WORCESTER POLYTECHNIC INSTITUTE
ROBOTICS ENGINEERING PROGRAM

Lab 3: Mobile Robot Path Planning

Submitted By:
April Bollinger
Pranav Jain
Michael Primavera
Cassie Youn

Date Completed: November 15, 2024
Course Instructor: Professor Aloi
Lab Section: RBE 3002-BX01

Introduction

In this lab, the A* path-planning algorithm was implemented using ROS packages and Python. The final result was a customized visualization in RViz that shows the C-Space, the progress of the A* algorithm, the optimized path, and the robot traveling from the start to end positions along the optimized path. These processes were then initiated with a launch file that starts the entire process with a single line of code through the terminal.

Methodology

The assignment began with learning how to work with maps in ROS. A map_server node was created to publish a map in a nav_msgs/OccupancyGrid message on the /map topic. OccupancyGrids can indicate the probability of a cell being occupied, or that the cell has not been mapped and the contents are unknown. In this lab, cells were binary: either certainly free (0) or certainly occupied (100). Some utility functions were written to translate from the one dimensional cell array, to a two dimensional cell position in a grid, and to a location in world coordinates.

Since the TurtleBot is not a point, obstacles needed to be inflated in the map to create the C-Space for path planning. A dilation algorithm was used to pad the obstacles, then the inflated map was published as a nav_msgs/GridCells message on the /path_planner/cspace topic. The “inflated” cells then had to be converted to certainly occupied (100) in a new OccupancyGrid so the robot would recognize them as obstacles.

Visualization of robot state, map/cspace data, frames and transforms using RViz were then explored. Up to this point, team members had created individual code branches to explore various approaches to implementation (see Appendix 1). The team met to discuss the solutions and decided to use execution speed to determine which code to use for the remainder of the lab.

The A* path-planning algorithm was then implemented. The algorithm results are published as nav_msgs/GridCells messages. The cells that have been explored by the A* algorithm are published on the /explored topic and the frontier is published on the /wavefront topic.

The path_planning node is changed to export a service named /plan_path. The service is called with three arguments, an initial pose for the robot (geometry_msgs/PoseStamped start), a final pose for the robot (geometry_msgs/PoseStamped goal), and a metric for the inflation of the obstacles in the C-Space (float32 tolerance). The service returns the path calculated by the A* algorithm (nav_msgs/Path plan). It publishes the planned path as a nav_msgs/GridCells message on the /path topic.

The code developed in Lab2 was used to move the robot along the path. Since the Lab2 code uses a “turn-drive-turn” system with smoothing, the waypoints on the path are found to optimize the path for the fewest drive segments. The optimized path is published as a nav_msgs/GridCells message on the /op_path topic. The /goal topic is used by the path planning node to publish the waypoints, and also by the Lab2 node to subscribe to target poses. The /point_reached topic allows the Lab2 node to indicate that the robot has reached a waypoint and that the path planning node should send the robot to the next waypoint.

Finally, a launch file was written to start all of the necessary nodes. The 2D Nav Goal tool in RViz published poses on the /move_base_simple/goal topic. The path planning node then runs the A* algorithm, publishes information about the planning as GridCells messages that are visualized by RViz, and then sends navigation commands to the Lab2 node to move the TurtleBot from waypoint to waypoint until it reaches the final goal.

Results

We were able to successfully determine paths using the A* algorithm, visualize them in RViz, and actuate the robot to follow the calculated path in simulation.

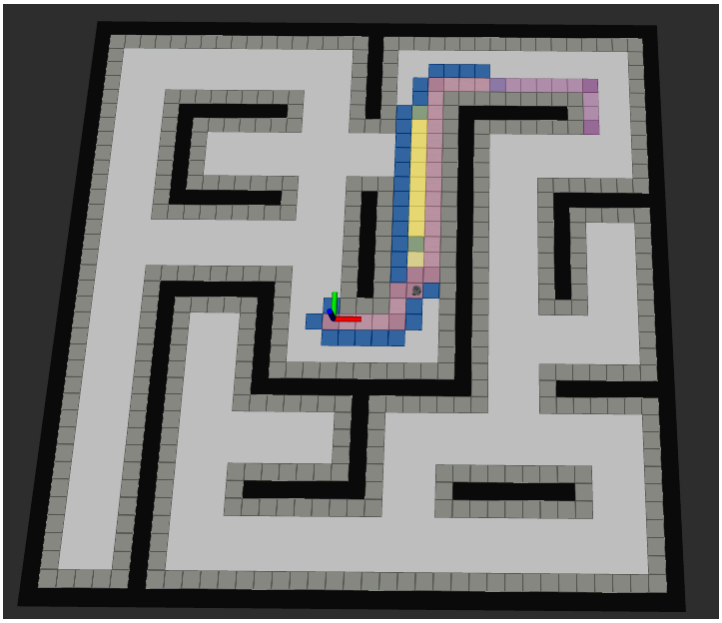

Figure 1: RViz visualization of the A* path-planning implementation. The cells explored by the A* algorithm are in yellow, the frontier cells are in blue, the goal is in dark purple, the path is in light purple and the waypoints are in dark purple.

Discussion

The goal of the lab was to perform the A* algorithm using ROS packages with the Python programming language. The A* algorithm is a modification of the Dijkstra and Best-First Search algorithm that estimates the cost of the path passing through a cell with:

$$f(n) = g(n) + h(n)$$
$$g(n) = Cost\ to\ reach\ node$$
$$h(n) = Cost\ to\ reach\ goal$$

The A* algorithm allows different weighting of g(n) and h(n), but for this lab they were given equal weight. The robot can reach the goal faster through the removal of redundant waypoints from the path. A function was written to check if the x-values or y-values are the same for two waypoints in a row. If that condition is true, the intermediate waypoint is redundant and therefore removed from the list of waypoints. The combination of these implementations resulted in a highly-efficient search and optimization process for generating the best path for the robot to follow.

Two interesting hurdles that needed to be overcome were the robot driving through padding, and also the robot not moving on the map in RViz. The first problem was resolved by creating a new OccupancyGrid from a copy of the original index with all the padding indices also set to occupied (100). The problem of the robot not moving on RViz was at first attempted to be corrected using the RViz configuration file, but after no success with that, the launch file was used to create a tf static_transform_publisher from /map to /odom 100.

The details of the launch file's function is to initiate all the nodes and communication with a single line of code in the terminal. The launch file starts roscore, describes the TURTLEBOT3_MODEL, points to the map, starts gazebo with an empty_world, spawns the robot in the gazebo and RViz, calls the saved RViz-parameters file, creates the lab2 and lab3 nodes, and the tf_publisher mentioned above.

Using RViz first depended upon getting the topics set properly. Then additional topics were chosen using the *ADD* button. The configurations on the left side of the screen had drop-down options to set. Once configured properly, the configurations were saved as a file for the launch file to use when starting the ROS environment

1. April Bollinger code release
2. Pranav Jain code release
3. Michael Primavera code release
4. Cassie Youn code release
5. Final code release

Section	**Contributors**
Individual	April, Pranav, Cassie, Michael
Group Code	April, Pranav, Michael, Cassie
Report	Cassie, Michael, April, Pranav