RBE 3002 Lab 4 Report

Team 12: Yiyi Wu, Puen Xu

I. Introduction

For the final project of RBE3002, we programmed a Turtlebot 3 robot to explore an unknown maze and generate a map for the maze using laser-based SLAM. In addition, we utilized the AMCL algorithm to enable the robot to localize itself in the maze using the generated map and navigate to a desired destination using the A* search algorithm. At the end of the lab, we deepened our understanding in mapping, particle filter, and search algorithms.

II. Approach

Part 1. Logic Flow For Three phases

a). Phase 1

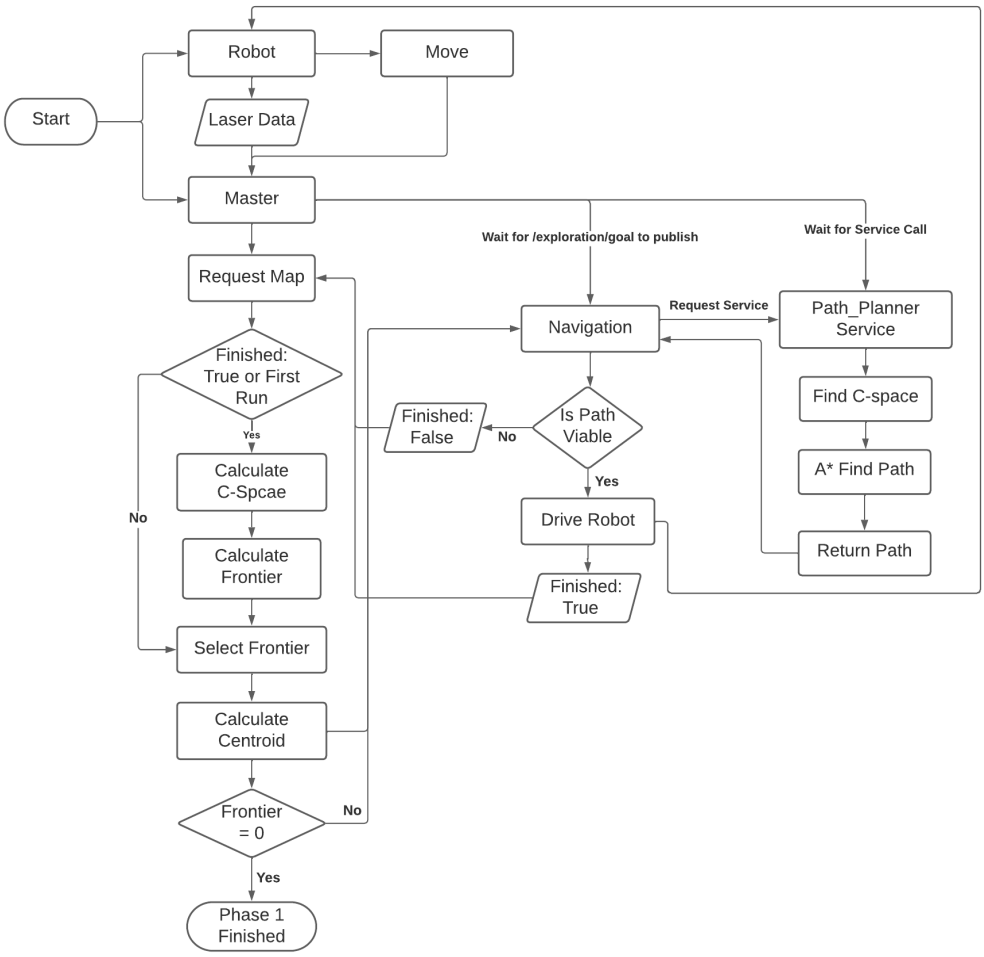

Figure 1. Flow chart of phase 1.

In phase 1, the main driving node frontier_exploration subscribed to the /map topic to retrieve the map generated by Gmapping. Then it calculated c-space and selected the local frontier after cleaning and binning. Next it calculated the centroid of the local frontier and sent it as a PoseStamped message on /exploration/goal. The navigation node subscribed to the /exploration/goal topic to get the destination and called a GetPlan service to request a path from the path_planner node. After the service proxy was made, the path_planner node used A* search algorithm to find an optimal path and return it to the navigation node upon service call. Next, the navigation node commanded the robot to follow the path and navigate to the desired point, which was the centroid of the local frontier. After navigation was completed, it sent a boolean message to /navigation/done topic which the frontier_exploration node subscribed to. After the boolean message, the frontier_exploration node got the updated map, and reiterated the process (calculating c-space, selecting frontier, computing centroid, and sending it as a goal message) until there was no frontier cell left.

b). Phase 2

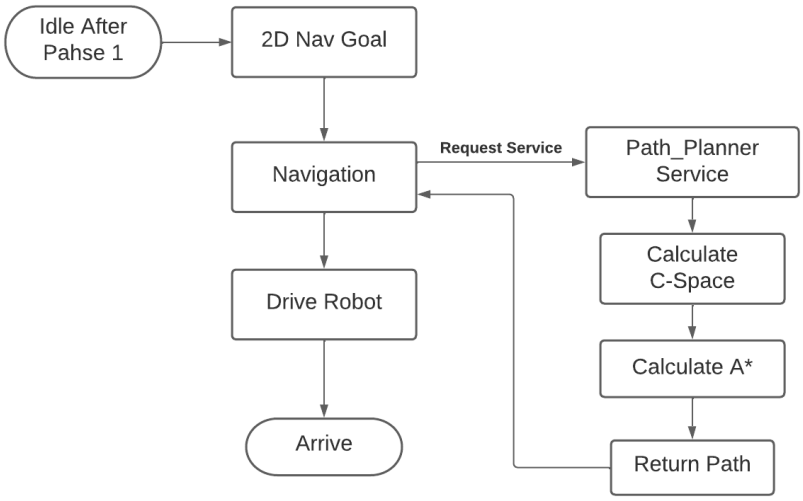

Figure 2. Flow chart of phase 2.

After the full map was explored, in phase 2, we sent a /move_base_simple/goal message manually by using 2D Nav Goal in Rviz to send the robot back to the starting point. Upon receiving the pose message, the navigation node requested path planning service. The path planner service then calculated the c-space, calculated A* prioritizing the cells with more free neighbors, then optimized the path, and returned the path back to navigation. Then the navigation

sent the points to the robot using go_to_position, which took in a point and drove the robot. After the robot arrived at the goal pose, it stopped and waited for the next command.

The map that was explored and created by the robot was saved manually by us using command line ROS map service.

c). Phase 3

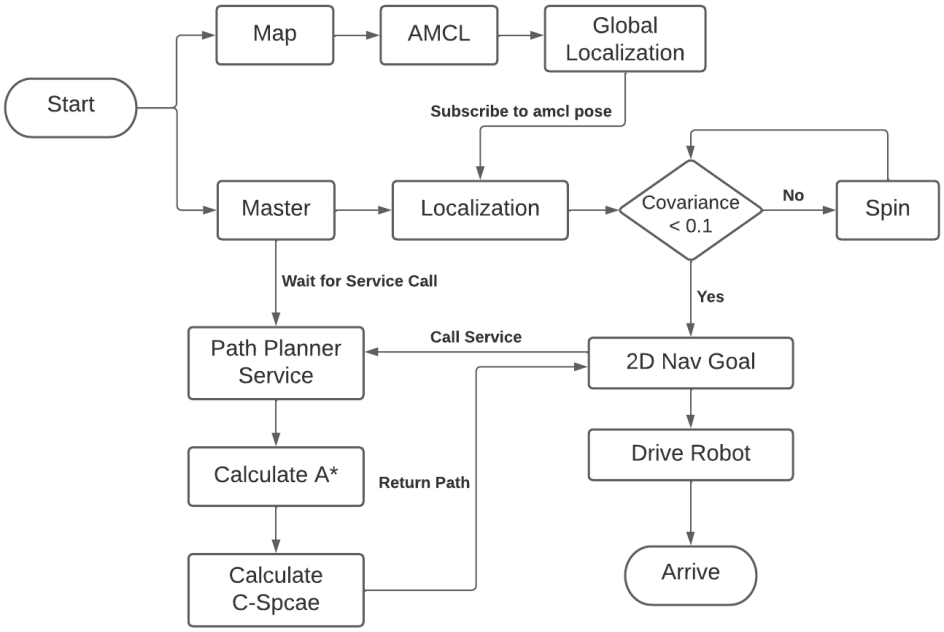

Figure 3. Flow chart of phase 3.

For phase 3, we restarted everything. The robot was first "kidnapped" and did not know its current location. We utilized the ROS service AMCL with the previously saved map to localize the robot. To improve the accuracy of localization, the robot began spinning immediately after the program was launched. The localization node was subscribed to the AMCL pose topic, where PoseWithCovarianceStamped messages were posted by AMCL. The covariance in this type of message is a list of length 36, where the final entry represents the accuracy of orientation (w). We set the threshold to 0.1 to ensure accurate localization.

Once the robot located itself on the map, a random goal point was selected and 2D Nav Goal was used to send the robot to that goal. The process was then the same as in lab3 and phase 2, where the robot navigated to the goal position.

Part 2. C-Space Calculation

For C-Space calculation, we kept the code from lab2, and used it as dilation for frontier exploration. The function took in a map and the number of padding, and returned a GridCells message that contained the map after adding the c-space. To be specific, the function would turn all walkable neighbors of an occupied cell into being occupied.

Part 3. Frontier Identification

a). Frontier calculation

The map after dilated with c-space was passed into the frontier calculation function. The frontier was defined as the boundary between known and unknown, which consists of every walkable cell that is connected with an unknown cell. Therefore, the function would loop through all the walkable cells and mark them as frontier cells if they had at least one unknown neighboring cell. All frontier cells were then returned as a list of GridCells.

b). Frontier cleaning

After the frontier cells were calculated, we employed a breadth-first search (bfs) to determine all the reachable cells. We first retrieved the position of the robot with regard to the odom frame by subscribing to the /odom topic. Then we started a bfs from the current position by using a FIFO queue and a list to store all the visited cells. By using a bfs, we could determine all the reachables cells, which could help us exclude cells out of the maze (not reachable). This method was crucial to solve the issue when the laser scanned cells out of the map and the robot kept visiting these unreachable destinations.

c). Frontier selection

After unreachable frontier cells were eliminated, we binned the frontier cells and selected the longest local frontier. To be specific, we first added the very first frontier cell to an empty list, and added the appended list to a list of lists. Then we used a nested for loop to go through the cells and the list of lists. If a cell was within a certain range (0.5 after some experiments) to the first element in a list, then we would add it to the list. Otherwise, we created a new list to the list of lists and added the cell to the new list. Now that we binned the frontiers, having a list of list of frontier cells, we found the list that had the longest length and selected that as the local frontier to visit.

Part 4. AMCL Localization

The AMCL service was embedded in the phase 3 launch file which contained the new map saved in phase 2, and stopped using Gmapping service. The localization node was

subscribed to /amcl_pose, which received PoseWithCovarianceStamped. There were a few variables that we adjusted to make the AMCL service more suitable. We doubled the default value of both of the max and min particles; set the update_min_d (minimum updating distance) and update_min_a (minimum updating angle) to 0.001, where the default values were 0.2m and pi/6 rad, which left a big gap between the previous reading and the current. After the map was loaded in RViz and AMCL running, we called the /global_localization service that was built-in with AMCL, then ran the code that spun the robot until the covariance of orientation w posted by AMCL is lower than 0.1.

III. Experiment

Part 1. First Demo

In the first demo, phase 1 and phase 2 were run with acceptable results. However, the AMCL was not updating the robot's current x and y pose. We integrated the update_odometry function from the TAs which implemented the transformation matrix, but it was set to be the callback function for /amcl_pose topic where it should have been just the /cmd_vel topic.

Part 2. Second Demo

For the second demo, the robot couldn't finish phase 1 because of how we picked the frontiers and the end exploration condition. We used global centroid, and set the end condition to if the robot explored 3 times and the frontiers left was less than 5. The laser was detecting cells outside of the map as obstacles, as a result, the line between that cell and the actual wall was categorized as frontier, and the centroid was calculated based on the location. Therefore, the map was never completed.

Part 3. Final Demo

Prior to the final demo, we corrected the odometry updating function and cleaned the frontier to avoid detecting cells outside of the maze. As a result, we had a perfect run in the final demo, having the robot explore the full maze, navigate to the starting point, and localize itself after being relocated without collisions in around 7 minutes.

IV. Discussion

Part 1. Things we improved after PDR

a). A* search

One thing we realized was that when we had a map of 0.05 resolution, if the padding for c-space was 1, the robot would likely hit the wall; if the padding was 2, however, the c-space

would become too large that sometimes blocked paths that ought to be legit. The SAs suggested that we could increase map resolution and made c-space padding 2. Through experiments, we found out that it would slow down the computation process. Instead, we decreased the map resolution (making each cell bigger) and used 1 as the padding. In addition, we added a heuristic which is the number of open 8 neighbors of the cell to the A* algorithm so that the robot would avoid picking a path next to the walls.

b). Frontier selection

Another major thing that we changed after our presentation in the beginning of the second week was that we commanded the robot to visit the local frontier insead of the global frontier. Visiting the centroid of the global frontier, the robot would probably get stuck in the middle if there were frontier cells on both sides and thus could not explore the full map. As a solution, we binned the frontier cells, as discussed above, to select the optimal local frontier to visit.

Part 2. Potential further improvement

To further improve the performance of the maze exploration robot, we could use OpenCV libraries for edge detection to increase computation efficiency instead of using customized functions that rely on for loops. In addition, we could use existing external libraries for erosion and dilation to create a smooth map of the maze which could benefit the localization process.

V. Conclusion

In this lab, we developed a robot that was able to autonomously explore an unknown maze by Gmapping, quickly localize itself by AMCL, and safely navigate to a certain point by c-space and A* search algorithm. Throughout the process, we have gained high-level programming skills and a deeper understanding of key concepts, such as sensor probability, particle filter, and SLAM.

VI. Reference

Code Gist: https://gist.github.com/PuenXu/a10fd8b1f9b2e1159614bcc8c135cfb8

Code Release: https://github.com/RBE300X-Lab/RBE3002_D23_Team12/releases/tag/v4.0