Autonomous exploration using RRT-SLAM

Aditya Jagani
Robotics Engineering
Worcester Polytechnic Institute
Worcester, United States
arjagani@wpi.edu

Rohan Walia
Robotics Engineering
Worcester Polytechnic Institute
Worcester, United States
rwalia@wpi.edu

Ali Abdelhamid
Robotics Engineering
Worcester Polytechnic Institute
Worcester, United States
aabdelhamid@wpi.edu

***Abstract*— This project addresses the motion planning subproblem of efficiently exploring unknown environments using RRT-SLAM to obtain the map of the environment and further use RRT* algorithm for navigating the mapped configuration space. Exploration is process of selecting target points that yield the biggest contribution to a specific gain function at an initially unknown environment. RRT is a widely popular path planning algorithm because it can efficiently search nonconvex, high-dimensional spaces by randomly drawing samples from the search space and building a space-filling tree that is inherently biased to grow towards large unsearched areas. We will implement RRT to explore various environments and create their maps using the Kobuki Turtlebot on Gazebo on the Robot Operating System (ROS) and observe its functionality for open and cluttered spaces.**

Keywords— RRT-based Exploration, SLAM, ROS Navigation Stack, Turtlebot, Husky, Gazebo, Rviz

I. INTRODUCTION

Exploration of an unknown environment is a fundamental step for autonomous navigation in the field of mobile robotics. Mobile robots need knowledge of their surroundings to navigate around obstacles and reach their destination. This information can be gathered by sensors like 2D/3D-lasers to construct a map for the environment by exploring regions of interest locally and subsequently concatenating several local maps to form a much larger map of the area called a global map. Simultaneous localization and mapping (SLAM) is a technique that is most used to build a map of the environment. The prime challenge in autonomous exploration is how robots should plan its path in an unknown setting to ultimately explore the whole region while minimizing total travelled distance.

To answer a question central to autonomous exploration: Given present knowledge about the world, where should we move the robot to get the most information?[3] This can be accomplished by learning the concept of sampling-based path planning algorithms of which the most popular is the Rapidly exploring Random Tree (RRT).

RRT is a path planning algorithm that randomly explores the configuration space to find a valid path from source node to destination node. RRTs are constructed incrementally by expanding the tree to a randomly sampled point in the C-space while satisfying constraints such as robot kinodynamic or obstacle constraints.

We intend to implement 'Active SLAM' for exploration, which is, basically deciding where to move next in order to build an efficient map, using RRT algorithm on a Kobuki Turtlebot. RRT will efficiently and continuously plan and update paths such that they cover the greatest number of unexplored regions in the environment with minimum travelling time, distance, and cost to construct the map. Kobuki Turtlebot is a differential drive non-holonomic mobile robot with an integrated laser range sensor to perceive obstacles in the environment in the form of point-clouds. The laser is used for scan-matching laser particles on the obstacles which inform the robot of obstacles geometric properties in the configuration space and maps those obstacles relative to robot's frame. This technique is called mapping. We also use odometry and Inertial Measurement Unit (IMU) data to accurately determine the robot's relative position and orientation in the map to prevent deconstruction of the map. The simulation shall be performed in Gazebo and visualized in Rviz using the ROS framework.

II. LITERATURE REVIEW [4]

The different steps in RRT algorithm which are to be implemented in to build a solution are:

A. Initialization

The RRT algorithm begins by first initializing a node as a root which is nothing but robot's initial position. This position is usually provided by the user. A step size is determined by the user as well, which places the next node at a distance of delta from the previous node. Having a higher step size makes exploration faster but also reduces the chance of covering narrow paths and sometimes might not converge on the goal node which would be surrounded by obstacles. Having a smaller step size makes converging to goal node more probable and enables exploration in cluttered spaces. However, having a very low delta increases goal convergence time and strains computational resources, therefore an optimum delta should be decided by the user.

B. Random State

A random position Xnew is generated in the C-space using random sampling. Once the random sample is generated, it is confirmed that the node is not obstructed by any obstacle using collision detection techniques.

C. Nearest neighbor

The newly generated random node measures its distance with all the nodes in the graph and declares the node with the least distance as its parent node.

D. New State

The New State selects a new configuration that is away from the nearest node by an incremental distance (delta) toward the direction of Xrand. The new vertex and the edge are added to the tree and this process repeats until the goal radius is reached. The new node (x,y) is selected based on the following distance metric:

$$x = x_0 \mp l\sqrt{\frac{1}{1+m^2}} \quad (1)$$

$$y = y_0 \mp ml\sqrt{\frac{1}{1+m^2}} \quad (2)$$

Where m is the slope of the line joining Xnew and Xrand and *l* is the branch size.

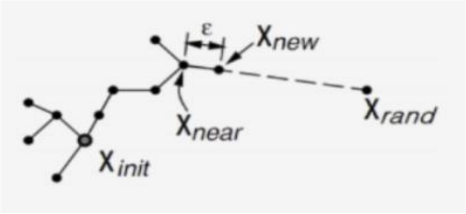

Figure 1: Mechanism of tree expansion

As represented in Figure 1. Xrand is replaced with the coordinates of the goal node for 10% of the step iterations. This is to allow for a bias to expand the tree towards the goal. The 10% (user defined) margin of bias (90% exploration) ensures that the algorithm continues to search free space even if the next step is not always in the direction of the goal. This is one form of the explore/exploit ratio generally faced by many algorithms. The higher the bias weight, the faster and more direct the (possible) solution which comes at the cost of a higher risk of being stuck in a difficult spot and not being able to explore a way out.

RRT Pseudo Code

```
Qgoal //region that identifies success
Counter = 0 //keeps track of the iterations
lim = n //number of iterations algorithm should run for
G(V,E)  //Graph containing edges and vertices, initialized as empty
While counter < lim:
     Xnew = RandomPosition()
     If IsInObstacle(Xnew) == True:
          Continue
     Xnearest = Nearest(G(V,E),Xnew) //find nearest vertex
     Link = Chain(Xnew,Xnearest)
     G.append(Link)
     If Xnew in Qgoal:
          Return G
Return G
```

The basis of SLAM is formed on probability and hence at any point the robot doesn't know for certain its position but takes its best guess using the information of current and previous states to determine its pose in the map. The motion model for the vehicle can be described in terms of a probability distribution on state transitions in the form:

$$P(xk \mid xk-1, uk)$$

That is, the state transition is assumed to be a Markov process in which the next state xk depends only on the immediately preceding state xk−1 and the applied control uk. The localization problem consists of estimating the probability density over the space of all locations. The Markov Localization framework combines information from multiple sensors in the form of relative and absolute measurements to form a combined belief in the location.

III. Simulation Setup /Implementation

Along with updates to the sections above, the deliverables for this timeframe were generally split into three sections: RRT, SLAM, and PID Control; each involving an implementation as grounds of proof.

The progress of the RRT implementation in Python so far is summarized as follows,

Assumptions:

- The workspace for this implementation consists of an arbitrarily (user defined) size and a predefined number of obstacles that are assigned randomly across the defined workspace
- In place of this defined workspace, we expect to feed the algorithm a binary occupancy tree/map instantaneously from the SLAM algorithm

General Construct:

- [Front End] PyGame library allows for an extremely efficient visualization tool which can be used to represent the C-space of the turtlebot. Every possible coordinate location on the simulation map can be translated to an x,y location on a PyGame window of predefined size (x,y). Start, goal, free, and obstacle nodes can all be represented on the window and recorded in the graph. Pygames circle and line functions are also useful for constructing nodes and edges efficiently
- [Back End] Lists are used to define the three major pieces of information regarding every node (x, y, and parent ID). Of course, each of these lists must be updated with entries matching the location of the corresponding node. Python's insert/append and pop built in methods allow for adding and removing nodes from lists, respectively. These lists are the backbone of the algorithm and are what PyGame visualization tools use to assign colors and shapes to the window

Key Functions:

- Distance functions for calculating the euclidean distance between two points for use in identifying the

nearest neighbor and for measuring if the new random node is further than the a predefined step distance or if the goal is within that distance

- Random Node Generator function for generating x and y coordinates that are within the defined map dimensions
- Collision detection functions that are split into two segments: the first function detects whether a randomly generated node happens to lie in an occupied/obstacle node and the second function checks for edges that intersect/collide objects by way of interpolation
- Step function for using the randomly generated node (or the goal node for 10% of the iterations) and taking a step of predefined distance from the nearest neighbor to that new node. Of course, the step function calls on the connection (edge creation) between the nearest neighbor and the step: if the new node is in a collision space or if the edge to take that step crosses an obstacle, a new random node is generated and the exploration continues
- Explore function for using the random node generator and the collision detection function to branch out
- Bias function which operates just like the explore function with the main difference being that the step is attempted towards the goal and not towards the randomly generated node

The simulation environment has been setup in Gazebo using the turtlebot3 simulation base code provided at [6]. (The robot of choice is a burger turtlebot3, which is based on the ROS implementation of a 2-wheel differential drive robot with a ball-and-socket caster wheel in the front.

Another part of this project focuses on building the map implementing various type of SLAM algorithms to analyze their performance with diverse indoor environments, varied robot velocities, localization accuracy over time and other such metrics to deduce the most robust method that could be implemented on the robot. Instances where robot is traversing in places with repetitive or similar features like a large open space, it might end up losing its sense of position in the environment and therefore, we are focused on deducing a SLAM technique that could solve this problem.

Gmapping was implemented and configured in the test environment on the ROS framework. To improve the localization of the robot, we have also fused sensor data from odometry and IMU while also working on estimating position of the robot using the scanned Laser data to lessen the position covariance.

IV. Modified RRT Exploration [5]

A. Main layout

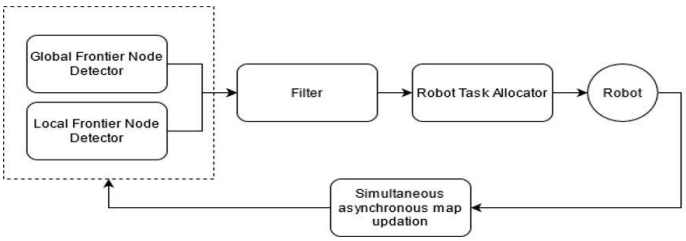

Figure 2: Main layout of Modified RRT Exploration Technique

Frontier nodes are the nodes in the RRT that separate known region from the unknown region in the task space.

Global frontier node detector: Keeps track of frontier notes throughout entire explored area.

Local frontier node detector: Explores frontier nodes in the immediate vicinity of the robot. Resets whenever a new node is added.

Filter: Reduces number of frontier points by selecting the center of clusters of all available frontier points. Additionally, it helps in removing old frontier points (which are no longer in the unexplored region), and invalid frontier points (which cannot be reached by the robot)

Robot task allocator: The robot task allocator uses a cost-benefit tradeoff to assign frontier points to the robot.

B. Task Allocation

Task allocation is performed based on a cost benefit tradeoff based on the following properties of the frontier point in consideration –

1. **Navigation Cost:** Distance between current position and frontier position.
2. **Information Gain:** Representation of unexplored area within a certain radius to be explored for a frontier point.
3. **Revenue:** Net gain associated with a frontier point, calculated using Navigation cost and Information Gain.

Each of these properties is calculated as follows –

- **Navigation Cost:**

$$N(x_{fp}) = \|x_r - x_{fp}\|$$

where –

x_r: current robot position
x_{fp}: frontier point position

- **Information Gain:**

$$I(x_{fp}) = n_{cells} * a_{cell}$$

where –

x_{fp}: frontier point position
n_{cells}: total cells in the information radius
a_{cell}: area occupied by each cell

- **Revenue:**

$$R(x_{fp}) = \lambda * h(x_{fp}, x_r) * I(x_{fp}) - N(x_{fp})$$

where –

x_r: current robot position
x_{fp}: frontier point position
λ: Information gain weight (user-defined)
h: hysteresis gain
I: Information gain
N: Navigation cost

The hysteresis gain is calculated as follows –

If $||xr - xfp|| > h_{rad}$

$$h(x_{fp}, x_r) = 1$$

Else

$$h(x_{fp}, x_r) = h_{gain} \; where \; h_{gain} > 1$$

Hysteresis gain dictates how well a robot is motivated to favor unexplored environments. A higher hysteresis gain would encourage the robot to explore frontier nodes in its vicinity, which would reduce the chances of overlapping. h_{rad} is a user-defined value which is set based on user experience/experimentation. The value of h_{gain} should be greater than unity to encourage the robot to explore frontier nodes closer to it.

V. Simulation Results

A. Map Descriptions

Three maps were explored in autonomous and manual mode to compare the number of nodes discovered over time. The first map is a 'turtle' room with 6 cylindrical obstacles evenly distributed in the center.

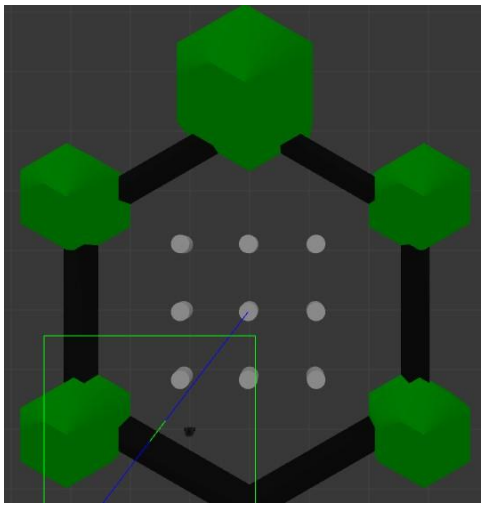

Figure 1: Map 1 shown in Gazebo

The second map mimics a house with obstacles in the form of tables, chairs and trash cans.

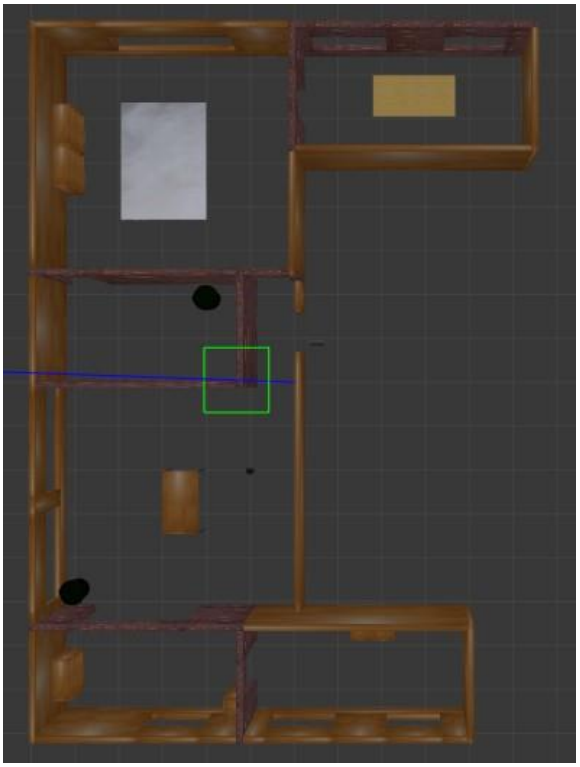

Figure 2: Map 2 shown in Gazebo

The third map is an open room with small barriers in the middle.

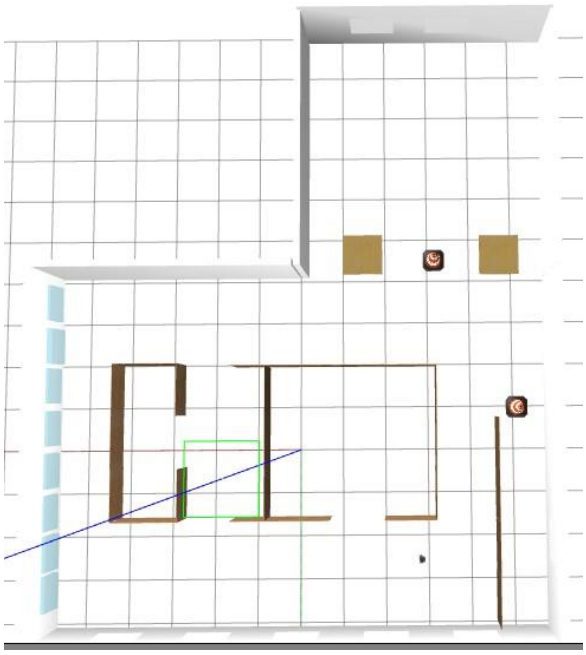

Figure 3: Map 3 shown in Gazebo

B. Map coverage (Before and After)

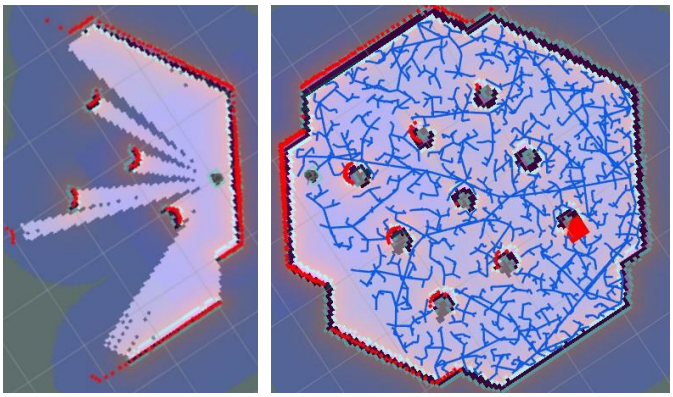

Figure 4: Before (left) and after (right) coverage for Map 1

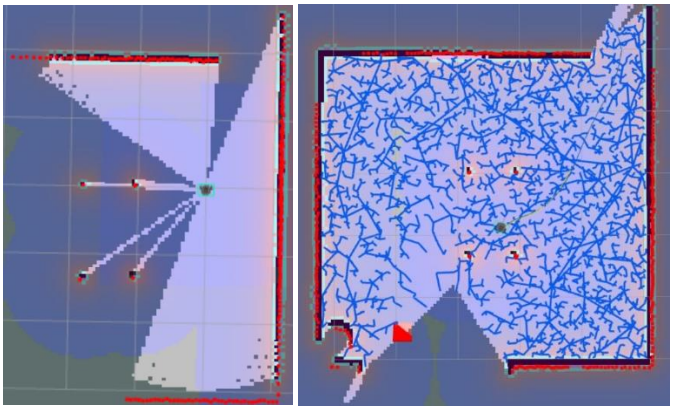

Figure 5: Before (left) and after (right) coverage for Map 1

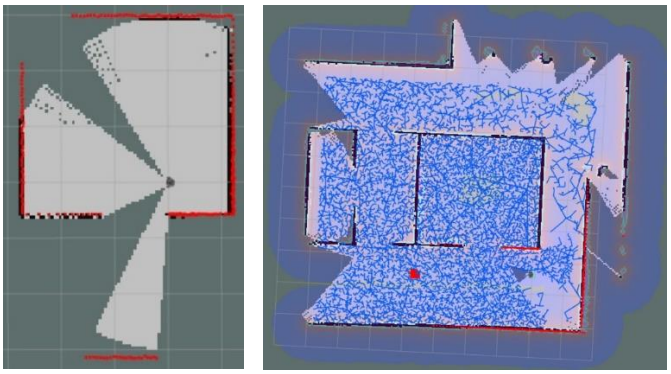

Figure 6: Before (left) and after (right) coverage for Map 1

C. Comaprison: Autonomous vs Manual Coverage of Points over Time

The following graphs shows how autonomous navigation fairs in comparison to manual navigation -

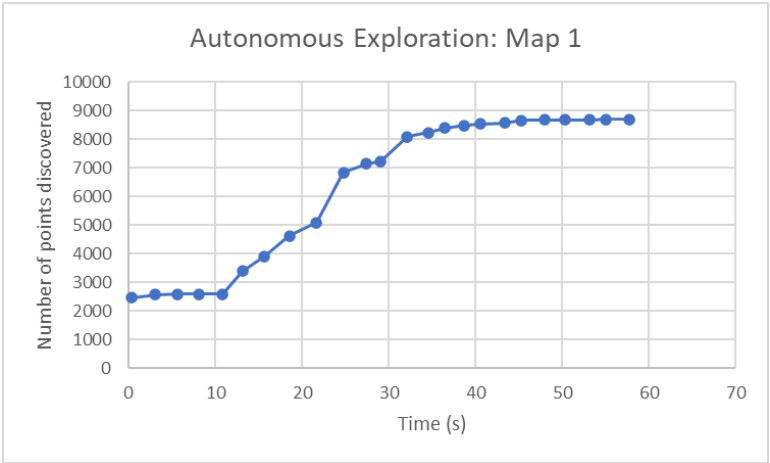

Figure 7: Number of points discovered over time for autonomous exploration of map 1

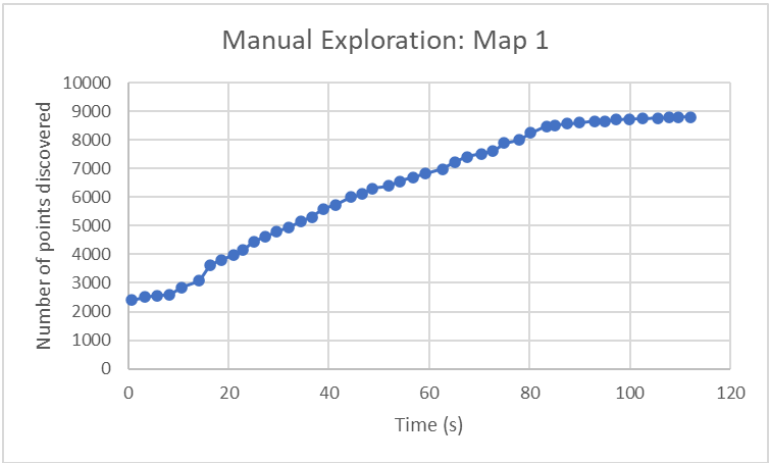

Figure 8: Number of points discovered over time for manual exploration of map 1

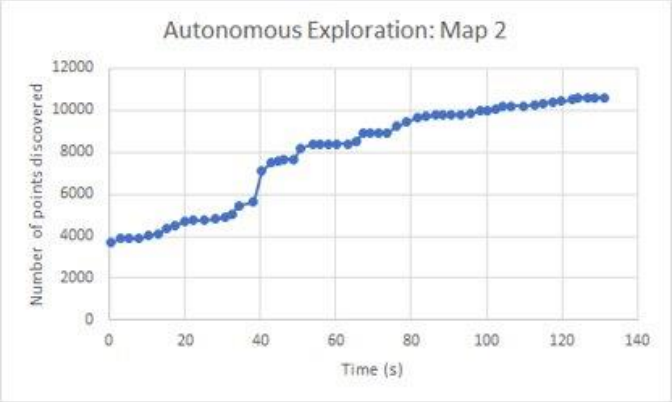

Figure 9: Number of points discovered over time for autonomous exploration of map 2

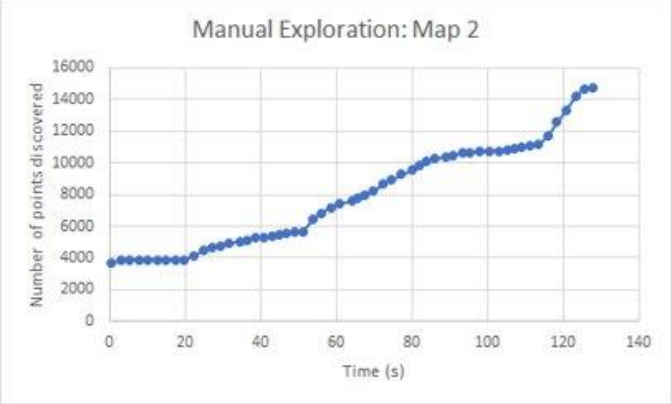

Figure 10: Number of points discovered over time for manual exploration of map 2

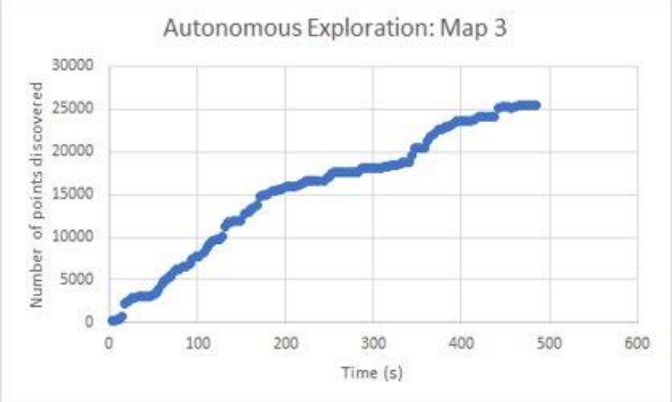

Figure 11: Number of points discovered over time for autonomous exploration of map 3

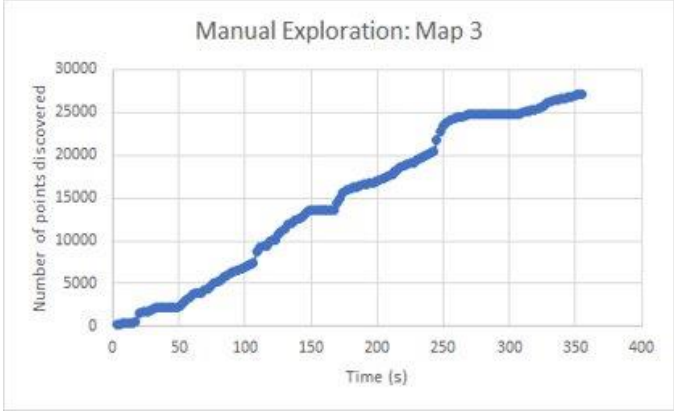

Figure 12: Number of points discovered over time for manual exploration of map 3

VI. Conclusions and Future Work

Based on the comparison of autonomous and the manual exploration of the three different maps, it can be said that autonomous exploration is comparable to manual exploration (maps 2 and 3), and sometimes better (map 1). However, it should be considered that manual exploration has the added advantage of the birds-eye view being available to the user. This makes it incredibly easy to control the robot based on visual feedback. For autonomous exploration, the robot does not have the map beforehand. It builds it on the fly.

The current software utility allows for single robot control. In future, multiple robots can be deployed to build and update their own local RRT, while another robot/communication server keeps track of the global RRT [5].

Research [7] shows that hysteresis gain is not the best metric to enforce the robot to choose the closest frontier points. A better metric might be used to improve this behavior.

VII. Acknowledgment

We would like to thank Mr. Kavit Shah who serves as our teaching assistant for the RBE 550 course at Worcester Polytechnic Institute. He gave us valuable insight to prepare and present our approach to solve the problem presented in this paper. We hope to seek his continued support as this project progresses through the semester.

VIII. References

[1] Marija Dakulović, Sanja Horvatić, Ivan Petrović, Complete Coverage D* Algorithm for Path Planning of a Floor-Cleaning Mobile Robot, IFAC Proceedings Volumes, Volume 44, Issue 1, 2011

[2] A. Le, V. Prabakaran, V. Sivanantham, and R. Mohan, "Modified A-Star Algorithm for Efficient Coverage Path Planning in Tetris Inspired Self-Reconfigurable Robot with Integrated Laser Sensor," Sensors, vol. 18, no. 8, p. 2585, Aug. 2018.

[3] A. Topiwala, P. Inani and A. Kathpal, " Frontier Based Exploration for Autonomous Robot," arXiv, June 2018.

[4] Huang, Yifeng & Gupta, Kamal. (2008). RRT-SLAM for Motion Planning with Motion and Map Uncertainty for Robot Exploration. 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS. 1077-1082. 10.1109/IROS.2008.4651183.

[5] H. Umari and S. Mukhopadhyay, "Autonomous robotic exploration based on multiple rapidly-exploring randomized trees," 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2017, pp. 1396-1402, doi: 10.1109/IROS.2017.8202319.

[6] https://github.com/ROBOTISGIT/turtlebot3_simulations.git

[7] R. Simmons, D. Apfelbaum, W. Burgard, D. Fox, M. Moors, S. Thrun, and H. Younes, “Coordination for multi-robot exploration and mapping,” in Proceedings of the AAAI National Conference on Artificial Intelligence, Austin, TX, 2000, pp. 852–858.