks the success/failure of the navigation [8].

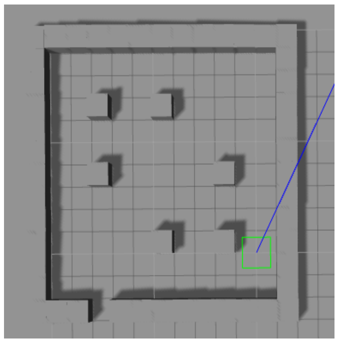
IV. Maps

The following maps were built in Gazebo:

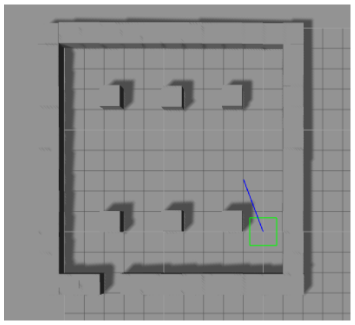
- Map 1:



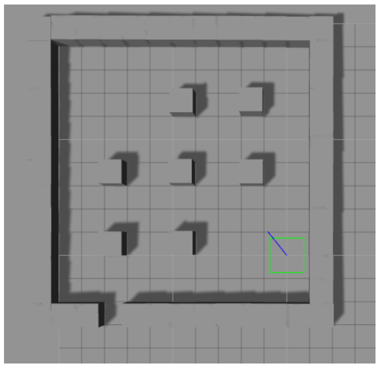
- Map 2:



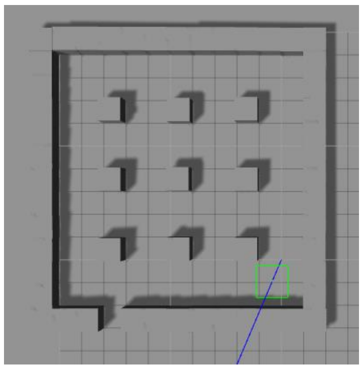
- Map 3:



- Map 4:



- Map 5:

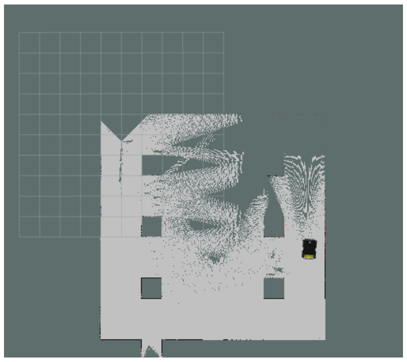


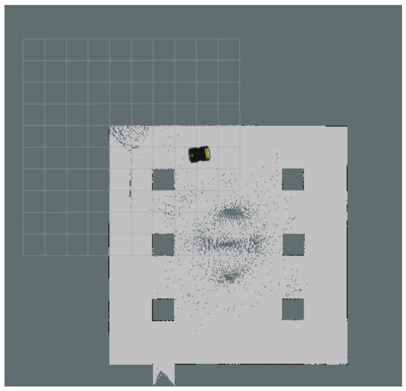
V. Results

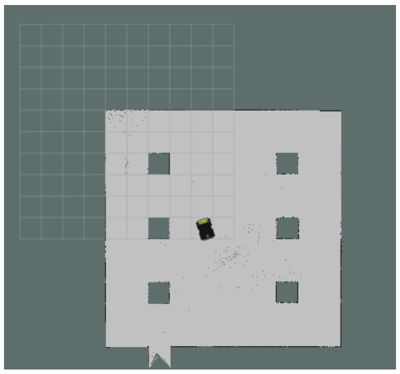
A. Mapping Results

1. <u>During Mapping</u>

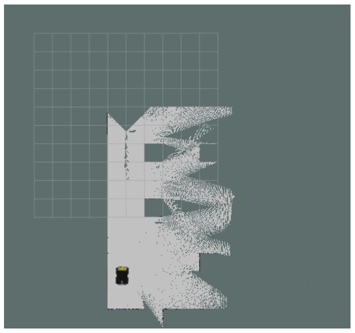
- Map 1:

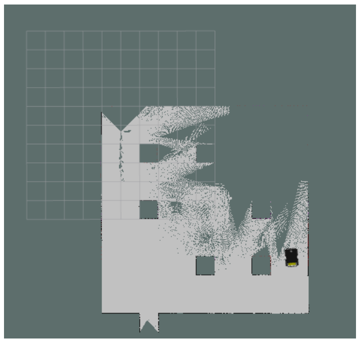


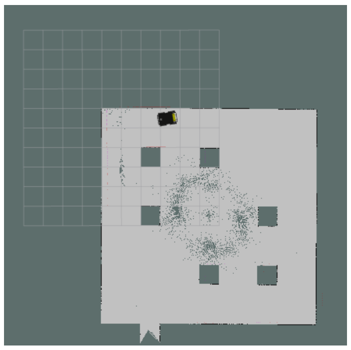




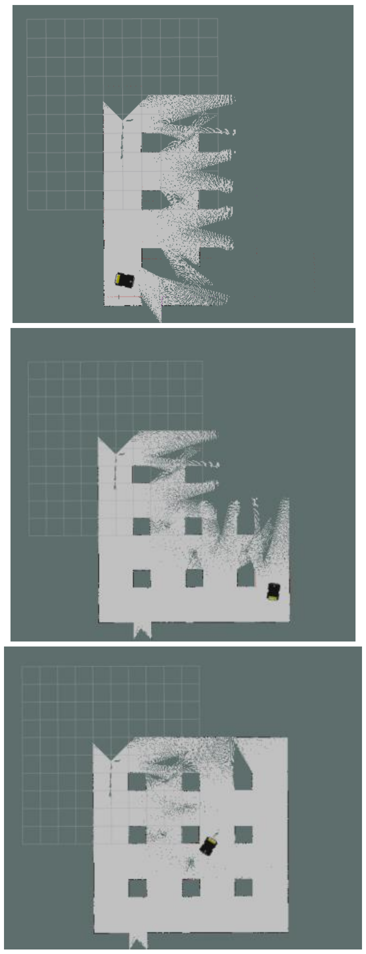
- Map 2:







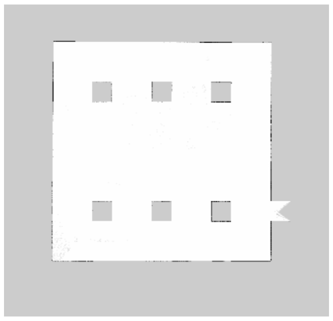
- Map 5:



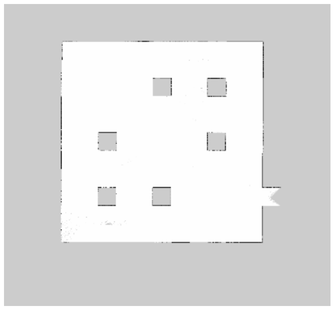
2. Generated Maps

The generated maps are saved in the home directory as 2 files; one is a pgm file (image file), and the other is a ymal file. The yaml file contains the some parameters and information about the map so that it could later be loaded in rviz.

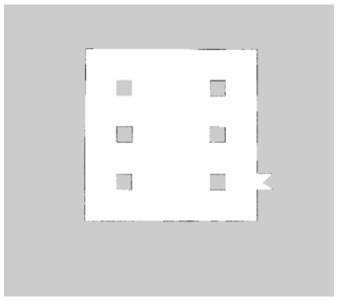
- Map 1:



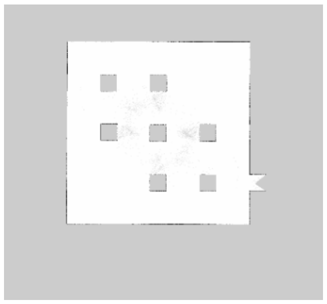
- Map 2:



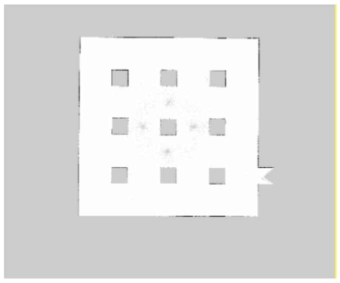
- Map 3:



- Map 4:

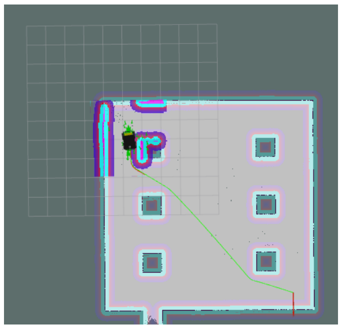


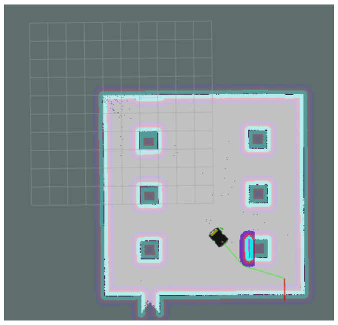
- Map 5:

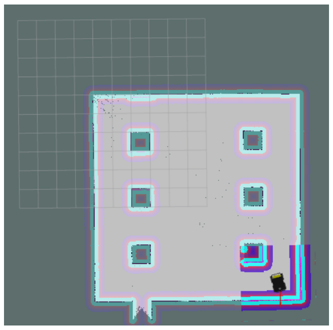


B. Navigation Results

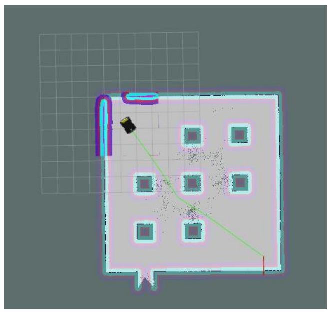
- Map 1:

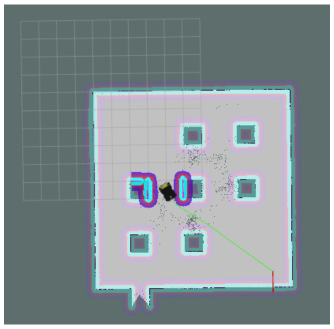


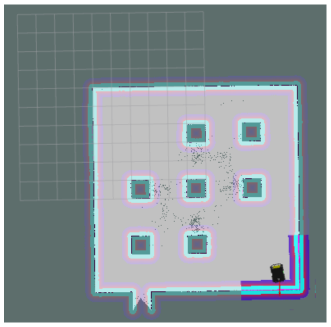




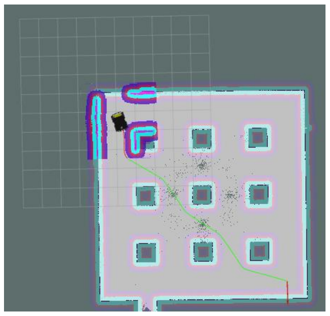
- Map 4:

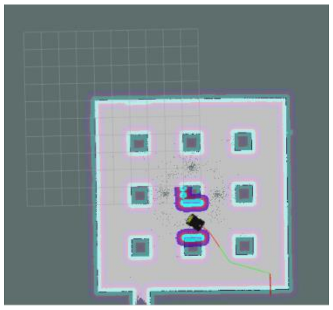


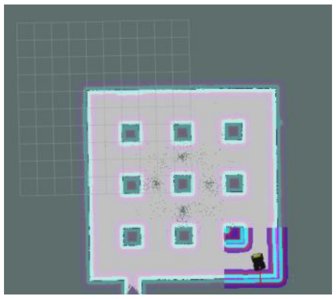




- Map 5:







VI. Conclusion & Future Work

As evident, Husky is a very robust robot whose laser scanner is quite powerful. It successfully built the maps of the different environments, localized itself, an performed navigation from point A to point B.

For future work, a more efficient algorithm for randomly exploring the environment is to be used. One strong candidate algorithm is the frontier algorithm. Another possible enhancement is using 2 robots to build together the same map. This would indeed save considerable amount of time.

VII. References

[1] A. Pajaziti, "SLAM Map Building and Navigation via ROS", *International Journal of Intelligent Systems and Applications in Engineering*, vol. 2, no. 4, p. 71, 2014. Available: 10.18201/ijisae.08103.

[2] S. Zaman, W. Slany and G. Steinbauer, "ROS-based mapping, localization and autonomous navigation using a Pioneer 3-DX robot and their relevant issues", *2011 Saudi International Electronics, Communications and Photonics Conference (SIECPC)*, 2011. Available: 10.1109/siecpc.2011.5876943.

[3] G. Grisetti, C. Stachniss and W. Burgard, "Improved Techniques for Grid Mapping With Rao-Blackwellized Particle Filters", *IEEE Transactions on Robotics*, vol. 23, no. 1, pp. 34-46, 2007. Available: 10.1109/tro.2006.889486.

[4] S. Thrun, W. Burgard and D. Fox, *Probabilistic robotics*. Cambridge, Mass.: MIT Press, 2010.

[5] "amcl - ROS Wiki", *Wiki.ros.org*. [Online]. Available: http://wiki.ros.org/amcl.

[6] L. Joseph, *ROS robotics projects*. Birmingham, UK: Packt Publishing, 2017.

[7] "move_base - ROS Wiki", *Wiki.ros.org*. [Online]. Available: http://wiki.ros.org/move_base.

[8] "navigation/Tutorials/SendingSimpleGoals - ROS Wiki", *Wiki.ros.org*. [Online]. Available: http://wiki.ros.org/navigation/Tutorials/SendingSimpleGoals.