Lab 5 Report

Team 7:
Eva Barinelli, Richard Hosea, Nicholas Lanotte

Abstract

Robots that can explore and map unknown environments have many applications in the real world including medical surgery, swarm robotics, and household robotics.The purpose of this lab was to provide an application for frontier-based exploration. The Turtlebot3 burger was able to explore the maze and produce a map after exploration, which it used to navigate. The exploration algorithm used was found to be inefficient as it would tread over already explored areas and try to explore areas outside of the map. In future Simultaneous Localization and Mapping (SLAM) applications, alternative methods of frontier based exploration should be applied.

Introduction

The objective of this lab is to use all of the skills learned throughout the RBE 3002 course to explore a maze of unknown size using ROS packages and nodes for the Turtlebot3 burger. This is done by actively navigating and mapping the maze using frontier-based exploration. Additionally, a map was generated such that pathfinding to any point in the maze can be achieved by using the A* search algorithm, waypoint generation, and obstacle expansion.

Methodology

In order to complete the final navigation to opposite corners, the maze needed to be explored using frontier exploration. Once the map was completely created, the robot needed to navigate from the start corner to the opposite corner using A*.

Maze Exploration

To explore the maze, the navigation stack (amcl, move_base, gmapping) was used with an exploration node and a control node. All frontier uncertainty and proximity calculations are calculated by the exploration node. The exploration node subscribes to the map topic published by gmapping. By default the map published by gmapping is 384 cells in width and 384 cells in height. The total area of the map is divided into a set number of zones. When determining the zone to explore, the uncertainty of a zone was calculated along with its proximity to the Turtlebot's current position. The uncertainty value of zones is calculated by adding +1 for every cell that has an uncertainty that is not -1 and adding -1 for all other cells. Zones with uncertainty values closer to 0 are prioritized as that means half of the zone is explored while the other is not, indicating a frontier. The center of the selected frontier is then published when the control node asks for a new position.

The majority of the communication to the navigation stack is achieved using the control node. The position of the Turtlebot compared to the position of the zone is calculated and a string requesting a new position is published when the positions are within a set tolerance. That new PoseStamped topic is subscribed to by the exploration node which is responsible for publishing a new PoseStamped message. Newly received PoseStamped messages by the control node are sent to the navigation stack to actuate to the point. The time to travel to the frontier in the zone is monitored by the control node, which will terminate the path if it has exceeded the allotted time for the movement. Figure 1 shows the flow of messages between the exploration node and control node.

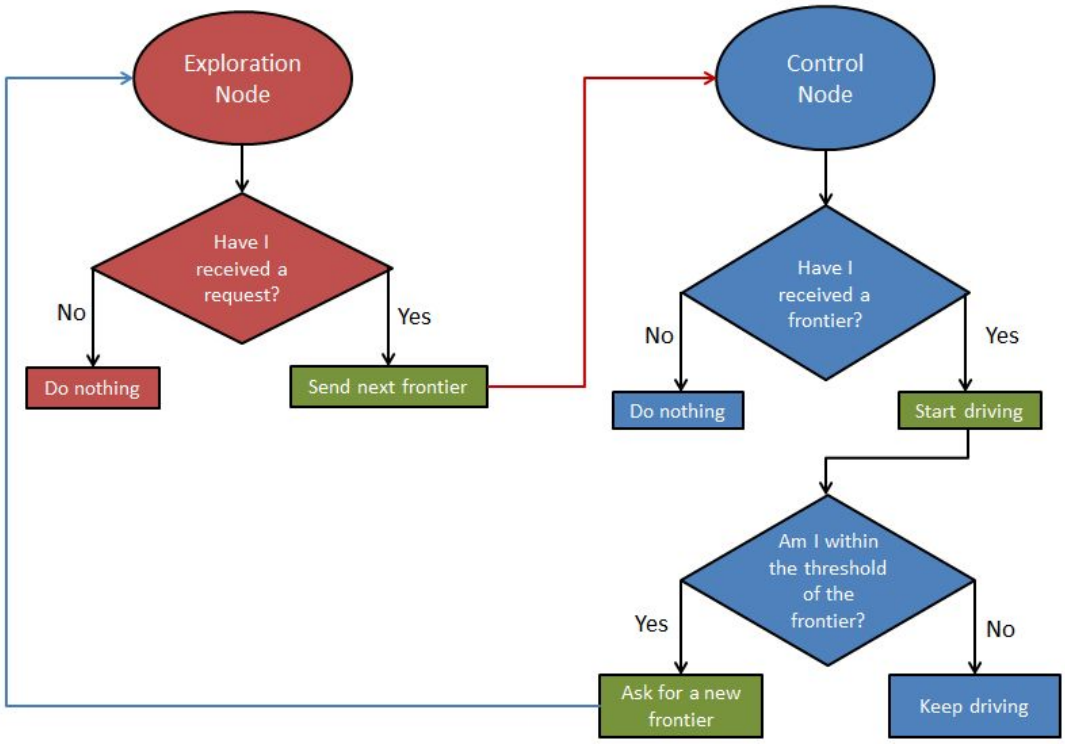

Figure 1: Exploration Node and Control Node Communication Diagram

Pathfinding

Once the map was complete, it was saved using the command, *rosrun map_server map_saver –f <map_name>*. The saved map was then used by the robot to navigate. A* was used to calculate the optimal path from the start corner to the direct opposite corner. The optimal path was considered the path in which the robot could traverse in the shortest amount of time. First, the obstacles on the map were expanded to reduce the chance the robot would hit an obstacle and the remaining cells were then used to calculate the optimal path. To calculate the lowest-cost path, the A* algorithm determined the cost to move through a specific path and added cost to turning since turning takes significant time. The algorithm calculates the cost to travel through the path up to the current node (g-score), the estimated cost to travel to the end location (h-score) and the cost to turn if there is a turn. Turns were determined if the angle made between the current node, its parent node, and its child node was calculated to be a value other than 180°. If it was not 180°, the absolute value of the angle was multiplied by a constant that set the cost to turn appropriately. The cost to turn would be the same as the cost to travel a distance in the same amount of time it takes to turn.

A waypoint generator node was created to handle the A* node and robot movement. The waypoint generator node sends the start and end positions to the A* node using an A* Service request, then receives the full path from the A* node, and then sends one point at a time to the robot on request. Figure 2 is a diagram showing the communication between the waypoint generator node, A* node, and the robot.

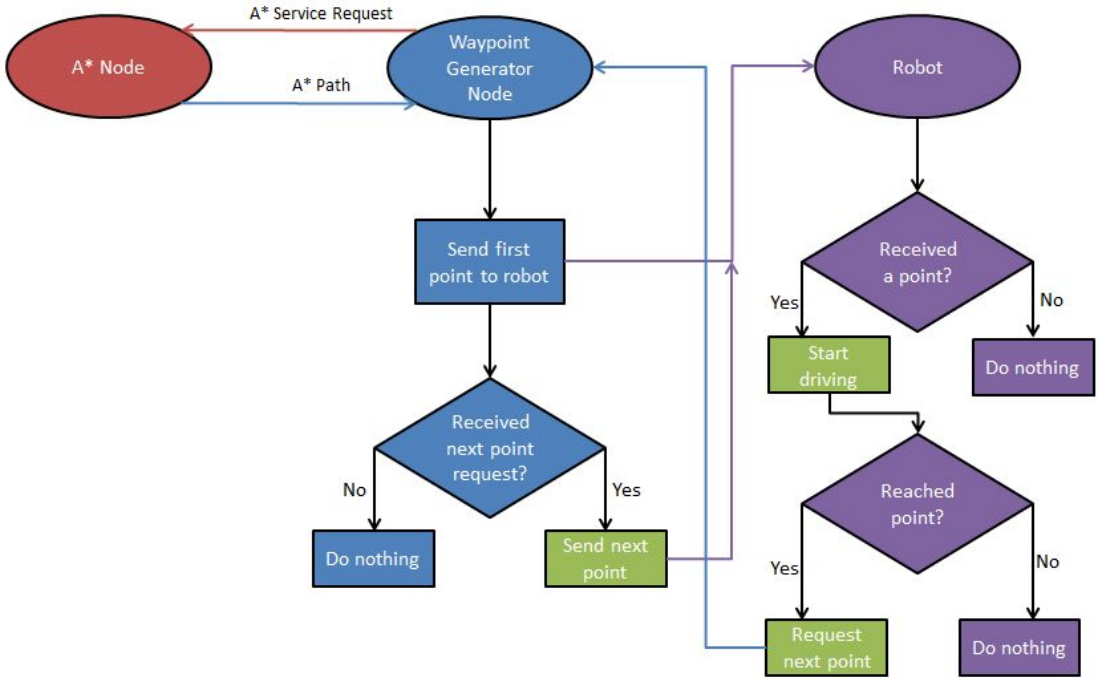

Figure 2: Communication Between Waypoint Generator, A, and Robot Diagram*

The waypoint generator node does not send every single point on the path; it sends the minimum number of points the robot can navigate to follow the path. These points are found by comparing the angles between the points on the path. Once the robot reaches the point on the path, it requests the next point until it reaches the goal point.

Results

The following section provides the results of mapping the maze during exploration and navigating the generated map.

Mapping the Maze

The robot was able to successfully map the maze for our demo, however, an error in RViz or in the GMapping Node, caused the map and robot to be rotated on the coordinate grid 180° while the odometry of the robot appeared to be in the normal position according to the exploration node. This resulted in the exploration node publishing places to explore outside of the maze. The robot successfully mapped the maze as it struggled to reach the points but this was not the intention. This was a new problem not present in previous test runs. The transforms between odom, the map, and the robot may not have been properly set.

Navigating the Generated Map

The robot was successfully able to plot the path to the end goal position but due to the odometry issues as described earlier, this caused the robot odometry to appear translated about two meters to the A* node than it actually was. This made the path very short and the robot ultimately hit a wall as it tried to get onto the path and follow it. This does confirm that the Robot movement node was functioning properly.

Discussion

While the robot was able to successfully map the maze and then nearly navigate from the start corner to the opposite corner using A* pathfinding, the exploration was not efficient. Aside from the odometry issues described in the results, the robot exploration algorithm often pointed the robot back over areas it had already mapped. This happened because of the way a “frontier” is defined; the frontiers were more like “zones.” Normally, a frontier is defined as the line of cells between the explored cells and the unexplored cells. All the possible zones were defined in the generated 384x384 map generated by gmapping regardless of if the robot could get there. The map was much smaller than the default map generated by gmapping so most of the frontier points were outside the actual map. The maze was mapped not because the robot was given a point it could navigate to inside the maze but because it was trying to find a way outside the maze to navigate to the point. Additionally, because the entire possible map generated by gmapping was broken into zones, the only way for the robot to know if the maze was completely mapped, was if all the “frontier” points were iterated through. A timer was set when each frontier point was received and if the robot did not get there with in ten seconds, a new frontier point was sent until no more frontier points were left in the list. To improve the efficiency last minute, the algorithm for selected a frontier was changed so that the closest zone was selected, ignoring the occupancy grid. This worked better at keeping the robot moving through the maze more efficiently but is not frontier exploration in the sense that the lab desired. This ultimately proved useless in the demo when the map and robot odometry appeared rotated to the exploration node resulting in the closest zones to be outside of the maze.

Testing Difficulties

The code was tested on the robot in simulation before it was used on the real robot. In simulation, the robot would drive into walls while trying to navigate to a “frontier.” A lot more time than necessary was spent trying to figure out why the robot was not working in simulation when it worked fine in the real world.

Before navigating from the start corner to the direct opposite corner, the robot sometimes had problems localizing. Depending on which corner the robot was placed in, the robot would take longer to localize. This could be seen using the particle filter. Even if the robot rotated 360°, sometimes the particle filter did not localize properly.

Improvements

Although the robot was able to complete the tasks of navigating and mapping the maze, the robot was not efficient while mapping. To improve efficiency while mapping, the frontiers need to be defined a different way, and the robot needs a better way to know when the map is complete. The frontiers should have been defined as the line of cells between the explored cells and the unexplored cells. A way to see if the map is complete is to see if there is a complete perimeter around the map and that all the cells within the perimeter have been explored.

Because of time constraints, there were parts of the lab that we were unable to implement such as having the robot return home after the map was created and turning without stopping. To have the robot return home instead of placing it back at the start position by hand, the robot could be given the completed map and use A* to navigate to the start position. Turning without stopping would have decreased the amount of time to navigate to the end point. To implement this, the robot would need to drive in arcs. Since we ran out of time, we were unable to test these two parts of the lab, so they were not implemented in the demo.

Conclusion

The objective of this lab was to use all of the skills learned throughout the RBE 3002 course to explore a maze of unknown size using ROS packages and nodes for the Turtlebot3 burger. Although exploration was done successfully, the manner of exploration was not efficient or true frontier exploration. Problems during testing and time constraints, as well as new errors with odometry, resulted in the robot demo unable to function as intended, only successfully mapping because the robot tried to exit the maze to reach the sent frontier points. If more time was available, further features like true frontier exploration could have been implemented and the issues with odometry and other transforms could have been solved, resulting in the robot behaving much more reliably.