UNITED STATES MILITARY ACADEMY

LAB 5 REPORT - NAVIGATION AND PATH PLANNING WITH THE TURTLEBOT

EE487: EMBEDDED SYSTEMS DEVELOPMENT
SECTION T2

COL KORPELA

BY

CDT NICHOLAS E. LIEBERS '23, C-2

WEST POINT, NY 10996

12 APR 2022

NEL

___ DOCUMENTATION IDENTIFIES ALL SOURCES USED AND ASSISTANCE RECEIVED IN COMPLETING THIS ASSIGNMENT.

___ NO SOURCES WERE USED OR ASSISTANCE RECEIVED IN COMPLETING THIS ASSIGNMENT.

SIGNATURE: LIEBERS.NICHOLAS.EDWARD.1557580448 Digitally signed by LIEBERS.NICHOLAS.EDWARD.1557580448 Date: 2022.04.12 07:57:01 -04'00'

Contents

1. Introduction

1.1. Purpose

The purpose of this lab is to implement and test the Navigation and Path Planning functionality on the Turtlebot3. The purpose was accomplished, and the following learning objectives were met:

1.2. Learning Objectives

1. Utilize the Gazebo simulation environment.
2. Utilize the RViz visualization environment.
3. Learn the principles of Robot Navigation and Path Planning.
4. Research and compare/contrast at least 3 Navigation techniques.
5. Utilize the Linux operating system, including its command line interface (CLI).

2. Theory

In order to complete this lab, we must clearly define what Navigation and Path Planning is, and we must research different methods 3 different algorithms to understand the field.

2.1. What is Navigation and Path Planning with respect to the Turtlebot3?

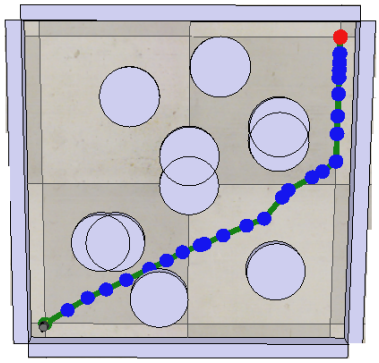

Figure 1: An example of how a robot may navigate through its environment. The red point is the starting position the green point is the desired ending position, and the blue line is the path the robot takes [7].

Navigation is simply moving the Turtlebot3 from one specified location to another given a certain environment [3]. Path Planning facilitates the process of navigation. When conducting Navigation and Path Planning, the Turtlebot3 must first define a path; then, the Turtlebot3 must follow that path to its destination. In Fig. 1, demonstrates how a path finding algorithm could find a path from point red to point green.

2.2. Navigation and Path Planning Algorithms

There are three Navigation and Path Planning Algorithms that will be discussed in this report: Dijkstra's Algorithm, A* Algorithm, and D* Algorithm. All three of these algorithms allow for a robot to determine the shorts path from a position to a destination; however, they all have different benefits and weaknesses that will be evaluated below.

2.2.1. Dijkstra's Algorithm

Dijkstra's Algorithm is a greedy algorithm that determines the shortest path from a start node to an end node by computing the shortest path to any given node–therefore computing the shortest path to the final node [5]. Versions of Dijkstra have been used in applications like Google Maps to determine optimal routes based on traffic, and Dijkstra's Algorithm ensures a shortest path is found in $O(n^2)$ time [5]. Unfortunately, Dijkstra is memory intensive since it is required to know/save all nodes in a network–no matter how large. Also, since Dijkstra can only deal with positive weights, any graphs with negative weights cannot use Dijkstra. Additionally, Dijkstra's Algorithm only works when dealing with a static environment where the obstacle are not ever moving [5]. This characteristic of Dijkstra elicited a need for alternatives.

2.2.2. A* Algorithm

A* Algorithm is exceptionally similar to Dijkstra's Algorithm, but A* contains an additional heuristic to focus its search towards the "most promising" nodes that will lead to a shortest path [5]. For example, when searching for the shortest path between two points A and B, rather than searching for the shortest path in all directions, A* will focus its search in the general direction of point B. Different heuristic methods that can be used are Euclidean distance, Manhattan distance, and Octile distance between two points. Other versions of A*, such as Hybrid A*, use the kinematics of an robot to predict the motion of the robot. This additional heuristic enables the A* algorithm to continue to calculate an optimal path while using the robot's positional data to help refine the algorithm's focus [5].

2.2.3. D* Algorithm

The D* Algorithm, also known as Dynamic A*, accounts for a dynamic environment, and due to this property, D* is particularly interesting for those dealing with autonomous vehicles [5]. When using D*, the cost map to get from two points A and B is not static–meaning D* updates these maps during run time. Since a dynamic environment can remove previous paths as obstacles change their location, D* must be able to update and adjust as the environment changes. This feature is a large change compared to A* and Dijkstra since both of those algorithms fail to account for a changing environment. The D* Algorithm contains other variants such as D* Lite and Enhanced D* Lite. These other algorithms attempt to fix issues associated with D*–such as large memory consumption, sometimes failing to avoid complicated obstacles,

failing to move through two obstacles, and failing to traverse around sharp corners [5]. D* is being used in a wide variety of robotic systems, and it is used in both indoor and outdoor environments [5].

2.3. Defining Costmaps, Adaptive Monte Carlo Localization, and Dynamic Window Approach

While path planning is important for navigation, the Turtlebot3 contains various sensors that enable it to determine its location in space, detect its orientation, detect close obstacles, and then avoid these obstacles. The following methods allow the Turtlebot3 to avoid obstacle in close proximity, thus ensuring it can navigate its environment. In order to orient readers to these concepts, the following ideas will be explained and illustrated below.

2.3.1. Costmaps

A costmap determines where obstacles are, the size of the obstacles, and where the Turltebot3 can move based on the obstacle information. There are two types of costmaps: a global and local costmap. A global costmap helps to setup a cost plan for the whole environment, while a local costmap is used for obstacle avoidance in the intimate area around the robot [6]. Costmaps are determined using an assortment of sensors on the Turtlebot3.

2.3.2. Adaptive Monte Carlo Localization

Adaptive Monte Carlo Localization (AMCL) is a pose estimating algorithm that wants to determine its x, y, and θ in an environment. It is based of the Monte Carlo Localization algorithm, but offers some benefits. As the Turtlebot3 is navigating throughout its environment, the AMCL predicts where the Turtlebot3 is at at any given time. As the robot navigates, the AMCL allows the robot to understand where it is in the environment [6]. This is especially important so the Turtlebot3 knows it is on the correct path.

2.3.3. Dynamic Window Approach

The Dynamic Window Approach (DWA) to object avoidance involves controlling the robot's movements using velocity vectors. Rather than concerning itself with x and y coordinates, the DWA only works with respect to velocities. With this different approach, DWA can calculate the optimal velocity (linear and

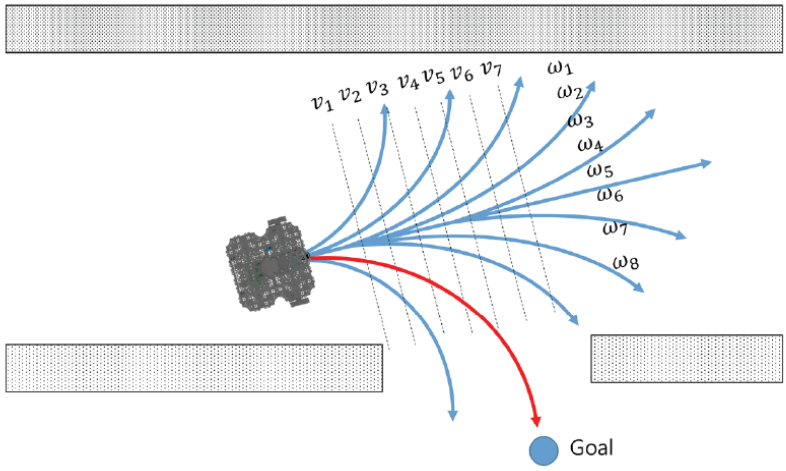

Figure 2: The translational and angular velocity of the Turtlebot3 [6].

angular) to prevent collisions and reach its goal. In Fig. 2, notice how the Turtlebot3 avoids the obstacle by simply adjusting its velocities in just the right manner to reach its goal [6].

3. Experimental Setup and Procedure

3.1. Required Equipment

1. Computer running Linux with proper software installed to interface with Turtlebot3[1].
2. A Turtlebot3.
3. Turtlebot3 battery.

3.2. Software Setup

This lab runs ROS on a computer and on the Turtlebot3. The following command will install required packages, Gmapping, and the ROS Navigation packages.

```
$ sudo apt-get install ros-noetic-joy ros-noetic-teleop-twist-joy \
  ros-noetic-teleop-twist-keyboard ros-noetic-laser-proc \
  ros-noetic-rgbd-launch ros-noetic-rosserial-arduino \
  ros-noetic-rosserial-python ros-noetic-rosserial-client \
  ros-noetic-rosserial-msgs ros-noetic-amcl ros-noetic-map-server \
  ros-noetic-move-base ros-noetic-urdf ros-noetic-xacro \
  ros-noetic-compressed-image-transport ros-noetic-rqt* ros-noetic-rviz \
  ros-noetic-gmapping ros-noetic-navigation ros-noetic-interactive-markers
```

[3]

3.2.1. ROS Setup on the TurtleBot3

The Turtlebot3 comes fully equipped with ROS; however, some configuration was required in order to get full functionality.

The TurtleBot3 Setup guide on the West Point Robotics Github offers configuration details for how we setup our robot [4].

[1] see `https://emanual.robotis.com/docs/en/platform/turtlebot3/quick-start/#pc-setup` for required packages.

In order to navigate and traverse an environment, the TurtleBot3 needs a map to reference. Using the gmapping tools installed on the Turtlebot3, a map can be generated and saved for later use. As stated in earlier labs, to launch gMapping, the following command was run:

```
$ roslaunch turtlebot3_slam turtlebot3_slam.launch slam_methods:=gmapping
```

[3].

To save the file of the environment, the following command was run:

```
$ rosrun map_server map_saver -f ~/map
```

[3].

To run the navigation script to allow the Turtlebot3 to navigate its environment, run the following from the computer that is running roscore:

```
$ roslaunch turtlebot3_navigation turtlebot3_navigation.launch map_file:=$HOME/map.yaml
```

[3].

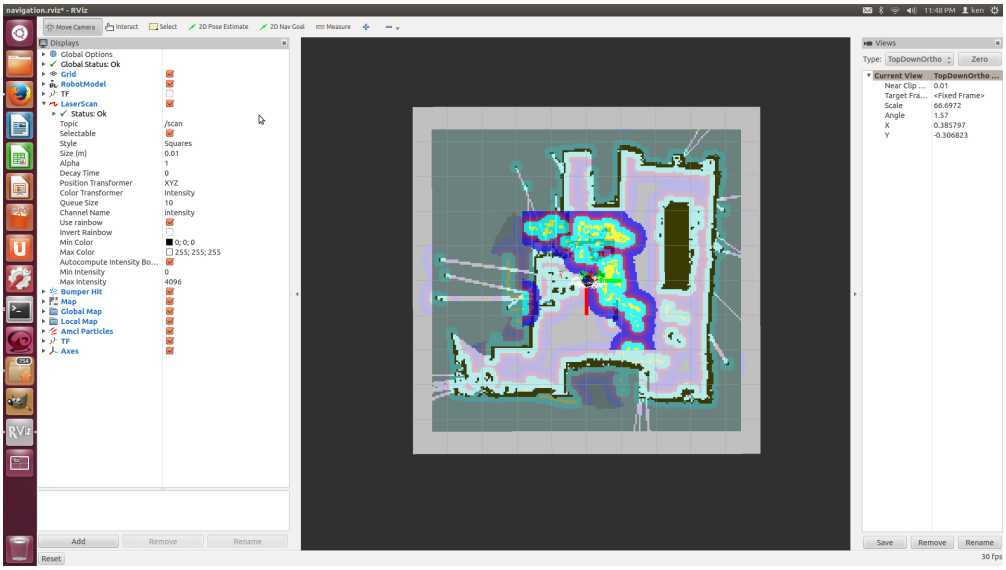

Figure 3: The window from RViz that will pop up when the navigation commands are run [1].

After running the above command, ROS visualization (RViz) will run and present a window that looks similar to Fig. 3.

Clicking on the "2D Pose Estimate" from Fig. 4 will allow the user to tell the Turtlebot3 where it actually is with respect to itself and its mapped environment. Once the Turltebot3's location has been correctly

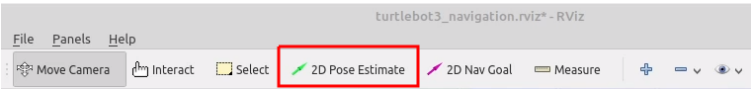

Figure 4: This is the top bar that contains the tools needed to establish where the Turtlebot3 is and instruct it where to move [2].

calibrated, use the "2D Nav Goal" button to give the Turtlebot3 a navigation goal to traverse to.

3.3. Physical Setup

No modification are required to the TurtleBot3.

4. Results and Discussion

The Turtlebot3 was able to create a path and navigate to the correct location using a map generated from gmapping of the EECs office areas.

4.1. Errors and Issues

During Lab4, multiple SLAM maps had been collected of the room, and after testing the navigation and path planning performance on each of them, the performance was highly depandant on the map quality. If the map was not detailed, the Turtlebot3 would have difficulty choosing a sound path to even get started, or the Turtlebot3 would attempt to go through walls and would eventually get nowhere.

4.2. Solutions to Errors

To fix the issue, I simply had to use the most detailed scans/maps that I had. Rushing the mapping process with SLAM is a mistake since an incomplete map will result in poor performance, and I was able observe the increased performance with better maps.

4.3. Incremental Software Building Process

This project consists of two parts:

1. The Computer.
2. The Turtlebot3.

This lab essentially only requires downloading the correct packages onto both the home PC and the Turltebot3. I attacked the installation processes individually. After finishing installing on both, I began testing using the physical robot inside of the lab.

5. Conclusion and Recommendations

In conclusion, this lab was successful. I was able to complete the required objectives for this Lab. An important take way for this lab is that navigation and path planning performance is highly dependant on the quality of the provided map of the environment. While this observation is intuitive, it is important to remember not to rush the mapping process or the navigation and path planning will not be as effective as users would hope.

6. References

[1] Obstacle doesn't show on rviz.

[2] TurtleBot3, 2022.

[3] TurtleBot3 Setup and Tutorials, 2022.

[4] Turtlebot3 Setup West Point Robotics, 2 2022.

[5] Karthik Karur, Nitin Sharma, Chinmay Dharmatti, and Joshua E. Siegel. A Survey of Path Planning Algorithms for Mobile Robots. *Vehicles*, 3(3):448–468, 8 2021.

[6] YoonSeok Pyo, HanCheol Cho, RyuWoon Jung, and TaeHoon Lim. *ROS Robot Programming Book*. ROBOTIS Co., Ltd., 1st edition, 12 2017.

[7] Kaan Ucar. Autonomous Navigation of a Mobile Robot: RRT Path Planning Algorithm, 9 2020.