Comparison of Multiple Slam Algorithms using ROS

Saandeep Sreerambatla

Department of Electrical Engineering, Case Western Reserve University, Cleveland, OH, USA
Email: *sxs2452[at]case. edu*

Abstract: ***This paper presents a comparative analysis of mapping and localization techniques for a single robot in a home environment. The TurtleBot3 robot, simulated in the Gazebo environment, is employed to generate maps, and the visualization of the mapping process is performed using RViz. Initially, a map is created by moving the robot using teleoperation and employing the Gmapping algorithm. The movement parameters are recorded and replayed to generate a map using the Hector SLAM algorithm. The final maps produced by these two algorithms are compared by overlaying them on the actual map, highlighting their respective advantages and limitations.***

Keywords: mapping techniques, localization techniques, TurtleBot3 robot, Gmapping algorithm, Hector SLAM algorithm

1. Introduction

Simultaneous Localization and Mapping (SLAM) is a critical functionality in robotics, involving the continuous estimation and adjustment of a map based on data from a mobile robot, while simultaneously positioning the robot on this map. An effective and precise SLAM algorithm is essential for navigation and other high - level tasks. Different SLAM algorithms utilize various techniques for mapping and localization, relying on diverse sensory information equipped on the robot for accurate map representation. This study aims to compare the Gmapping and Hector SLAM algorithms to understand their respective strengths and weaknesses. The project leverages the Robot Operating System (ROS), a flexible framework that controls robotic components from a computer. ROS consists of various nodes that communicate based on a publisher/subscriber model, offering robust tools, libraries, and conventions to facilitate the creation of reliable robotic behaviors.

2. Simultaneous Localization and Mapping (SLAM)

Simultaneous localization and mapping (SLAM) is the computational problem of constructing or updating a map of an unknown environment while simultaneously keeping track of an agent's location within it. While this initially appears to be a chicken - and - egg problem there are several algorithms known for solving it, at least approximately, in tractable time for certain environments [3]. In probabilistic view, the SLAM problem has two main kinds which are online SLAM and full SLAM. Online SLAM involves estimating the posterior over a brief time that is: p (xt, m | z1: t; u1: t) where xt is the position of robot at time t, m is the map, z1: t is a set of measurement data acquired from 1 to t, and u1: t is the sequences of control data from time 1 to t. Online SLAM eliminates the previous measurements and controls once it has estimated the map [3]. The other form, full SLAM involves estimating the posterior over the entire path the robot has travelled, x1: t: p (x1: t, m | z1: t; u1: t) Unlike online SLAM, full SLAM doesn't discard the previous measurement and control data.

3. Gmapping

Gmapping solves the Simultaneous Localization and Mapping (SLAM) problem using the Particle Filter (PF) technique, which is a method for model - based estimation. In SLAM, we estimate two things: the map and the robot's position within the map. Each particle in the PF is assumed to be a candidate solution to the problem. By using multiple particles, we estimate the true probability distribution through Importance Sampling.

The key idea of using a particle filter is to estimate a joint posterior of a map p (x1: t, m∣z1: t, u1: t−1) and the trajectory $x1$: t=$x1$, …, xtx1: t=x1, …, xt of the robot. This estimation is performed given the observations $z1$: t=$z1$, …, zt and the odometry measurements u1: t−1=u1, …, ut−1 obtained by the mobile robot.

The Rao - Blackwellized particle filter for SLAM makes use of the following factorization:
p (x1: t, m∣z1: t, u1: t−1) =p (m∣x1: t, z1: t) ⋅p (x1: t∣z1: t, u1: t−1)

This factorization allows us to first estimate only the trajectory of the robot and then compute the map given that trajectory. Since the map strongly depends on the pose estimate of the robot, this approach offers efficient computation. This technique is often referred to as Rao - Blackwellization.

4. Hectorslam

Hector mapping leverages the high update rate of modern LIDAR systems, such as the Hokuyo UTM - 30LX, and provides 2D pose estimates at the scan rate of the sensors (40Hz for the UTM - 30LX). While the system does not offer explicit loop closing capability, it is sufficiently accurate for many real - world scenarios. The system has been successfully used on Unmanned Ground Robots, Unmanned Surface Vehicles, and Handheld Mapping Devices.

Hector SLAM is considered to be using the state - of - the - art particle filter, which works well even in the absence of odometry information. Additionally, it performs effectively in scenarios where roll, pitch, and yaw are present in the system.

5. Experimental Model

5.1 Software Development

In this project, ROS (Robot Operating System) is used to control the TurtleBot. The bot scans the area through a laser scan and reports the environment. The laptop represents the movement of the bot in Gazebo, and the visualization is recorded in RViz. A Linux OS on an Ubuntu desktop is required to run ROS. ROS utilizes the OS's process management system, user interface, file system, and programming utilities. The mapping will be shown in the RViz application, as depicted in Figure 1.

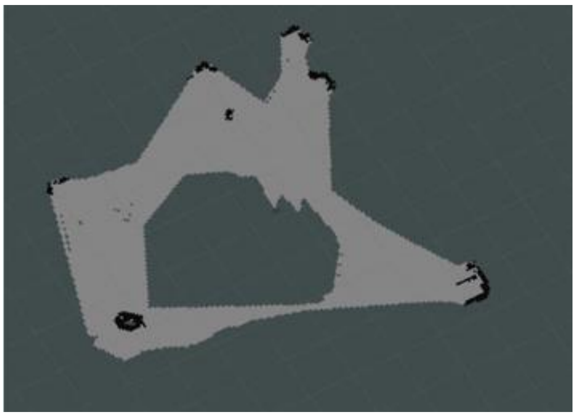


Figure 1: Sample rviz map

Robotic Operating System (ROS) is a system for controlling robotic components from a computer. The ROS system is comprised of a number of independent nodes, each of which communicates with the other nodes using a publish/subscribe messaging model. ROS is a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms. ROS was built from the ground up to encourage collaborative robotics software development.

5.2 Implementation

In this project, a world was created using existing Gazebo models, and a TurtleBot [4] was introduced into the world. By using the teleoperation (teleop) feature of ROS, the robot was moved around the environment to perform SLAM. The robot utilized laser scans to understand the environment and localize obstacles. SLAM was performed using the inbuilt Gmapping algorithm, which employed default parameters except for the updated Urange. The robot's movements (linear velocity and angular velocity) were controlled via teleop methods.

The robot's movements were recorded from beginning to end using rosbag. Rosbag is a subscriber that subscribes to all the movements of the robot in RViz2 and creates a record every second. This record was saved for use with the Gmapping algorithm. The next step in this project was to playback the robot's movements, but this time using a different mapping technique: Hector SLAM. Hector SLAM moved according to the rosbag and performed SLAM on the environment. Figure 2 shows the map of the actual environment.

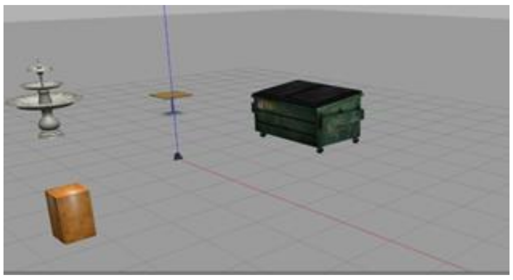


Figure 2: Actual Environment

6. Validation

In this project, the centers of the images from Gazebo were taken, and the predicted centers were obtained using the publish point feature in RViz for the centers of the objects. The root mean squared error (RMSE) for the distance of each object was calculated, as well as the cumulative error for all the objects. A similar analysis was performed for both mapping techniques, Gmapping and Hector SLAM. Below, we present the graph representations of the error plots for the algorithms. The error computation was done in Python, and the graphs were generated using Matplotlib.

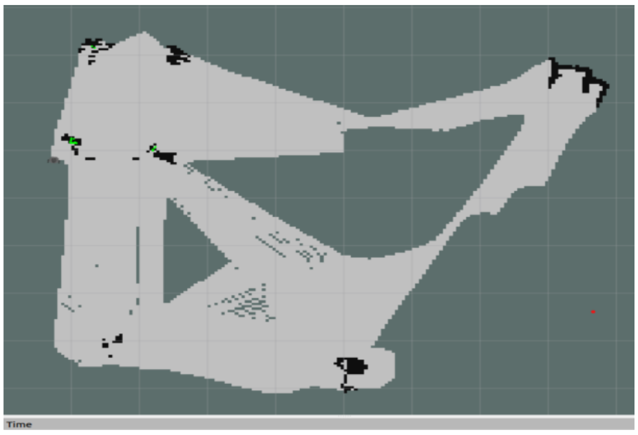


Figure 3: Map produced by Gmapping Algorithm

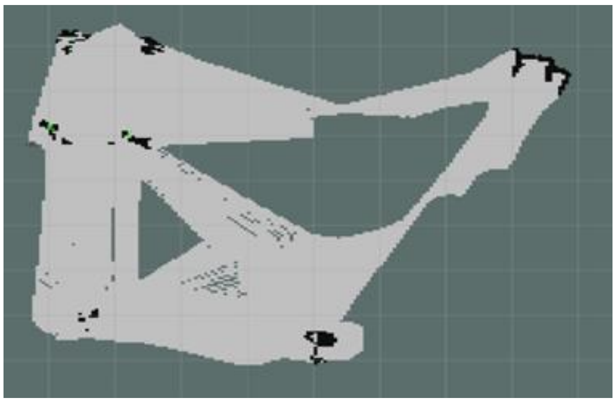


Figure 4: Map produced by Hector Slam

7. Results

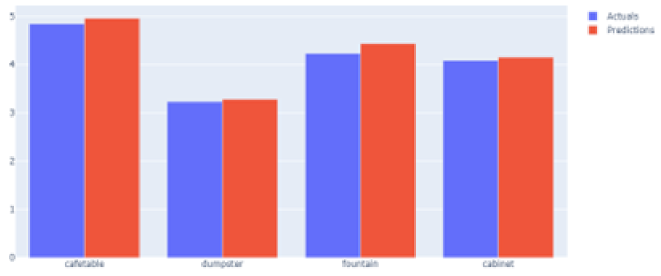


Figure 5: Results for Gmapping actual vs predicted

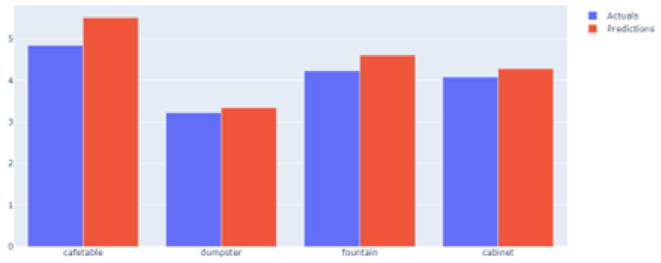
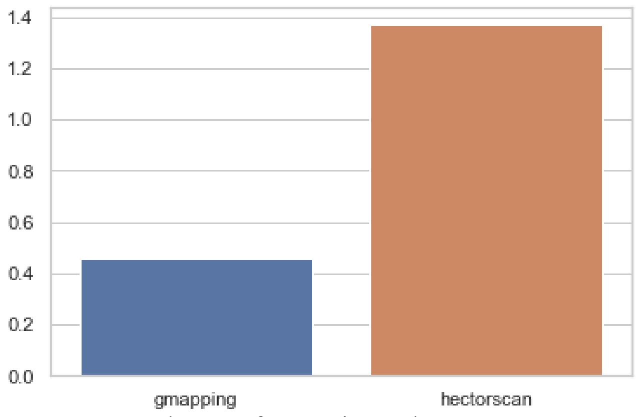


Figure 6: Results for Hector slam actual vs predicted

The Gmapping algorithm performed slightly better compared to the Hector SLAM algorithm. The next steps would involve introducing a more complicated world and performing the mapping algorithms to validate the behavior of the system. Another step could be to perform Hector SLAM first to record the rosbag and then playback for Gmapping to assess the performance.

8. Conclusion

It can be concluded that performance testing of multiple SLAM algorithms was successfully conducted in ROS using TurtleBot and Gazebo. Both Gmapping and Hector SLAM performed well, with Gmapping showing slightly better results compared to Hector SLAM. However, several assumptions were made in this experiment. Firstly, it was assumed that all obstacles the robot might encounter are stationary. Secondly, friction was not considered. During real - time application, the predicted state of the robot based on constant velocity may be inaccurate.



Total error of gmapping vs hectorscan

Acknowledgement
I would like to express my sincere gratitude to Prof. M. Cenk Cavusoglu for his guidance and patience.

References

[1] http: //www2. informatik. uni - freiburg. de/ stachnis/pdf/grisetti07tro. pdf
[2] S. Thrun, W. Burgard, and D. Fox, “Probabilistic robotics (Intelligent Robot and Autonomous agents) ”, MIT Press, 2005
[3] https: //emanual. robotis. com/docs/en/platform/turtlebot3/overview/
[4] http: //wiki. ros. org [2]rviz
[5] https: //answers. ros. org/question/239143/how - does - gmappingwork/