Worcester Polytechnic Institute

Robotics Engineering Program

LAB 3&4: Robot Mapping and Navigation

Submitted By : Yichen Guo, Yuhan Wu, Haojun Feng

Date Submitted : 05 / 02 / 2022
Date Completed: 05 / 02 / 2022
Course Instructor: Prof. Michalson
Lab Section: RBE 3002 D01 Group 5

Introduction

In the final lab, we used A* path planning algorithm and showed it in Rviz. We managed to use ROS service, which is a useful tool to communicate between nodes in ROS, to run algorithms between each node. Then we utilized functions we achieved to navigate through an unknown maze with the help of Gmapping during phase 1 and 2, and recorded the map robot explored. Then for phase 3, we used AMCL to do the localization for our robot and navigate in the map recorded in prior phases. In phase 3 we select arbitrary points and navigate the turtle bot to the point selected. The robot does not necessarily need to follow an arc, but it should move as fast as possible to win a potential award.

Methodology

We wrote mainly two nodes to control the motion of our robot. The path planner service handled finding paths from one point to another such that the robot could be efficient and not run into objects. The path planner service also handled finding frontiers in a map and finding the representative points of those frontiers.The main node handled calling services and moving the robot which is the/lab2 node (Since it mainly uses the functionalities we implemented in lab2).

For each of these two nodes we have a launch file to bring up the turtlebot and set up the environment parameters and SLAM (Simultaneous Localization and Mapping) as well as calling the AMCL and Gmapping. In simulation, we have a launch file set up in a Gazebo environment for simulating our robot model navigating through a simple map for testing some of our functions.

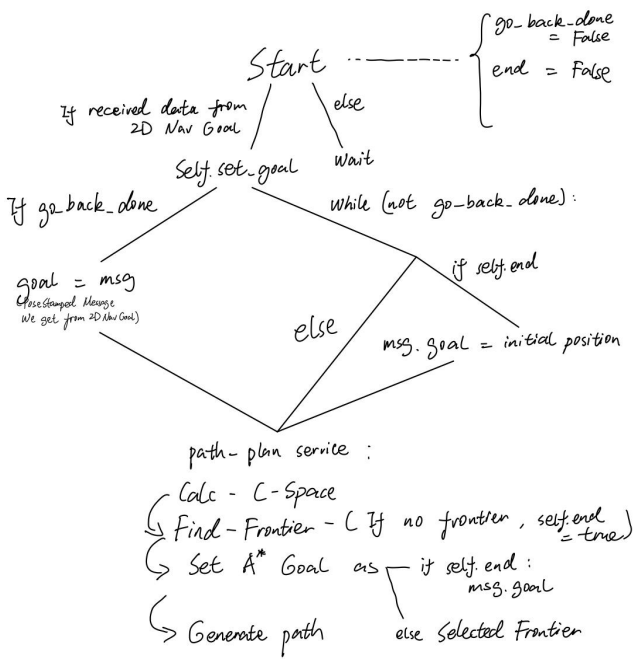

Figure 1. Program Flow Chart

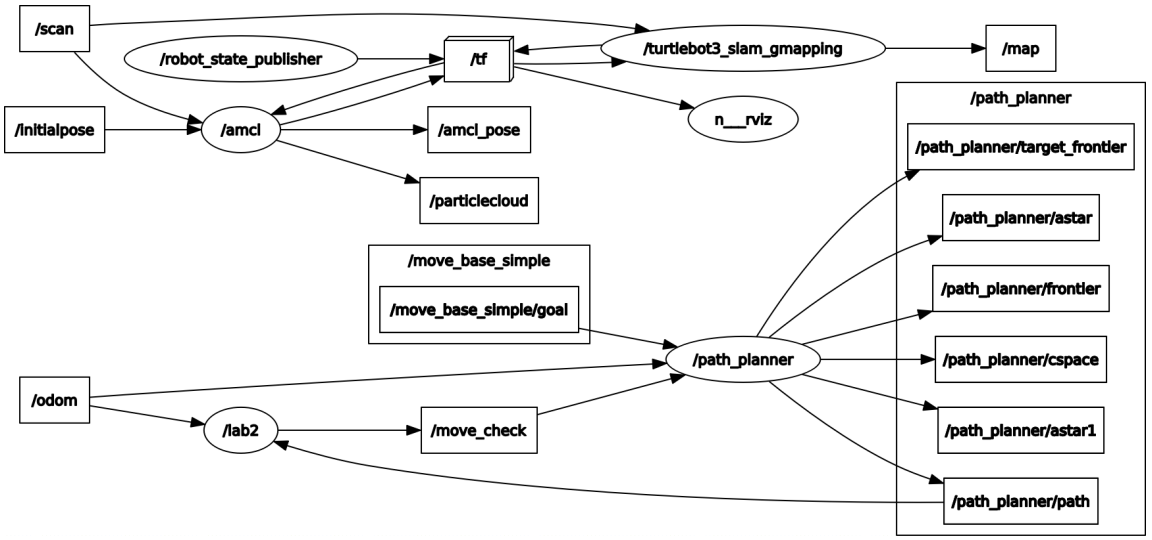

Figure 2. Planned node for phase 1 and phase 2

In phase three, the /lab2 node is slightly changed to use the coordinates in the AMCL frame instead of the Odometry frame to be able to localize itself in a known map.

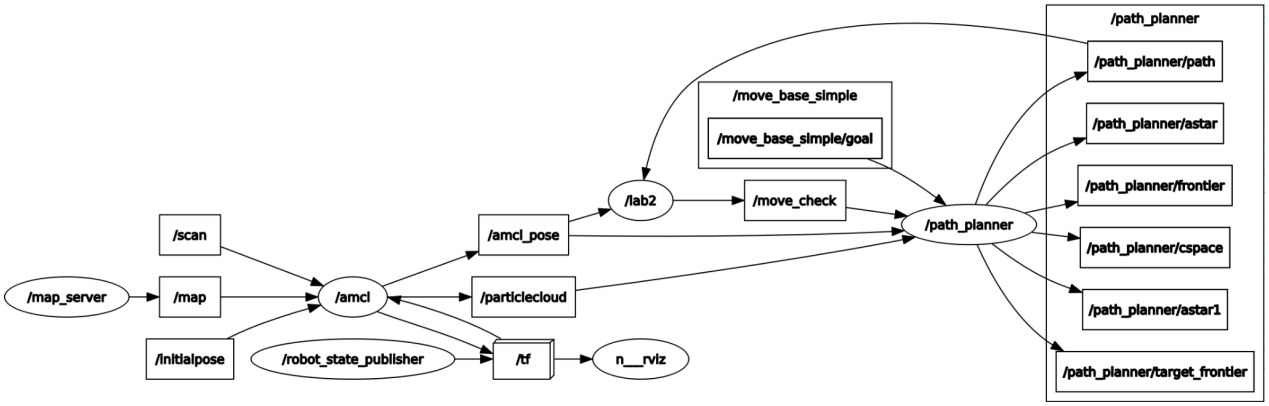

Figure 3. Planned node for phase 3

Our Rviz interface during planning is set up as below to show all the messages of all the topics we used in our programs:

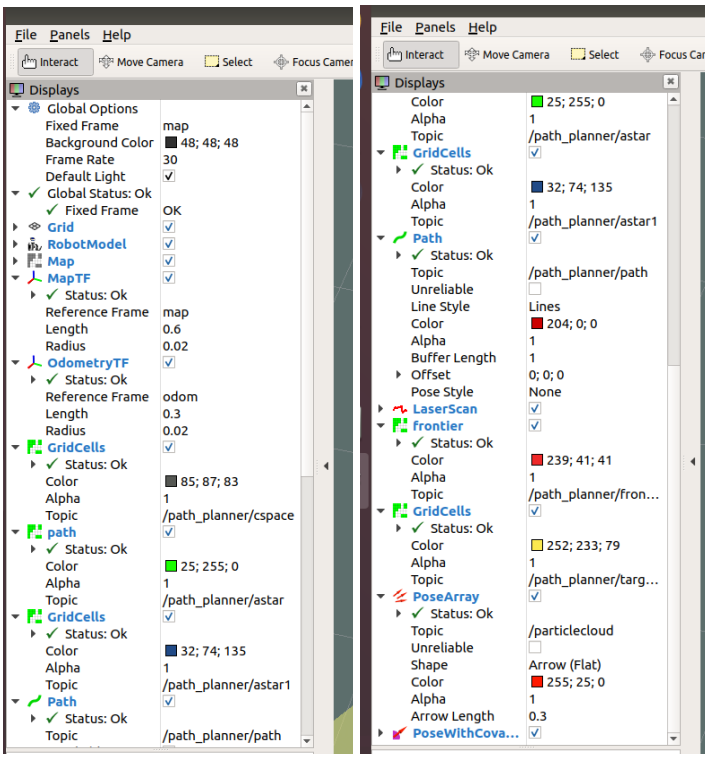

Figure 4. Rviz Interface

1. C-Space Calculation

   We use a C-space calculation function to set a frontier as the goal for A* algorithm, and the purpose of this calculation is to expand the unwalkable cell (obstacle) to decrease the chance of hitting the wall when our robot is navigating using A*. The Li-dar sensor on turtlebot will detect the obstacles as well as exploring the unknown areas into known areas. In this function, we define all the cells in the map, if the cell in known areas is unwalkable (mapdata value > 19), append the cell with x,y coordinate within the range of the padding value we assigned to the unwalkable cell list and update the map data with the list. After tuning the padding value in the real map, we would be able to have a reasonable c-space that can prevent the robot from hitting the wall and have enough space to navigate the whole map. Every cell that is neither walkable nor unwalkable will be considered as unknown cells.

   Moreover, we applied a filter to filter out the cell outside the wall that we can't navigate to. Sometimes when the Li-dar sees through the gap between two boundary boards, providing a stripe of walkable cells outside our map; we filter out all walkable cells with one or more adjacent unwalkable cells in the map. This will sacrifice some walkable cells within the map, but since the Li-dar can detect a large area, it won't affect the overall outcome for frontier selection, and it could significantly decrease the possibility for our code to locate a frontier outside the map range.

2. A* Algorithm

   In A* function, we utilized a helper function called neighbor_of_8 to get all the walkable cells around a given cell. Our A* algorithm is based on the principle of priority queue. We set a frontier that represents the cell that will be searched by our robot, and this is given by the neighbor_of_8 function of all frontiers. We list our frontier in an order of lowest cost to highest cost. The greedy cost is calculated by the number of cells visited to reach the current cell plus the euclidean distance which is the heuristic cost between the current cell to the goal. For the next step, the A*

algorithm will move to the frontier that has the lowest cost in the priority queue and apply the same step as before. Throughout this process, the algorithm keeps track of the list of the cell to reach the goal position; thus, by reversing the list, we can get a path from our start position to our goal position. One interesting hurdle we encountered was the heuristic of the A*, we tried calculating the euclidean distance for both past costs and cost to the goal position, but it didn't turn out well since it didn't take into account the obstacles that we passed. Our current heuristic method utilized the cell traveled in the past so that we could generate a more efficient path.
Then the planned path will be published to our main node, where the navigation functions will extract the waypoints from the published planned path and let the robot move to each of them from its current coordinate.

3. Frontier Selection
   We navigate through every cell in the map and find the frontier. Our frontier is defined by the walkable cell that has one or more unknown cells next to it (within the range of neighbor_of_ 4). The target position will be set to the nearest frontier in euclidean distance. In this way, we can move more efficiently since we can rely on the Li-dar range to detect the unknown space of further frontiers. And if there's no frontier detected, our target position will be set to the initial position, so that our robot will navigate to the initial position after it's done exploring the map.
4. Map saving
   We utilize the map_server ROS node, more specifically, the map_saver that retrieves map data and writes it out to pgm and yaml format. We save our explored map data after we return to our initial position when our mapping is done.
5. Amcl localization
   We import the map that we saved in step 4 as the map for our phase 3. For localization, we utilize the global_localization service provided by amcl and create a separate launch file so that it will be easier for us to evoke the service at any time. The global localization will spread the particle cloud across the whole map. Then, when we send the robot to move, the service will calculate the place where our robot is most possible to be at based on the comparison between the Li-dar sensor and the static map data.
6. Navigation
   We subscribe to the /amcl_pose as our current position after we successfully locate our robot to ensure that all our position is in map frame. We subscribe to the 2D nav goal that gives the coordinate of our goal. We set our goal and utilize the plan_path service to execute the A* algorithm to plan a path from the starting point we get in /amcl_pose to the goal position assigned by 2D nav goal. Before executing A*, we calculate the C-space to prevent our robot from hitting the wall.

Results

• Robot behavior in simulation

The C-space calculation and the A* algorithm works well in simulation. The robot calculated a proper C-space, choose the right frontier, then follow the most optimized path calculated from the A* algorithm accurately. The C-space, frontiers, and A* algorithm planned paths are all visualized in Rviz, and a simulated robot is moving accordingly in Gazebo.

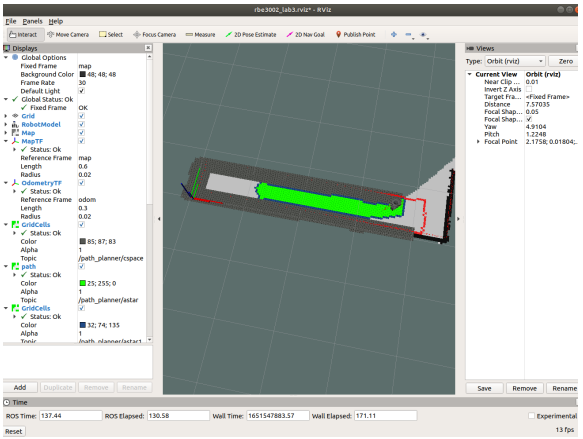

Figure 5. Turtlebot runs in simulation (view in Rviz)

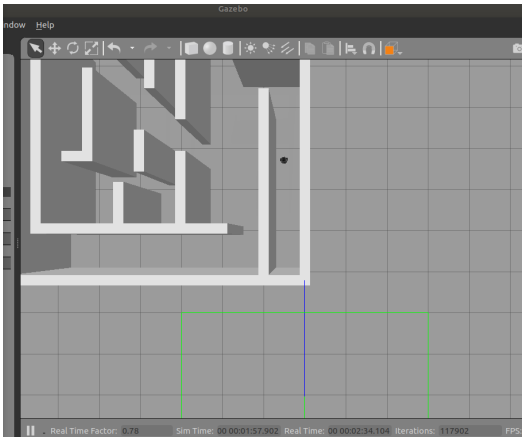

Figure 6. Turtlebot runs in simulation (view in Gazebo)

• Robot behavior in Phase 1 and Phase 2

Everytime, the robot calculates the C-space and plans a path to the frontier between known and unknown areas. Tuse, the unknown map was updated by radar on turtlebot. The turtlebot followed the path accurately, except for some drift of odometry coordinates caused by the environmental noise factors.

In phase 2, the turtlebot navigated back to the origin accurately after it explored the whole map (after completing phase 1), but the same problem in phase 1 appeared. This will be discussed in the Discussion section.

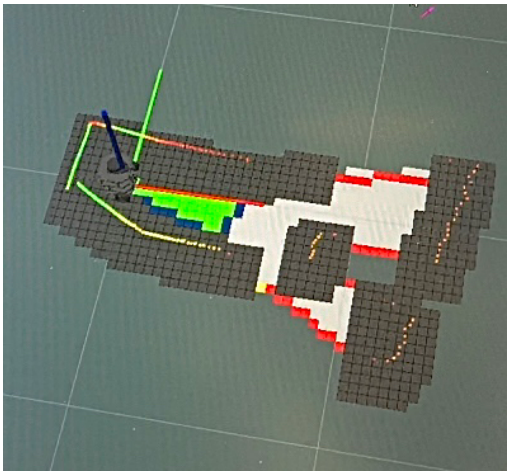

Figure 7. Turtlebot runs in real map (phase 1 and 2 view in Rviz)

• Robot behavior in Phase 3

The global localization successfully locates the new position of the robot in the maze under the ACML frame which is aligned with the map frame. It also successfully generated the path going from where it was to the point given using A*. However, the robot followed the path not very accurately.

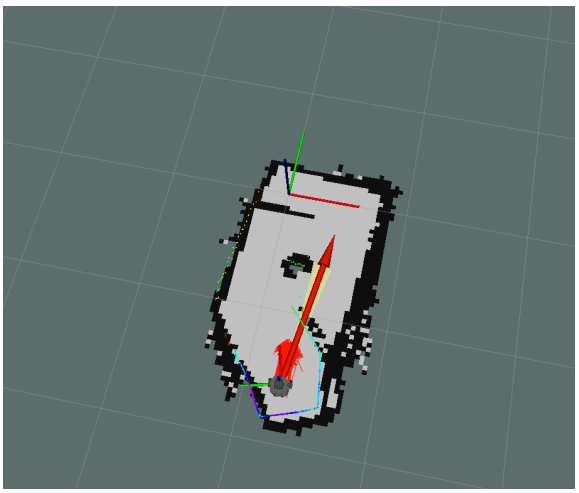

Figure 8. Turtlebot runs in real map (phase 3 view in Rviz)

Discussion

In reality, the sensors on turtlebot can be disturbed by the environment. For example, the wheels can experience different friction in reality. As a result the odom frame we used in phase 1 and 2 will start to drift from the map frame, which leads to the problem we encountered in our result of phase 1 and 2: the robot would not be able to localize the current position accurately anymore. To solve this problem we can apply a transformation function between odom frame and map frame to avoid problems caused by the environmental noise factors.

The range sensor can also find the spaces outside the walls, which is different from simulation, but we have written a filter to filter spaces outside the walls. The rigid objects in reality are also harder for range sensors to detect its shape. Implementing a motion of arc in the future would help to move around a rigid object better. It is also beneficial to implement PID control in our main node.

The result of our phase 3 is not achieving the goal point due to that it was following the planned path inaccurately. This problem is caused by the AMCL localization not being accurate enough to give the exact current pose of the robot in the map, there are always some offsets when the robot tries to move point to point along the path. The solution is obvious: tuning the AMCL parameters in that map. However, since this lab is squeezed to a three weeks lab from four weeks lab, we do not have enough time to tune those parameters.

Conclusion

In this lab, our group has successfully learned to develop the methods to explore an unknown maze and navigate through the maze we explored, using two main nodes which mainly includes C-space calculation functions, A* algorithm path planning functions, and navigating functions following a planned path. Although we do not tune the AMCL well enough, we still manage to complete the whole lab with some decent results.

Code link

https://gist.github.com/Abysssigher/bd4d9401ebf36a358e123c5750ea3af5