RBE 3002 Lab 5

Gabriel Entov, Sam White, Cooper Wolanin

May 1, 2018

Introduction

The final project for this class challenges students to develop ROS packages and nodes for the turtlebot3 burger to allow it to navigate an unknown maze of size MxN (to be decided) from a specified corner to its diagonally opposite corner in the shortest time. The robot has to explore an unknown environment and then use A* to navigate to the opposite corner of a map once a the map is known

Methodology

The node class

The team created a node class which contains attributes such as an x-coordinate, a y-coordinate, and its parent node. The x and y coordinates are used to calculate the node's index in the array of OccupancyGrid data.

In the constructor of the class, the object calls the function calcFCost, which takes in the nodes x and y coordinates, and spits out the calculated f costs for the node based on the previously inputted goal node. The f cost incorporates the time it takes to turn, the manhattan and euclidean distances, and the occupancy data.

A*

First, the team made a PriorityQueue called it pQueue. The team also created a dictionary for visitedNodes and an array for visitedIndexes. A start node a value of -1 appended to the dictionary. The index of the starting node was added to the visitedIndexes array. After this, the team appended the node to the pQueue.

While the pQueue is not empty, the top of the priority queue is examined, and if the current index is the not the goal, its neighbors are looked at. For each of the neighbors, the team look to see if the neighboring node has been visited and looks to see if the cost the team has in the dictionary is larger than the one currently being examined. If it is, the team puts this new node with updated parents into the priority queue, and updates the dictionary and visitedIndexes array. If the node was never visited, the team populates the dictionaries and array regardless, but there isn't a need to check if the previous cost was larger than the current cost.

The Robot Class:

The robot class handles all of the robot's movements. The robot's core functions including driveStraight() and rotate() are used in higher level functions such as followPath, calculateDistanceTraveled, and checkPath.

Twist messages are published in order to move the wheels of the robot. The x and y positions are extracted from the pose of the robot, and are used to figure out how far the robot needs to travel. The same twist message is used for rotating. The robot rotates from 0 to pi, and then from -pi to 0. This makes it so that rotating past pi is not intuitive, as the sign flips. To account for this, if the initial yaw, which is the current yaw plus the angle, is less than negative pi, the team adds 2pi to it. Likewise but inverted for when initial yaw is greater than pi. The robot should only need to make 90 degree turns when running A*, so this conversion worked fine.

Finding Frontiers:

In order to find the unknown frontier areas, each of the grid cells were checked. If the cell was unoccupied, its neighbors were checked. If the neighboring cell was unknown, then the current cell is a frontier cell. This was changed from examining the unknowns then their bordering unoccupied cells to save time and memory.

Navigation to Frontiers and Exploring the Map:

To navigate to the frontier, the indexes of the array holding the frontier values are checked, and turned from indexes into points on the map, using our indexToPoint function. The closest point to the robot is passed to the Navigation Stack, and the robot moves to that point. Once it has reached that point, it finds new frontiers based on the updated map data. It continues to explore until the map no longer has unknown frontiers.

Navigating back Home:

Once the entire map had been explored, the robot once again uses Navigation Stack to navigate back to its home position. The origin position is stored in a variable that is used as the “home position”.

Results

Exploration:

The first two times that the robot ran during the demo, Navigation Stack had failed. As previously aforementioned, when Navigation Stack failed, a quick fix was to change map padding, and the run was successful on the third attempt. Instead of padding the unoccupied cells of the map once, the function was ran twice. This prevented the robot from getting too close to an obstacle and being unable to continue moving. The figures below demonstrate what the simulation shows as the robot is exploring.

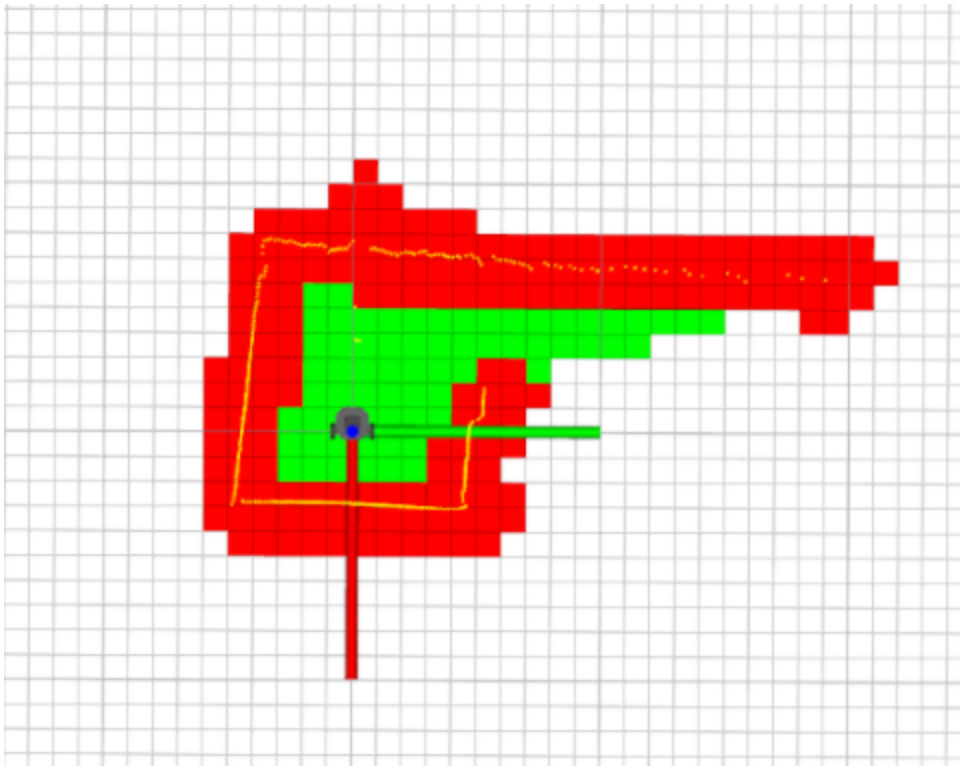

Figure 1: Robot Makes Grid Data

Figure 1 shows the robots grid data. The white cells are unknown, the red cells are occupied based on LiDAR data, and the green cells are unoccupied.

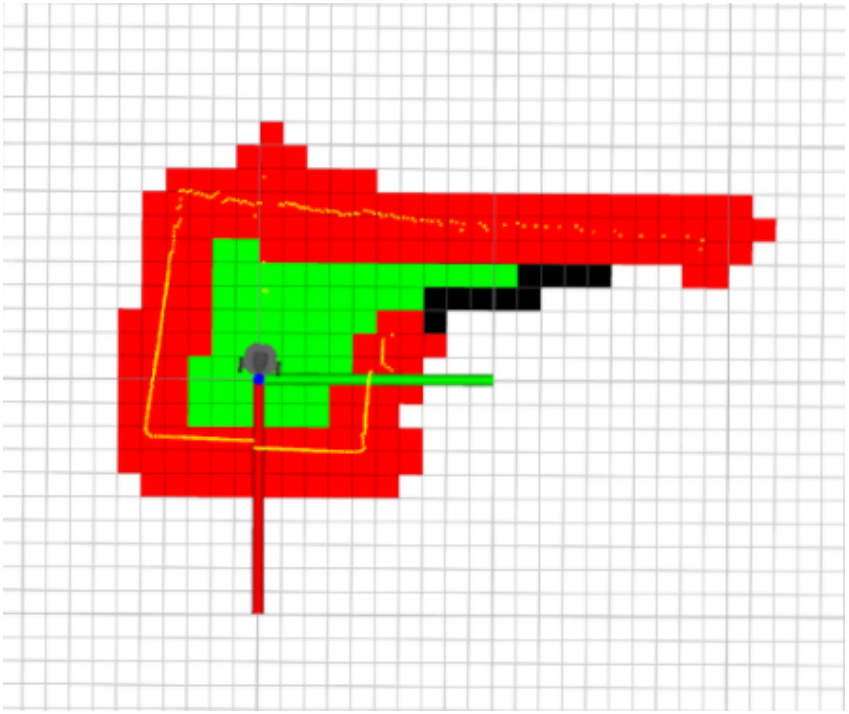

Figure 2: Robot Finds Frontier and Moves Towards It

Figure 2 shows how the robot finds the frontiers, which are labeled in black. Here the robot moves towards the closest Grid Cell in the frontier array, before trying to find more open Grid Cells, shown in Figure 3.

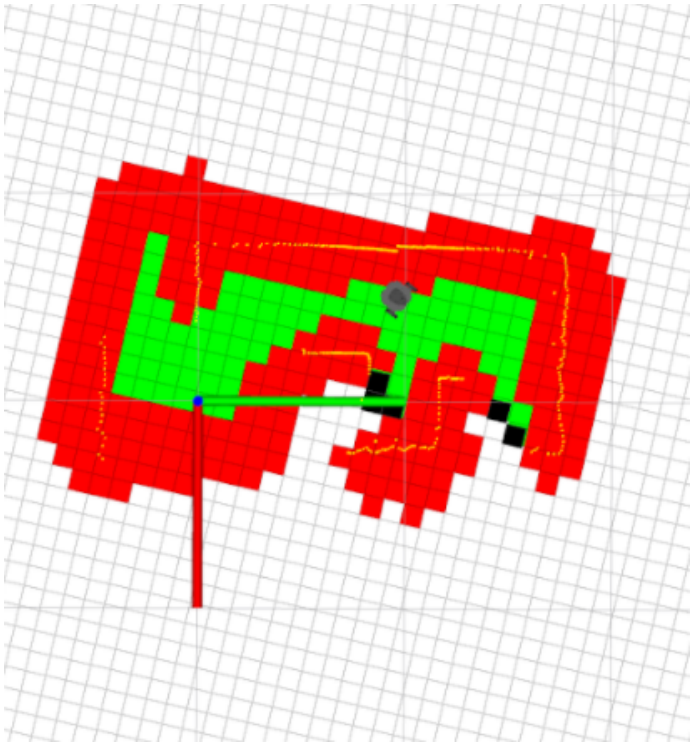

Figure 3: Robot Finds New Frontiers and New Open Cells

Figure 3 shows that once the robot has navigated to the frontier it previously discovered, it finds new open cells based on LiDAR data, as well as new frontiers. Once it figures that there are no more frontiers to explore, it has finished exploring the map. This is seen in Figure 4 .

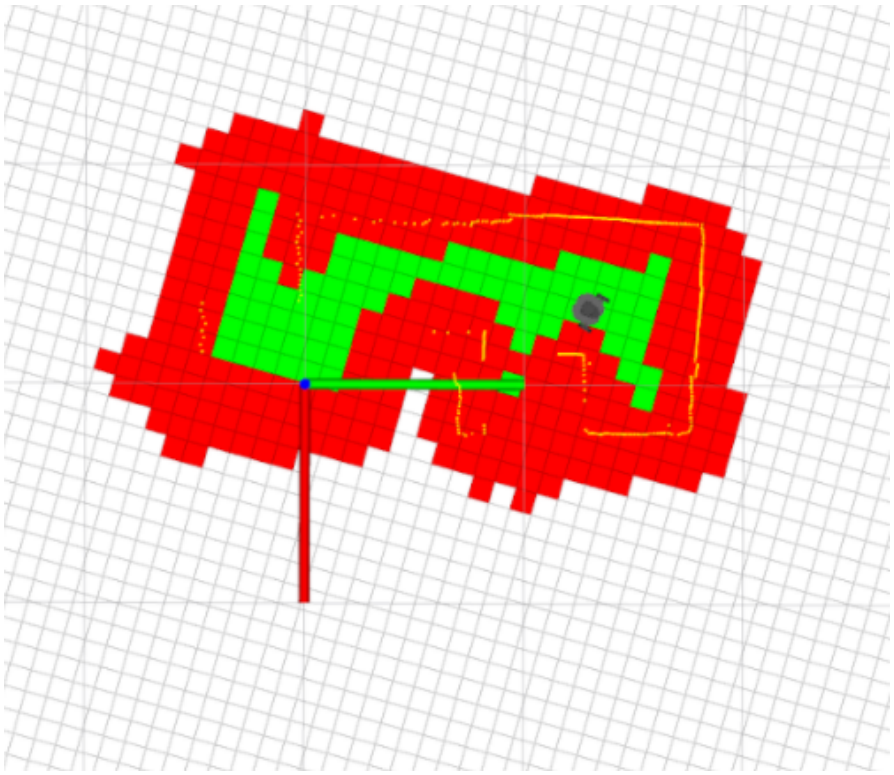

Figure 4: The Robot Has Finished Exploring, and Returns to Home

Figure 4 shows the explored map. Now that there are no more frontiers to explore, the robot begins to navigate back home. Once it is home, it is given a point to navigate to, and does so using the A* algorithm. This behavior is shown in Figure 5.

A*:

The team was struggling to get rotation working, as the robot would attempt to drill a hole in the field. To compensate, the team substituted the rotate code of one students with another (in the same group), and things the went smoothly from there. Figure 5 shows the path generated by the robot. The robot was successful in navigation from one corner to the next during the demonstration.

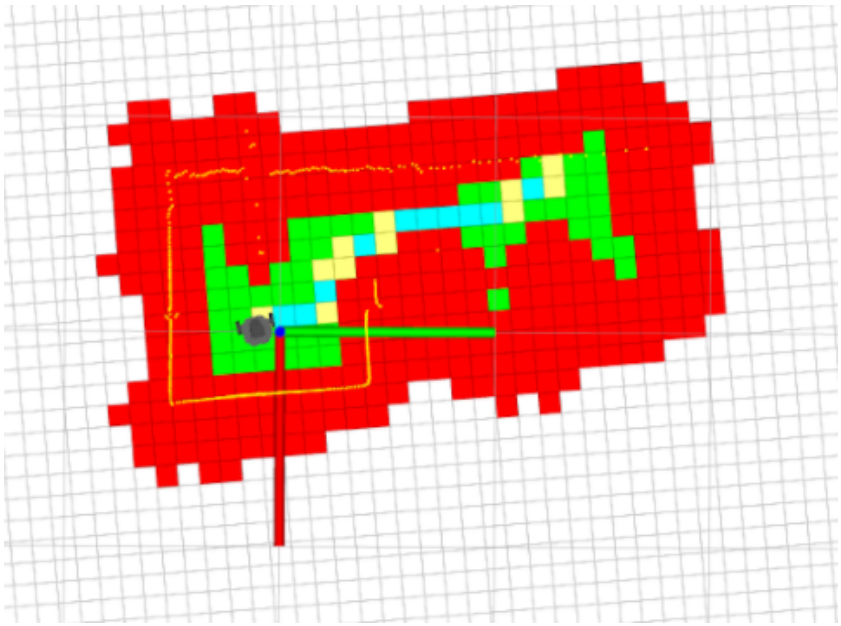

Figure 5: Robot Plans Path to Destination

Figure 5 shows the path generated to the destination using the A* algorithm. The blue cells are the path, and the yellow cells are the waypoints.

Discussion

The goal of this lab was to demonstrate the ability to explore the maze and find frontiers, return to home, and use A* to navigate a mapped environment. The team was able to accomplish the task on the third attempt, without touching any of the walls or other obstacles. Figures 1 through 4 show how the robot navigates the map using the unknown frontiers, and Figure 5 shows the path generated by A* to navigate between points on the map. These techniques are used on search and rescue missions, and other tasks where mapping is necessary. This project was especially helpful as the precursor to the KINISI MQP, on which all three members of the team belong.

Conclusion

This lab put all the pieces of the other labs together: A*, Occupancy Grids, ROS, and robot commands. It also helped strengthen the almost non-existent understanding of the connection between ROS, rViz, and python. The team also learned how to make a launch file, something that the team was unsure of before this lab. Having a team of three made it much easier for members of the team to bounce ideas off of eachother, and correct any mistakes that the other member couldn't see. This lab was very difficult, as sometimes the robot behaved differently than expected. Some things that the teamre out of the team's controls, such as Navigation stack, sometimes crashed as it suspected it was near an obstacle. This was fixed by padding the map with a different resolution. Before trying to complete the lab on the robot, the team made sure that everything worked in simulation, under any edge cases. Our simulation was very robust, and this proved to be a wise decision, as the transition to the real robot was almost flawless. Debugging some problems was extremely difficult, but these challenges were overcome by talking the problems out with the other members of the team and the lab staff, solutions came to the surface (albeit sometimes slowly).