Efficacy of the MNL Controller in Dynamic Indoor Environments

Evan Arenburg, Spencer Belleville, Ethan Chandler

***Abstract*— By using a middleware such as robotic operating system (ROS), the implemented basic forward and inverse kinematics in lab two, the implemented C-Space and A* algorithm in lab three, and the frontier-search algorithm implemented in lab four, an autonomous path planning and mapping routine was implemented via a mounted Lidar sensor on a TurtleBot3 Burger.**

I. INTRODUCTION

The TurtleBot3 Burger is a two wheel drive robot equipped with a Lidar sensor. The two wheels have a maximum velocity of 0.22 m/s with a Raspberry Pi to control the robot.

This project aims to use robotic algorithms and control methods to control the TurtleBot3 and have it autonomously map out a workspace without running into obstacles and traverse the mapped out workspace in the most efficient way possible. In order to do so, the Lidar sensor must be able to detect and report the surroundings of the TurtleBot3.

The robot's Forward Kinematics, Inverse Kinematics, Navigation, and Frontier Planning have to be derived and constructed as well as the path planning for the robot to be able to properly navigate the workspace as described.

II. METHOD

A. Mapping

1) Costmaps: A Costmap is a representation of the robot's environment that divides the environment into three different sections, unknown space, free(non-obstacle) space, and obstacle space. Each of these sections are assigned different values, free space being the most cost efficient(least cost) and obstacle spaces being the least cost efficient(most cost) to travel.

An image processing pipeline is later used on the occupancy grid to create an occupancy grid with a gradient padding around obstacles that makes up for the width of the robot. Meaning, the center of the robot should not pass into the gradient padding as this will mean the edges of the robot will collide with an obstacle.

To achieve the gradient padding, the occupancy grid goes through an image processing pipeline that dilates the grid, applies a Gaussian blur to the grid, and lastly, normalizes the occupancy grid.

2) RRT and Frontier Search: The frontier search pipeline begins with the use of a rapidly exploring random tree (RRT) search, which utilizes five self-extinguishing, probabalistically complete local RRT search nodes, accompanied by a persistent, global RRT node. Leaves are identified as frontiers by comparing the occupancy grid data at the node to the parent branch. If the leaf is within unexplored space and the parent branch is in explored space, the leaf is marked as a frontier, and is published to a collective topic shared by the frontier detectors. This approach is effective because RRT searches are heavily biased towards exploring unknown regions. Vertices with larger Vornoi regions are identified to be closer to the frontier regions of the map, and since RRT is based off the principle of expanding from the nearest neighbor, the search tree is more inclined to explore these regions.[1]

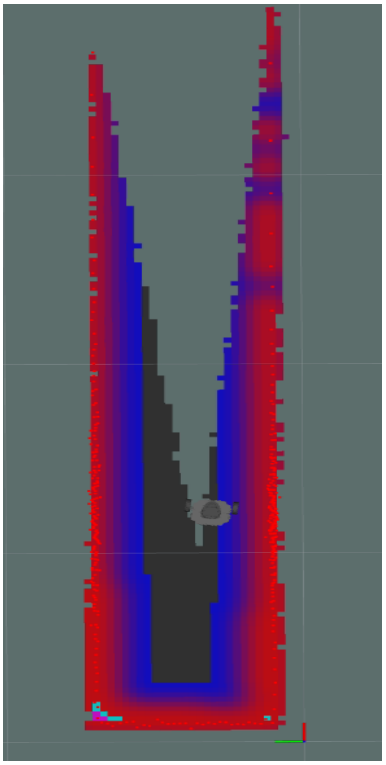

Fig. 1. Global Costmap

In addition to the RRT frontier search, an OpenCV image-based approach is taken, which identifies the edges via the Canny[2] algorithm, drawing contours on the processed map. The two operations are applied to the map via a boolean mask. The frontiers are then identified by calculating up to the third order moments of the strong-edged image.

A filtering node is subscribed to the filter detection topic, which utilizes a K-Means Shift algorithm to fit and cluster the frontiers into discretized bins (not the grid). The centroids

[1]"Autonomous Robotic Exploration Based on Multiple Rapidly-Exploring Randomized Trees." IEEE Xplore, https://ieeexplore.ieee.org/document/8202319.

[2]"An Improved Canny Edge Detection Algorithm." IEEE Xplore, https://ieeexplore.ieee.org/abstract/document/6885761.

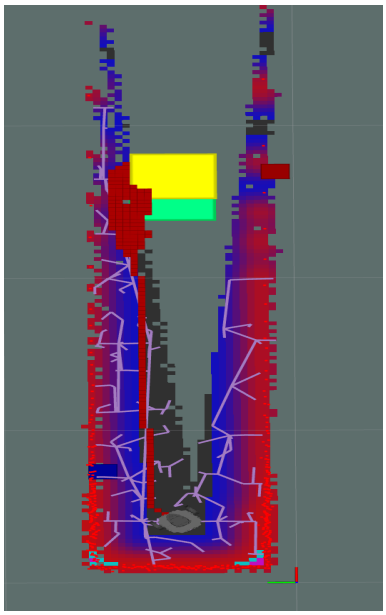

Fig. 2. Rapidly Exploring Random Trees

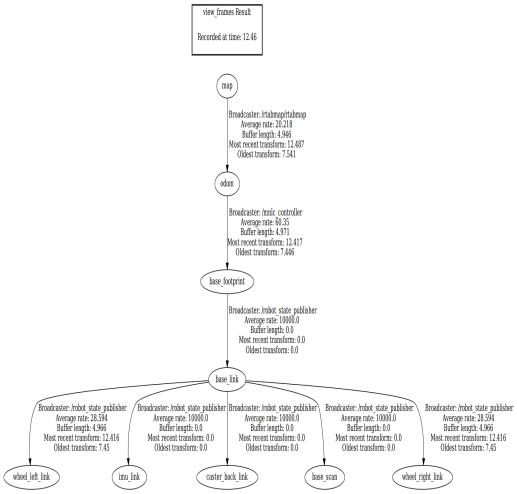

Fig. 3. TF Tree

of the clusters can then be retrieved to be analyzed further by the filter.

Each centroid that passes through the clustering algorithm is then assigned a weight based off of the estimated amount of information the robot would gain if it moved to said centroid. This information gain algorithm analyzes all nodes within a radius of the centroid, and considers the surrounding nodes' cost in the costmap, the distance to the current robot position, and the potential to discover new centroids (which is determined by the area of the unknown grid cells within the information radius of the centroid). This process is repeated for each centroid, and those which fail to meet a certain threshold or are obstacles are removed from the filtering process. The remaining centroids are then published, to be assigned to the robot by the frontier assigner.

The frontier assigner subscribes to the filtered frontiers' topic, and applies an information revenue-based sorting algorithm which accounts for information lost by the robot if it moves to the centroid, and information gained if the robot goes to the centroid. This sorting algorithm is similar in nature to the one employed in the filter.

3) Real-Time Appearance-Based Mapping: A ROS package called RTAB-Map(Real-Time Appearance-Based Mapping) uses an RGB-D SLAM approach based on a global loop closure detector with real-time constraints. RTAB-Map can be used to create a 3D map, however, in the scope of this lab, the more important ability is its ability to create a 2D occupancy grid map for navigation.[3] The map generated by RTAB-Map is then given to RVIZ, another ROS package that is used to visualize the robot's environment and can handle, among many other things, a 2D Navigation Goal(a 2D coordinate point that is the desired position and heading of the robot).

B. Navigation

The intent of the navigation pipeline is as follows: Using the globally scoped path generated by the global planner, the local planner will attempt to satisfy robot kinodynamic constraints to navigate to the next pose in the local window (defined by the local costmap), accounting for any errors in the control loop with a series of recovery methods implemented in the active action callback. This methodology leads to a segregated global and local frame, with which the global apparatus view the big picture, and the local apparatus focus on immediate concerns in the robot periphery.

1) Global Planners: A* and D* Lite were the two global planners implemented for use in the system. [4]

2) Local Planners: Multiple potential local planners were devised which were intended to follow the general path created by the global planner, while taking into account the dynamics of the obstacles in the environment and the kinodynamic constraints of the robot. The two primary approaches were a pure pursuit controller[5] and a dynamic window approach. [6]

C. Localization

1) Odometry: The odometry of the system is calculated via a Kahlman Filter applied between the Turtlebot's lidar readings and the encoder readings from the Turtlebot's '/Odometry' topic. This filtered odometric transformation is applied through RTAB-map. The idea of fusing IMU data with the raw encoder readings was also considered, but the benefit was deemed too little compared to the associated

[3]"RTAB-Map." IntRoLab, https://introlab.3it.usherbrooke.ca/mediawiki-introlab/index.php/RTAB-Map.

[4]D* Lite - Idm-Lab.org. http://idm-lab.org/bib/abstracts/papers/aaai02b.pdf.

[5]edf42001, et al. "Paper: Implementation of the Adaptive Pure Pursuit Controller." Chief Delphi, 11 Aug. 2018, https://www.chiefdelphi.com/t/paper-implementation-of-the-adaptive-pure-pursuit-controller/166552.

[6]Ri.cmu.edu. https://ri.cmu.edu/pub$_files/pub1/fox_dieter_1997_1/$.

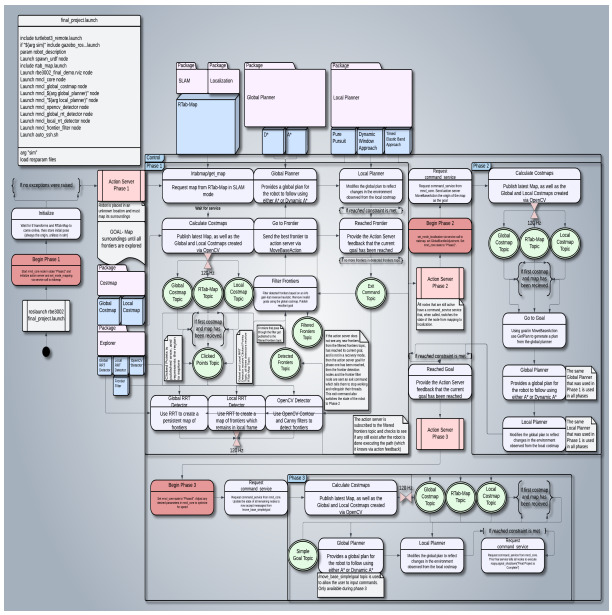

Fig. 4. MNL Controller State Diagram

effort, considering that most of the odometry error in the Turtlebot is due to horizontal wheel-slip that an IMU's readings would likely not be able to account for regardless of being fused.

D. System Architecture and Control

1) MNL Controller: The MNL Controller serves as the command center for the system- synchronizing callback timing, node initializations, phase transitions, and performing active error checking.

2) ActionLib and Service Structure: ActionLib actions are the primary communication provider for state transitions and is the backbone of the MNL architecture. Three action clients exist within the MNL Controller, each of which request results from their respective phases. By outsourcing functionality to more specialized nodes, the controller is able to act as a state machine for the program, and manage phase transitions with ease.

Additionally, several services are used, which allow for short information requests and error checks that would require too much overhead to warrant an action client or topic subscription. Initial map error checks and global plan requests rely on this service architecture to function.

III. RESULTS

A. Phase 1

The first phase of the project involves the use of the mapping, navigation, localization, and control methods, to search for frontiers, select one, and drive to it efficiently and without hitting obstacles.

Once this process is done once, the process is done again until all frontiers are discovered or outside of the map.

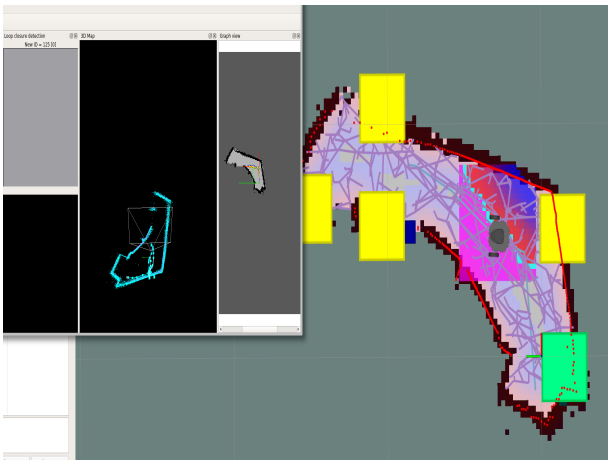

Fig. 5. Real Mapdata from Phase 1

B. Phase 2

After Phase 1, we have a fully explored map. For this phase, the robot must traverse back to the starting position still without hitting obstacles.

To do this, the robot will use its navigation and control methods to calculate a path from its current position to the starting position and then follow that path back to that position. The current position of the robot is recorded by updating the odometry of the robot and the starting position of the robot is recorded prior to starting Phase 1.

C. Phase 3

For this phase, we will need to navigate to a specified point on the map. The robot will need to successfully navigate to a point on the map selected in RVIZ using the 2D nav goal

Since the map has already bean explored and fully mapped, path planning a navigation will be used to get the robot to drive to the newly selected point.

IV. DISCUSSION

The goals of this project was to use all of the tools and information taught in lecture and labs to collaboratively code a robot that path plans efficiently and explores a map autonomously. To accomplish this, the robot must successfully complete all three phases.

As seen in 1, the robot was capable of discovering and mapping its surrounding by using the lidar sensor that supplies enough information to RTAB-Map to generate an occupancy grid of the robots surroundings. This occupancy grid is sent through an image processing pipeline where it is dilated, blurred, and normalized.

As seen in 2, the robot was able to detect a frontier (yellow) and select one point (green) near the frontier. This point is then sent through the path planning algorithm where control algorithms are used to keep the robot on the path.

In 3, the tree of coordinate frames of the system using the tf packages is displayed.

In 4, the node structure of the MNL controller system is displayed.

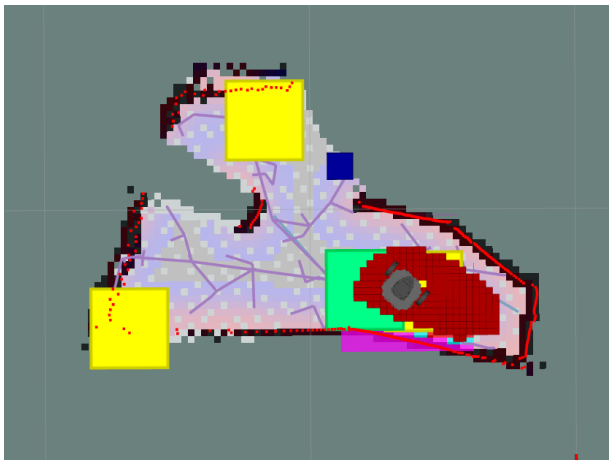

Fig. 6. Real Mapdata from Phase 2

5 shows the result of using RRT, frontier searching, and path planning to generate the fully expanded mapdata.

Ultimately, the robot plans a path to either a point on the map or a point next to a frontier. The path is calculated quickly and consists of the shortest travel distance while avoiding obstacles. The robot does all of this without given(hard coded) the information of its environment but still given the ability to detect its surroundings.

V. CONCLUSIONS

In this project, we used mapping, path planning, navigation, controls, and localization in order to explore a workspace. We implemented this with the use of the Python programming language and its many libraries as well as the utilization of ROS and its packages. With this, the TurtleBot3 can navigate the workspace it is placed in and perform the three phases described in order to map and navigate the said workspace.

Our project contributions are as follows:

Planning	Evan, Spencer, Ethan
Code	Evan, Spencer, Ethan
Experimentation	Evan, Spencer, Ethan
Results	Evan, Spencer, Ethan
Write-Up	Evan, Spencer, Ethan

REFERENCES

[1] "Autonomous Robotic Exploration Based on Multiple Rapidly-Exploring Randomized Trees." IEEE Xplore, https://ieeexplore.ieee.org/document/8202319.

[2] "An Improved Canny Edge Detection Algorithm." IEEE Xplore, https://ieeexplore.ieee.org/abstract/document/6885761.

[3] "RTAB-Map." IntRoLab, https://introlab.3it.usherbrooke.ca/mediawiki-introlab/index.php/RTAB-Map.

[4] Ri.cmu.edu. https://ri.cmu.edu/pub_files/pub1/fox_dieter_1997_1/.

[5] edf42001, et al. "Paper: Implementation of the Adaptive Pure Pursuit Controller." Chief Delphi, 11 Aug. 2018, https://www.chiefdelphi.com/t/paper-implementation-of-the-adaptive-pure-pursuit-controller/166552.

[6] D* Lite - Idm-Lab.org. http://idm-lab.org/bib/abstracts/papers/aaai02b.pdf.