Final Year Project Report

Path Planning in SLAM

Mukhtar Bazylkhanov, Aida Sultan

A thesis submitted in part fulfilment of the degree of

BSc in Robotics and Mechatronics

Supervisor: Dr. Ton Duc Do

Department of Robotics and Mechatronics
Nazarbayev University

May 2, 2024

Table of Contents

Abstract

Simultaneous Localization and Mapping (SLAM) is a technology that helps an autonomous robot to localize itself in the unknown environment creating a map and further navigating without any human interference. Within the SLAM technology we are able to make detailed maps of the environment both in outdoor and indoor spaces where GPS signals are not available. Overall, SLAM is highly efficient technology that can help robots to properly navigate, inspect and monitor industrial environments autonomously through proper path planning. Identifying the shortest collision-free path and bypassing obstacles quickly is done by path-planning that applies different algorithms and their combinations. Due to the challenging environments, sensor limitations and discrepancies in localization estimation, it can be hard for robots to efficiently bypass and arrive at the desired destination. Therefore in our thesis we aim to familiarize ourselves with the path planning algorithms to fit the turtlebot. Our research focuses on individually examining the pivotal path planning algorithms (Djikstra, A* and RRT) to test it in the application of turtlebot to identify which algorithm is suitable for this robotic device. The research is done by analyzing existing literature on path-planning to further apply the knowledge in an experiment. It was found that despite the better intrinsic capabilities of A* and RRT algorithms, the Dijkstra appears to be the most advisable algorithm for the turtlebot).

Acknowledgments

We would like to express our deepest appreciation to our Professor - Dr. Ton Duc Do. This endeavor would not have been possible without his proper guidance and willingness to help us on the every stage of the project.

We are also deeply indebted to our Teaching Instructor Yussuf for explaining the project procedure, helping us with building our milestones and our project constraints, sharing beneficial knowledge to succeed in our Thesis.

Chapter 1: **Introduction**

Simultaneous Localization and Mapping (SLAM) is a new technology in the field of Artificial Intelligence and mobile robotics that solves the issue of localization and mapping the environment when a prior map is not available. Existing only for almost 40 years, the technology is applied in many directions as of wheeled mobile robots to medical operators, rescue robots, flying objects (drone delivery), autonomous driverless vehicles, urban planning, underwater submarines, telecommunication devices to identify the location and many other machines such as robot vacuum cleaners and driverless cars, delivery machines and drones that may be a part of a daily use soon [5]. Moreover, SLAM technology is also of a great use in a military sector by helping with reconnaissance, surveillance and target tracking. The robots with SLAM settings do not depend on the external factors and can function despite challenging and hostile environments. Therefore the development of SLAM technology is a step into an innovative future that benefits not a single field, but many from the essential ones as medicine and military to overall people's comfort in daily chores.

SLAM work procedure

Being a robot with an Intelligence system, it uses sensors to find its own location in a real-time and also making a map of its environment [8]. Then the position of the robot is identified through the data provided by sensors that build maps of the environment. After the position was obtained, the robots plans the trajectory of its path [5]. General procedure is presented in the Figure 1.1.

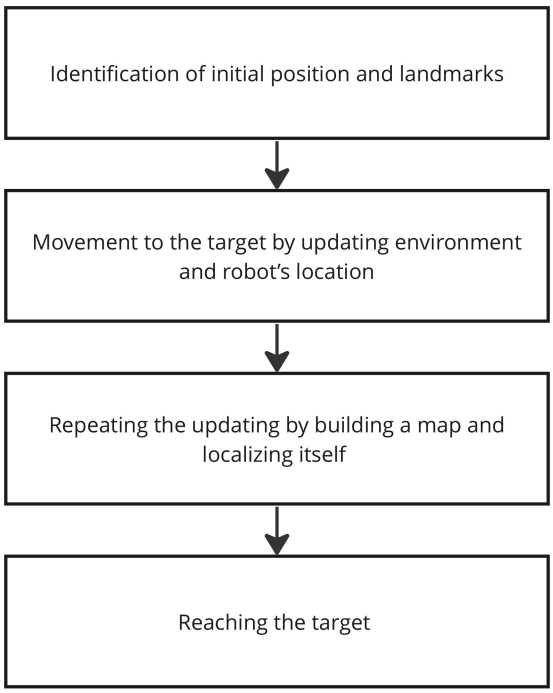

Figure 1.1: The SLAM framework flowchart

Path-planning

Being a multi-objective optimization problem, path planning in SLAM is in charge of finding the shortest path, the smallest distance to the target and bypassing the obstacles [9]. This research will discuss SLAM for indoor environments. As for outdoors we have a global positioning system (GPS) to easily localize the object through satellite positioning signals, we do not have one for indoor environments.

The issue of navigation in SLAM is of great importance nowadays. Having driverless vehicles, they still pose a risk as there is still not a single fully robust algorithm ensuring a total obstacle avoidance both in static and dynamic environments. The same thing for indoor mobile wheeled robots . This issue can be solved through proper path-planning. Due to the factors as working in dynamic environments, individual sensor limitations and possible errors in localization estimations, the path planning might provide wrong results. For that purposes, we primarily rely on sophisticated algorithms of SLAM specially adapted to different environments and cases. Path planning and SLAM algorithms are tightly related in a sense that it is one of the main factors that ensures the robustness of the navigation and achieving a collision-free path.

The thing with path planning is that it does not only require proper algorithms to identify the most efficient route, but also ensure that the robot will reach the destination despite the dynamic complications on the way and terrain changes. For instance, the Djkstra algorithm, knowing the map of the path, finds the shortest path, while A* integrates heuristic function to investigate the fastest route to reach the point. On the other hand, the RRT algorithm is much different from the above mentioned 2 methods, as it is more applicable in dynamic, unpredictable surroundings by utilizing stochastic approach. Being similar in a parameter of application in dynamic environments, Dynamic Window Approach (DWA) is also another beneficial method in dealing with active obstacles. Presenting this algorithms and thoroughly analyzing and comparing them, our thesis will deepen into intricacies of path planning of the SLAM technology assessing its methods based on the factors like robustness, efficiency, accuracy in their experimental performance. For that reason paper will mainly focus on three main algorithms: Djkstra, A* and RRT for both static and dynamic challenging environments. Currently, within the active research of SLAM, everyday people come up with new approaches as dynamic window approach (DWA) for local obstacle avoidance and algorithms as ant-colony, TEB algorithm, A*, RRT, D* and etc.[4]

In this paper, the 2 SLAM related issues were covered:

- examination of different path-planning approaches both in theory and practice to investigate each one to identify strength and weaknesses, applied environments applicable for the turtlebot;
- analysis of the process of multi-fusion of sensors evaluating its benefit. Due to the time limitations of the course, the multi-fusion part was not developed fully, however, given the theoretical knowledge in this paper, we plan to develop it further.

For those purposes, the report is organized as follows. Section 1 will provide description on path-planning algorithms, current approaches and overall, consult secondary research data to get an understanding of each algorithm. Then comparing them to different cases, we test the in the experiment and provide comparative analysis with a test of a theoretical knowledge.

Section 2 would present data on sensors. Though the paper does not fully cover multi-fusion, the background knowledge on sensors is still benefits the understanding the SLAM technology.

Chapter 2: **Background Research**

Multi-fusion

The SLAM framework operates through different sensors as LiDAR, Camera or IMU (Inertial Measurement Unit). For now, single sensor application is considered to be fully studied and being considered as "simple technology", so the research turned in favor of multi-fusion to increase the quality of the existing technology. As each sensor has its strengths and weaknesses it was proposed to incorporate them to get a better output, high precision estimations in a robot's positioning and mapping. for now, there was an experiment comparing different single sensors as of ZED binocular camera, a Real- Sense D415 depth camera, and a RoboSense16-line 3D LiDAR with color and 2D LiDAR and camera fusion system, that explicitly showed how sensor fusion highly contributes to the minimization of the error. According to the statistics, while the error of single sensors were up to distance error - 2.98, length error - 6.07 and width error 5.33, the fusion system reduced it to 0.23, 0.17 and 0.43 respectively [2]. In a further paragraphs sensors are individually introduced and current experiments of sensor integration would be thoroughly discussed to identify its current state and applicability in our own experiment.

Sensors:

According to the research articles, each single sensor has its advantages and limitations depending on a complexity of the environment. Here are the short descriptions of single sensors.

Generally, LiDAR is a common sensor in usage. Having good properties to ensure better precision in a large scale environments, it also creates high accuracy point cloud maps and localization. However, LiDAR has drawbacks in terms of firstly, differentiating visual details as color, shape and dimensions; secondly, operating much slower in scenes with reduced geometric features causing errors in calculating its pose; and thirdly, inability to update its location in a fast manner[2] [6][7].

On the other hand, Camera sensors excel in discerning visual details. Nevertheless, it is much reliant on a light and will be no use at all in case of darkness or rapid movement as the image will get blurred. (Wang) Moreover, camera has a limitation in the face of its intrinsic properties: "scale drift in monocular cameras, computational intensity in stereo cameras, and limited applicability of RGB-D cameras to indoor scenes" [7].

In contrast, IMU allows high-frequency, precise outputs in the odometry for short time periods and performs better in a combination with other sensors. For instance, according to the current experiments in fusing IMU and Camera, it was investigated that IMU helps to minimize the dependence of the camera on lighting quality providing better efficiency in tasks [6].

As applying only single sensor in SLAM is already outdated and rather simple activity, fusing them would bring stronger, more exact results. Therefore, for now, incorporating single sensors in a one SLAM framework is a main purpose and source of optimization of technology.

Path-planning

Path-planning in SLAM is a key parameter for the functioning of the technology properly, as it helps robot to choose the most optimal route from the original point to the desired destination without obstacles or helping to bypass them easily. Above we have presented the steps of a SLAM framework, but now we will dive into the steps of path planning part of the technology [2][4]:

1. Perception: The robot collects data on its environment: data on distances, visual features, obstacles and overall terrain.
2. Localization: Using sensor capabilities robot finds itself on the map and localizes.
3. Map Updating: moving in the given space, the robot constantly improves the accuracy of the map over time paving the way for better localization.
4. Path planning: after the precise results obtained from perception, localization and map updating stages, the robot plans ahead its trajectory. This is the point where robot incorporates different algorithms. The specific flowchart for each algorithm would be presented below under the description for better understanding.
5. Motion Control: Having the path planned, the robot now controls variables as steering, speed and kinematic and dynamics for the accurate navigation.
6. Obstacle Avoidance: in case there is an obstacle that was not identified in prior, the robot re-plans its route and updates the map until reaching the final point.

The path planning offers 2 different methods: global path planning and local path planning. To be short, if global path planning takes long-term decisions in advance planning everything from the start till the end by analyzing all possible routes, local path planning is about short-term decisions taken in real-time, whenever the robot meets with an obstacle [4][2].

Global path planning

As the name suggests, the global path planning concerns creating a whole trajectory of the robot from the initial position to the target, taking into account the complete space. Here, it essentially focuses on reducing cost function meaning to find the path with the shortest distance and also with a better time efficiency. Mentioning that global path planning is used for bigger range spaces, it is mostly applied for Djkstra and A* algorithms as in those algorithms the mapping environment if fully or partially known.

Djkstra Algorithm

Djkstra algorithm is another synonym for "the shortest path" that was introduced by Djkstra in 1959 [2]. Basically, the Djkstra algorithm scans its surroundings layer by layer until detecting the target point [3]. Starting from the point X(start) it builds paths to each vertice of the environment computing all the possible ones. This way it finds out the shortest one[2]. Even though it can identify the shortest path it takes much time, because the algorithm computes each possible distance to find the shortest path, while we need only one [3][1]. It does not utilize any heuristics, therefore and explores all possible routes. For that reason, when the time cost is important, the Dijkstra is not the optimal choice.

The following figure illustrates the work of Dijkstra algorithm. Here, initially we introduce two different sets A and B, where A presents an initial point to different vertices and records the vertices that does not lead to the shortest path, while B comprises points other than the values stored in A. Then, having the beginning point, Dijkstra finds distances from other points to the starting position. If the smallest distance from A is found, it is extracted from the set B and added to A as the shortest path. Furthermore, the distance is updated from

each vertex in the B set till the A point. The procedure is reiterated until we reach all shortest paths and compare them to each other. [10]

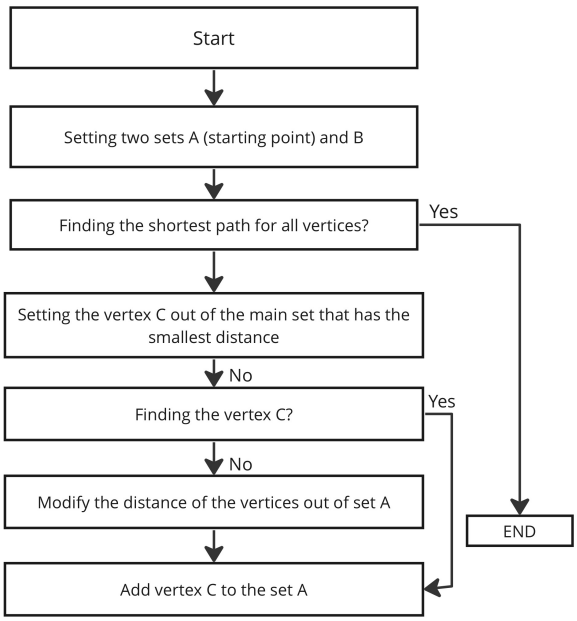

Figure 2.1: The flowchart of Djkstra Algorithm [10]

A* star Algorithm

Another algorithm is A* - the fastest one developed by Hart in 1968 [1]. This algorithm applies heuristic function of a Djkstra algorithm paving a way for its optimization [3].

A heuristic function, in the context of the SLAM, implies the algorithm that efficiently browses possible path variants running them through a stated function to find the efficient one. It's main goal is to provide the path with a cheaper cost in terms of time and the distance traveled.

For A* algorithm, heuristic function aims to identify the shortest distance to the final destination. However, given the heuristic function, it does not imply that A* is a perfect algorithm. Because it does not always provide the best route, but still ensures the cost efficient trajectory for the robot. Reducing the computational effort for the robot and saving time, A* is beneficial for large range spaces, where we cannot waste time computing all the possible points as in Dijkstra. A* runs the heuristic function and once it finds the path that leads to the shortest path, it implements it, while Dijkstra firstly analyzes everything and only finding the best one, it starts to move to the target [4].

Its work can be explained by following heuristic function:

$\mathbf{f(X) = g(X) + h(X)}$ to evaluate the efficiency, cost of path. Here, g(X) - an actual cost; h(X) - expected one. So, this heuristic function works this way. For instance, at the state h(n)=0, it is Djkstra Algorithm, however as we get larger h(n)¿0, it does not analyze all the area spending much time, but rather limits its environment concentrating on the most optimal paths that it didn't yet cover [1][2]. Through this function, it allows A* to consider paths that it did not yet build, therefore being one of the fastest algorithms [3].

RRT Algorithm

Even though A* is considered to be one of the fastest algorithms, it is mostly used for low-

dimensional mobile robots and cannot carry out complex tasks. What in other hand, RRT appears to be able of. This algorithm is said to excel both in obstacle bypassing and planning the most efficient path to the target.

Regarding the work procedure of RRT, as the name says rapidly it explores the configuration space to derive a path. First of all, the origin and the target point are saved in memory, then RRT algorithm chooses a node in a collision-free path applying "sample function" at each iteration. The chosen node is then compared with the nearest node on the graph, so that the Euclidean distance (the length of the line) between the selected node and the nearest one is less than a constant value of the system, the selected node is replaced with a new point (x(new)) by adding a direct line to the graph. However if the distance is bigger than a constant, "steer function" replaces the node again by x(new) and places it on the direct line between the initial node and our nearest point. The procedure is iterated until specific amount of iterations or finding the needed path [9].

The following flowchart explains the logical steps of the RRT algorithm:

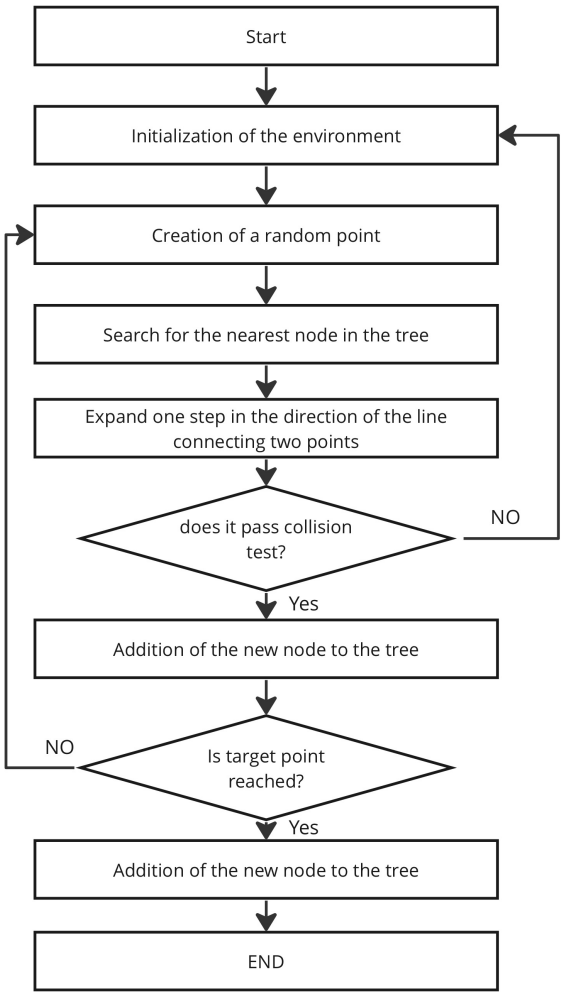

Figure 2.2: The flowchart of the RRT algorithm [11]

Local path planning

Though global path-planning is a common one, local path planning is essential in dynamic environments. For instance, in case we have moving obstacles it would be better to apply algorithm with a local path-planning in a DWA (Dynamic Window Approach) platform [3]. Local path planning works on short range trajectories providing smooth path in dynamic

Algorithms	Djikstra	A*	RRT
Advantages	For the static environment, it ensures the shortest path with an accurate navigation. It is easy to implement and the most reliable.	Cost efficient than Djikstra in parameters of time and memory as it prioritizes routes closer to the final point. Better adaptability than Djikstra to the environmental alteration	Highly efficient for the high dimensional spaces.
Disadvantages	Computationally intensive for large maps and not suitable for dynamic environments as it weakly adapts to the terrain changes	Even though is the fastest, it is highly dependent on heuristic function meaning if the function is wrong, the whole algorithm will provide invalid paths	The algorithm is not always perfect:sometimes it immediately find the desired path, but other time it may require program repetitions to accustom. It requires multiple iterations to work properly. Also, RRT does not always provide the shortest path.
Recommended cases	Static environments	Both static and dynamic environments	Perfect for dynamic environments and making real-time decisions. RRT can deal with many obstacles and topology changes.

Figure 2.3: Comparison table of three algorithms [1][9][10][14]

environments. Using sensor feedback , it helps robot to immediately respond to obstacles and make decisions in real-time. One of the popular algorithms of local path planning is DWA (Dynamic Window Approach) that we incorporated into our experiment. It is mostly effective in moving environments with many obstacles. Basically, it is an algorithm that makes decision based on robot's internal kinematic, dynamics and sensor capabilities and path's future state.

Dynamic Window Approach Algorithm

DWA (Dynamic Window Approach) is an algorithm of local path planning that accomplishes tasks under the dynamic environment map. Firstly, the state-space representation of the robot is identified: its position, orientation and velocity (linear and angular components). Furthermore, the algorithm defines "dynamic window" around the robot's current state, meaning the analysis of the different velocities the robot can have given the short period for overcoming current constraints in the map. So, the DWA tests several trajectories (path that robots can follow in the exact moment) according to the given linear and angular velocities based on the criteria of the proximity to the target and physical constraints. Finally, the algorithm chooses the trajectory that is the closest to the destination, but free of obstacles. Overall, The laser detects obstacles in front of the robot and generates data on the distance between the robot and an obstacle, and finally DWA turns on the autonomous navigation and obstacle bypassing abilities of the robot [3].

Chapter 3: **Approach and Methodology**

The experimental part of the project includes two phases: 1st phase, simulation of the path planning algorithms in Gazebo and 2nd phase, simulation of the algorithms using real life turtlebot. The results will be used to make comparisons of algorithms to identify the most effective, in addition simulations and real life experiments will be compared to ensure the accuracy of the results, and how may robot differ in its behavior using those algorithms in different environments (simulation and reality).

The simulations are run on the melodic version of ROS 1. From sensors only 2d Lidar is simulated.

Small maze was constructed for the real robot. The experimental environment has no windows for natural outside light, it has good artificial lightning over the whole room, which will be beneficial for LIDAR performance. Since any lighting changes will be affecting the performance of the sensor, affecting the movement of the robot. The maze is 370 x 195 cm, width of the corridors are minimum 40 cm, for the robot to have enough room for maneuvers. The maze is of a simple complexity, currently no need for complex patterns. Such environment is suggested to be beneficial in having smooth experimental work.

Continuously during the real experimentations some of the corridors, which are open in the known map were closed, so the robot needed to navigate and understand the environment in the real time

Every algorithm will be run at least two times from one side to another and in reverse. It also will be rerun in simulations. A 3d model of the Maze will be constructed and integrated into Gazeboo world.

Chapter 4: **Implementation & Execution**

The most challenging part of the project at this current moment is the implementation part. Initially the project faced some difficulties with preparing the software environment for the robot and simulation part.

Initially it was attempted to integrate new path planning algorithms from scratch on clear ros workspace. The difficulty of the task made us consider ready made workspace from github, such us https://github.com/SakshayMahna/Robotics-Playground/tree/main.

This workspace had already implemented A* and Dijsktra algorithms for turtlebot. Upon further work and were faced difficulty with calibrating the behavior of the robot to follow the trajectory given by the algorithms. The inability of the robot to adapt and constant replans made us reconsider the approach of our implementation.

It was decided to again start from scratch and build more clear and robust workspace which was successful. Thus, we managed to integrate RRT, Astar and Dijkstra path planning algorithms in our ros workspace of turtlebot.

The standard package of turtlebot3_navigation was upgraded with global_planner plugin and packages from https://github.com/ros-planning/navigation and https://github.com/mech0ctopus/rrt-global-planner/blob/main/src/rrt.cpp for usage of Astar Dijsktra and RRT path planning algorithms.

The experimental environment was constructed refer to the figures below.

First simulations were run using inbuilt gazebo open world, Figures 4.4, 4.5, 4.6.

Second simulations were run using the 3d Model of the Maze which was made from map using the SLAM function of turtlebot. And then additionally reworked with photoshop.

The real robot experiments were run after improving the parameters of DWA in simulations. Each algorithm had two runs, from one side to another and in reverse. Then algorithms were checked in changed maze configurations.

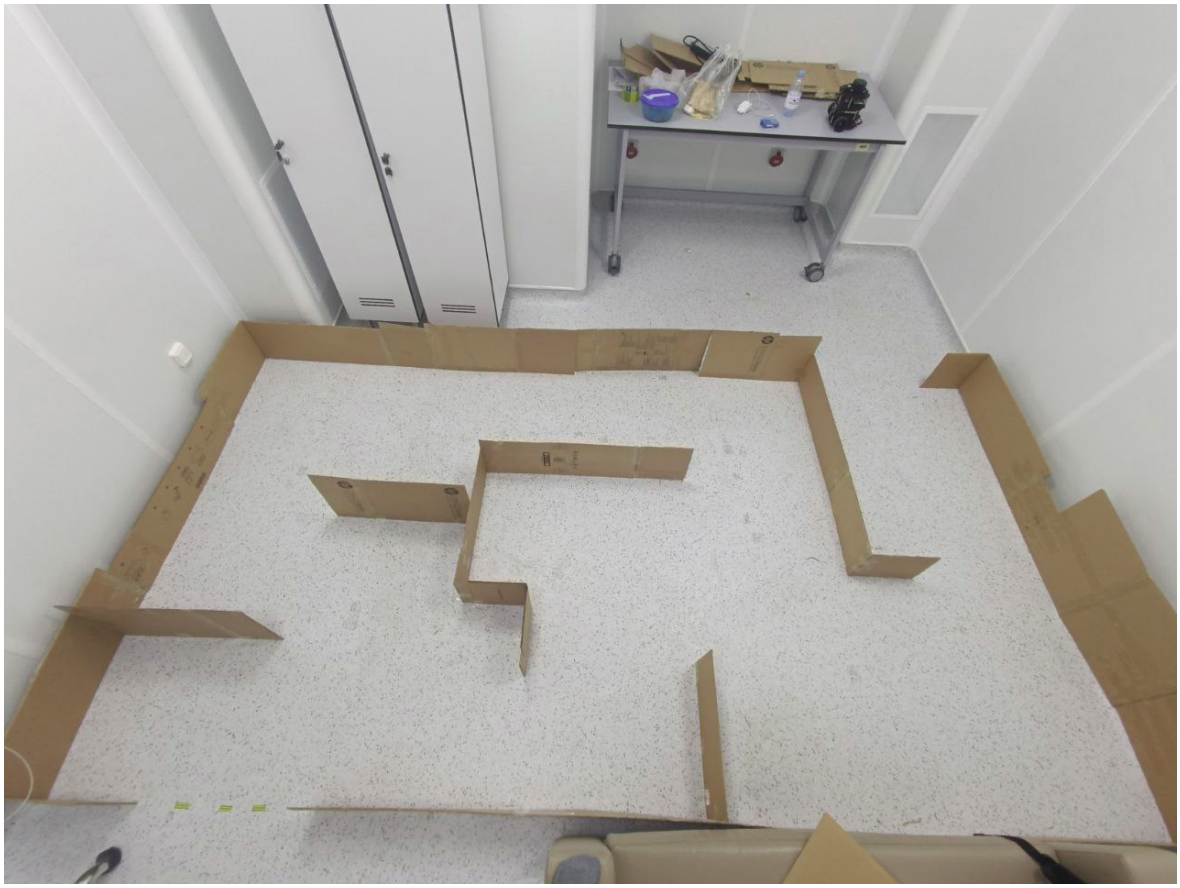

Figure 4.1: Maze

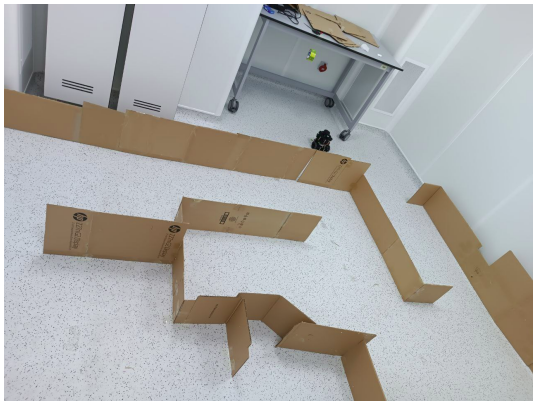

Figure 4.2: First maze change configuration

Figure 4.3: Second maze change configuration

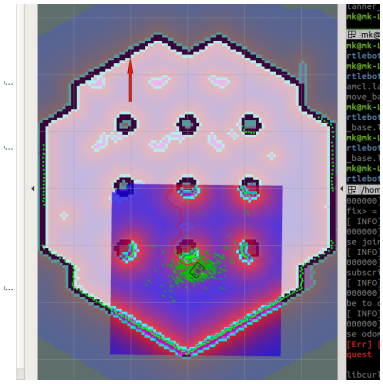

Figure 4.4: A-star

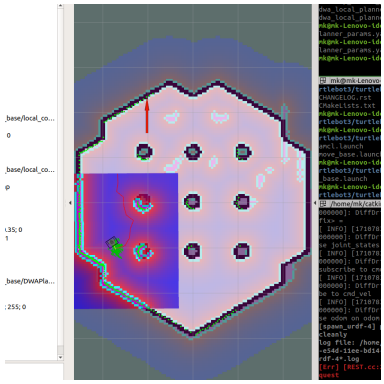

Figure 4.5: RRT

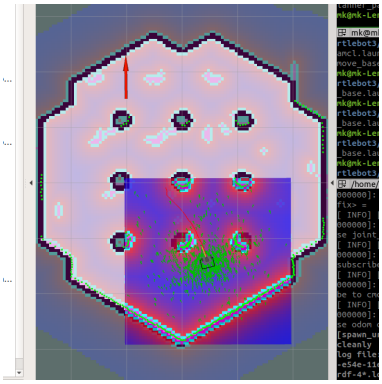

Figure 4.6: Dijkstra

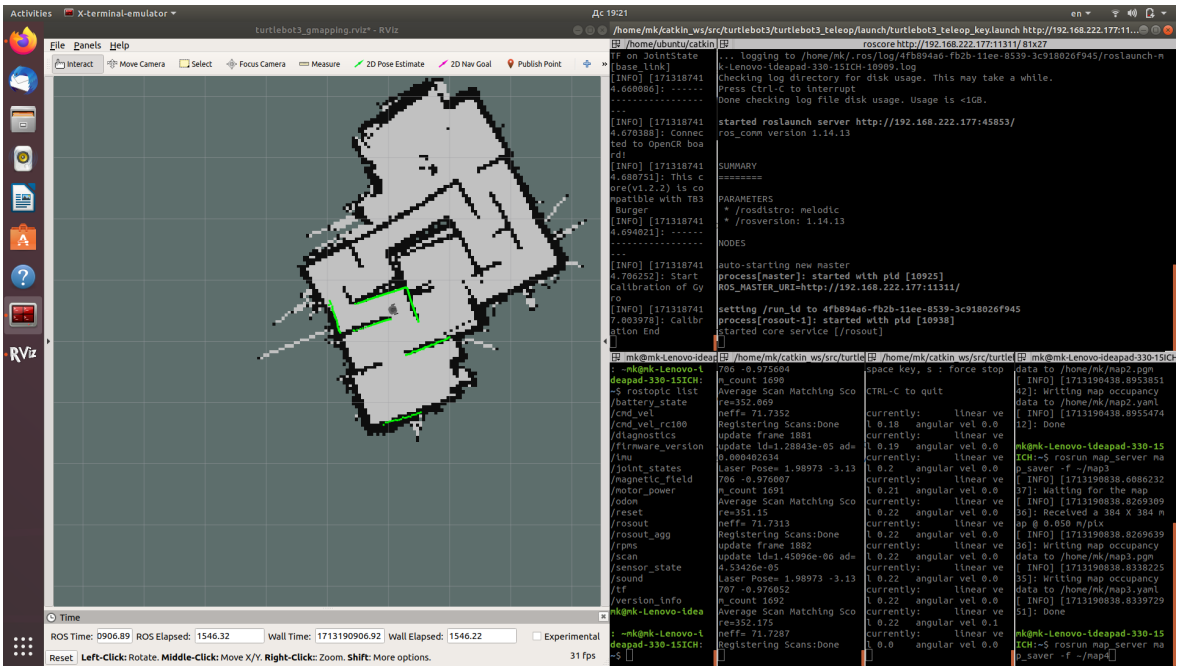

Figure 4.7: Mapping Stage

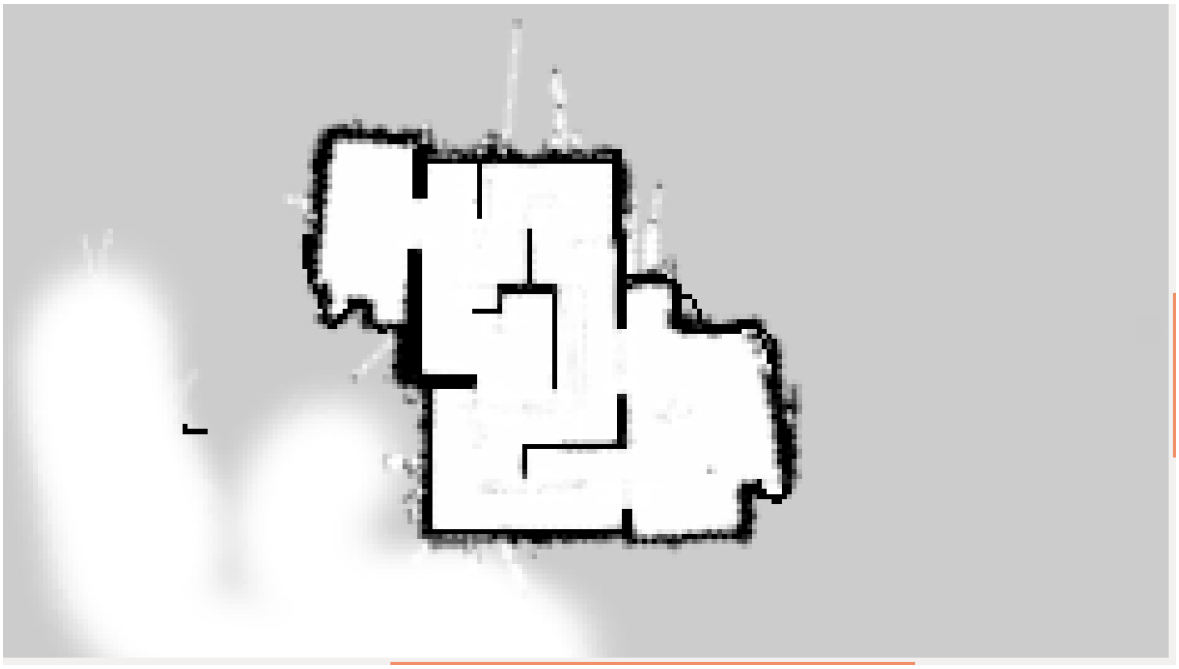

Figure 4.8: Maze map

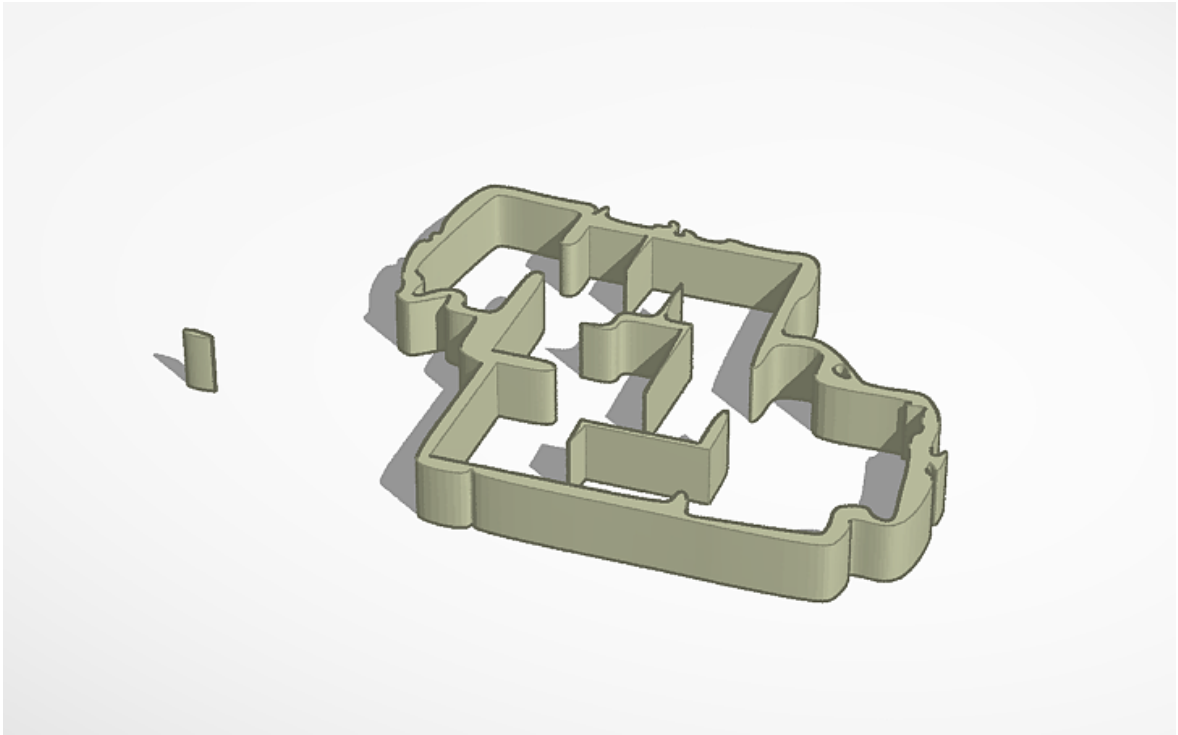

Figure 4.9: 3d Model of the Maze

Chapter 5: **Results and Discussion**

First simulations results are as follows in table 1.1.

Algorithm	Time (minutes)
RRT	01:02:19
Astar	01:22:67
Dijkstra	00:45:31

Table 5.1: Results of planning algorithms in the first simulations

The most time taken was using A*. The second place for the RRT algorithm, the robot didn't start the movement instantly and took time for planning the road. The best results was given by the Dijkstra algorithm.

After this simulation a lot work was redone especially with the DWA parameters and setting up the testing environment. From the results we acquired from second simulations, Figure 5.1,

Algorithm	Time (minutes)
A* (1st run)	1:13
A* (2nd run)	1:27
Dijkstra (1st run)	1:09
Dijkstra (2nd run)	1:30
RRT (1st run)	2:42
RRT (2nd run)	2:02

Figure 5.1: Second simulation results

it was seen that the difference between A-star and Dijkstra are small. In the first run Dijkstra was faster in 4 seconds. When in the second A-star was faster in 3 seconds. Eventually, we suggest that the reason is the following, for small-range environments, just like our small and simple maze, A* algorithm's heuristic function advantage is diminished since Dijkstra will cover the whole environment quickly[12][13]. The slowest algorithm was RRT, it is suggested to be due to the complexity of the calculation of possible trajectories. Also in the parameters we work with the number of vertices to be covered for having smoother trajectories in the complex environment, which increases the planning time. Additionally, turtlebot sometimes gives an error of DWA planner being unable to plan the motion in accordance with the global path provided by the RRT algorithm.

From the results we acquired from real robot standard maze, Figure 5.8, configuration we also notice almost same results for A-star and Dijkstra. First runs gave same time, however it needs to be noted that during the run turtlebot got confused just at the exit of the labyrinth, and hesitated at the end point, if there was no such occasion maybe it would reach the goal a couple of seconds faster, however still this is very similar results as in Dijkstra case. In the second run such hesitation and robot thinking were a little more in case of A-star, and with Dijsktra there was no such slowdowns. The similarity of the results we again attribute to

Figure 5.2: A-star simulated 1st run

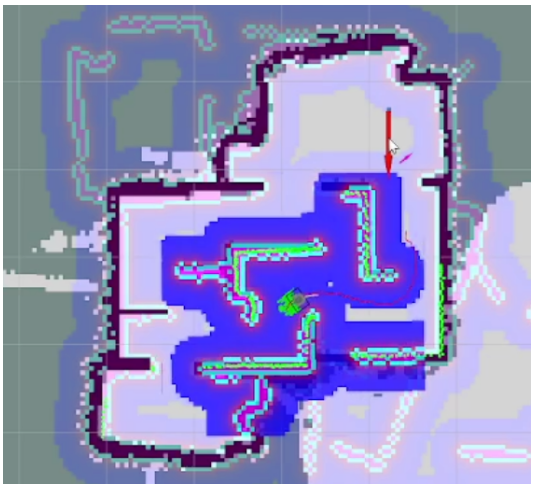

Figure 5.3: A-star simulated 2nd run

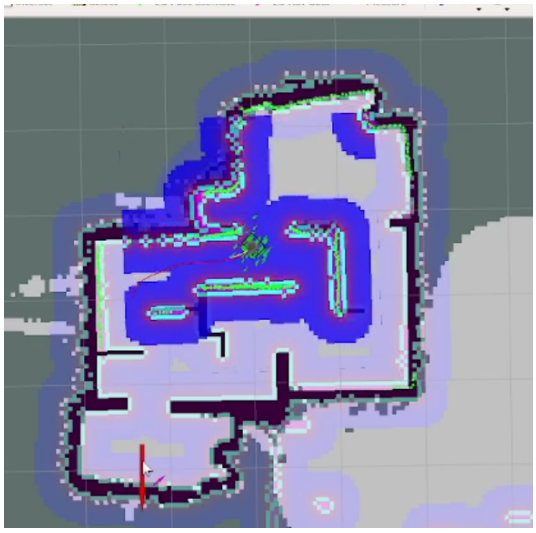

Figure 5.4: Dijkstra simulated 1st run

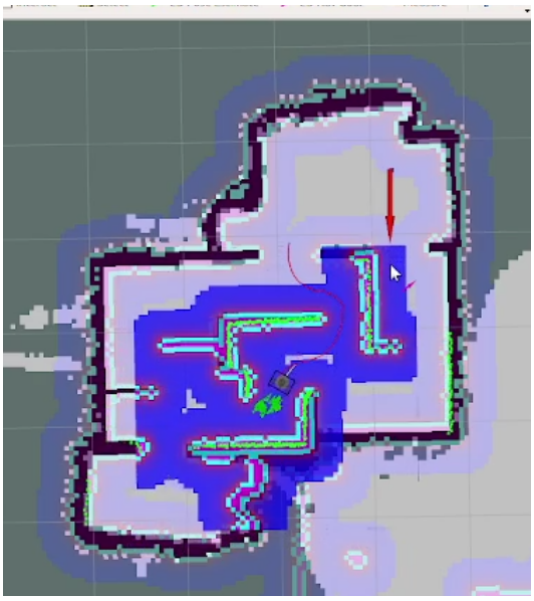

Figure 5.5: Dijkstra simulated 2nd run

the fact of having small testing environment where advantage of the A-star can not be fully expressed, and the Dijkstra has enough power to quickly explore the nodes. As for the RRT we have very slow results. This maybe reasoned with an intrinsic features of the RRT itself, as it can be caused by its random nature[14] and the calculations severity. The planning of the first trajectory takes a good chunk of time for the robot using RRT, moreover when the DWA fails to follow the global path of RRT, robot starts to generate a new trajectory, in the spaces where different paths can be chosen, robot again may hesitate on which path to follow.

In the changed maze configurations, Figure 5.15, first problem we encountered was the inability of the RRT to solve the maze with the newly closed corridors, however it needs to be noted that RRT was not working good enough in the standard maze as well, nor in the simulations nor in the real maze. Which is actually is not in line with the stated advantage of the RRT Figure 2.3, however it is also can be justified by its own disadvantage of being repetitive. Since that this experiment run has only Dijkstra and A-star results. As we can see from the Figure 5.3, we have slightly inconsistent results for both in comparison with

Figure 5.6: RRT simulated 1st run

Figure 5.7: RRT simulated 2nd run

Algorithm	Time (minutes)
A* (1st run)	0:55
A* (2nd run)	1:11
Dijkstra (1st run)	0:55
Dijkstra (2nd run)	0:57
RRT (1st run)	3:22
RRT (2nd run)	3:55

Figure 5.8: Real Robot in Default Maze

each other. Dijkstra went faster than A-star for almost 30 seconds, in the first maze configuration. However in the second configuration Dijkstra came slower for 20 seconds then A-star. It must be noted that A-star again had hesitations and slowdowns during the motion and planning, and the Dijkstra went smoothly, except in the second maze configuration run, Dijkstra spend a lot of time on trajectory planning in the start, this is attributed solely to the turtlebot feature, which is a common occasion for all the algorithms, but need to be noted that such behaviour is encountered more for A-star and especially RRT, for Dijkstra it is a rare occasion from our observation from working with turtlebot.

However it also important to note that with the not efficient DWA parameters, to be exact small cost scaling factor and too big inflation radius, the Dijstra global path tends to cut the turns, which makes hard the motion execution for the turtlebot, however with the same DWA parameters, A-star doesn't have such problem, it doesn't cut the turning trajectory and takes a good radius due to it's heuristics approach. Maybe this is the only part we notices in the small environment as a advantage of an A-star algorithm, however this feature was quickly overrun by simply bettering the costmap parameters for the turtlebot, by decreasing the inflation radius and increase cost scaling factor, so the trajectory of both algorithms became the same.

Figure 5.9: Real Robot A-star 1st run

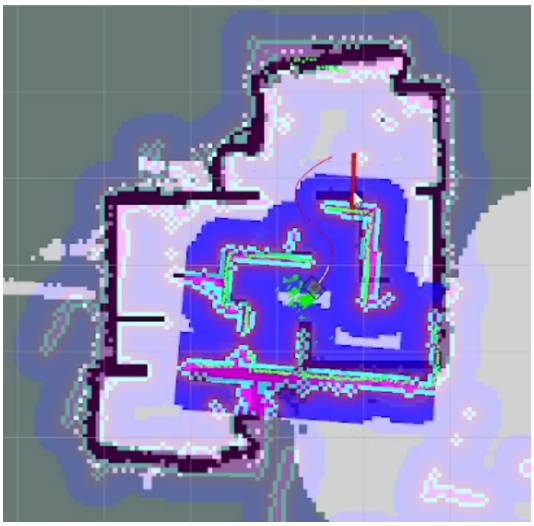

Figure 5.10: Real Robot A-star 2nd run

Figure 5.11: Real Robot Dijkstra 1st run

Figure 5.12: Real Robot Dijkstra 2nd run

Figure 5.13: Real Robot RRT 1st run

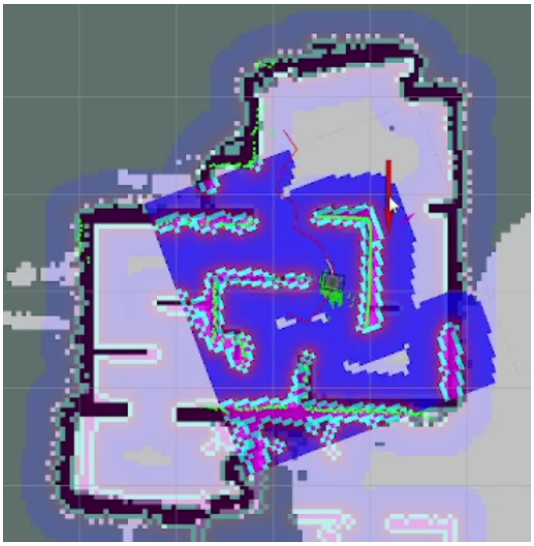

Figure 5.14: Real Robot RRT 2nd run

Algorithm	Time (minutes)
A* (1st run)	1:52
A* (2nd run)	1:40
Dijkstra (1st run)	1:28
Dijkstra (2nd run)	2:04

Figure 5.15: Real Robot in a changed Maze

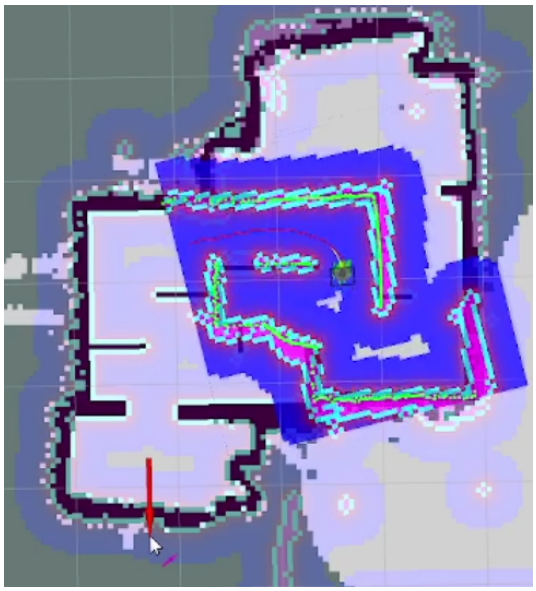

Figure 5.16: 1st Maze version A-star

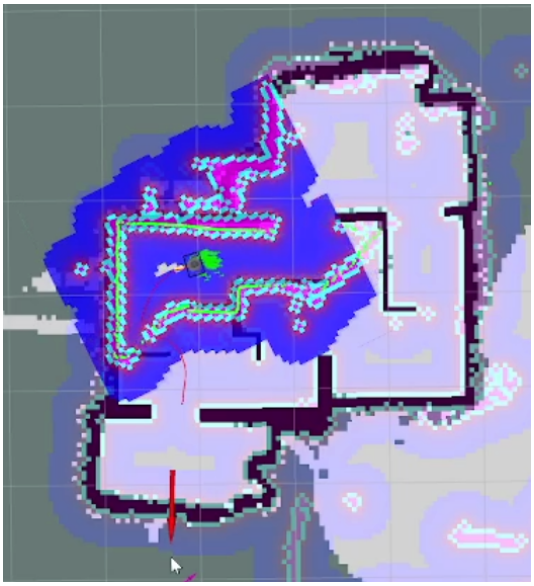

Figure 5.17: 2nd Maze version A-star

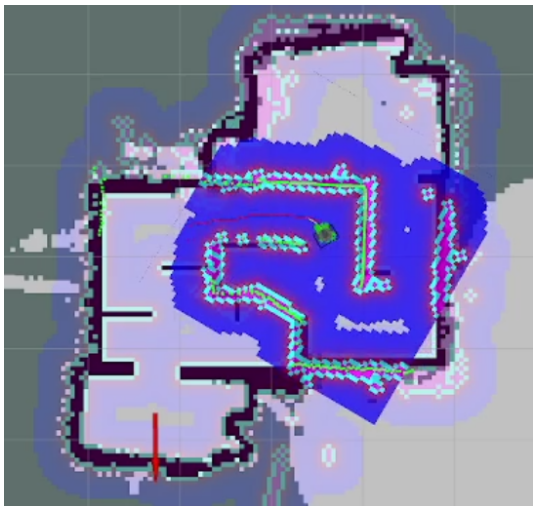

Figure 5.18: 1st Maze version Dijkstra

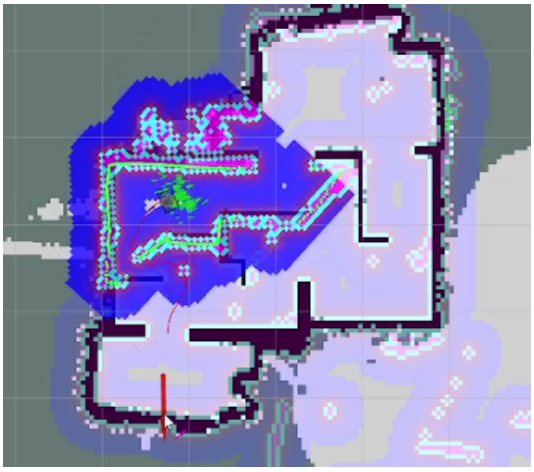

Figure 5.19: 2nd Maze version Dijkstra

Chapter 6: **Conclusion and Future Work**

During the experimentation certain aspects of every algorithm was partially observed. The random nature of RRT, makes it hard to incorporate to turtlebot, it needs a lot optimization and more understanding of how its parameters affect the trajectory planning and why DWA planner can not follow it. The A-star advantage over Dijkstra in the usage of the heuristic approach is diminished considerably due to certain environment settings, namely small-range and simple maze. Even though the heuristic function gives A-star ability to make better trajectory on the turning points in comparison with Dijsktra in case of turtlebot, such feature is easily overrun by introducing smaller inflation radius and big cost scaling factor. In addition, having that we observed multiple times the slowdowns in the motion and trajectory planning of turtlebot using A-star, more than compared to Dijkstra, we are suggesting that, at this point, having the settings of our environment, to be exact, small ranged labyrinth with relatively simple corridors, Dijkstra algorithm is most efficient for usage in turtlebot robot.

Within our work, we have arrived to 2 main weaknesses of our approach:

1) Having an aim of testing algorithms, we could not get time and resources to try different configurations such as different mazes, bigger maze area and more complexity, open spaces and open spaces with dynamic obstacles to verify each algorithm fully.

2) Despite the 1st limitation of our project, we tried to test each condition a bit and achieved working state of algorithms where the individual performances of Dijkstra, A* and RRT were recorded. So, we could not optimize algorithms for this moment, but examined their plain behavior.

For our future work, we will mainly focus on the aforementioned two constraints and based on that knowledge implement the multi-sensor fusion to optimize and get more sophisticated, accurate results. Having 2D LidAR and 3D Camera in the laboratory, we anticipate it would majorly contribute to the accuracy of mapping and path planning. In addition with GPS for outdoor mobile robotics projects.

Bibliography

[1] A. Kudriashov, T. Buratowski, M. Giergiel, and P. Małka, SLAM Techniques Application for Mobile Robot in Rough Terrain. Springer Nature, 2020.

[2] H. Ren, Y. Zhao, T. Lin, and J. Wu, "Research on multi-sensor simultaneous localization and mapping technology for the complex environment of construction machinery," Applied sciences, vol. 13, no. 14, pp. 8496–8496, Jul. 2023, doi.org/10.3390/app13148496.

[3] J. Ren, T. Wu, X. Zhou, C. Yang, J. Sun, M. Li, H. Jiang, A. Zhang, "SLAM, path planning algorithm and application research of an indoor substation wheeled robot navigation system," in Electronics, vol. 11, no. 12, pp. 1838-1859, June. 2022, doi.org/10.3390/electronics11121838.

[4] R. Roy, Y. Tu, Long-Jye Sheu, W. Chieng, L. Tang and H. Ismail, "Path planning and motion control of indoor mobile robot under exploration-based SLAM (e-SLAM)," in Sensors, vol. 23, no. 7, pp 3606-3627, March. 2023, doi.org/10.3390/s23073606.

[5] H. Taheri, Z.C. Xia, "SLAM; definition and evolution," in Engineering Applications of Artificial Intelligence, vol. 97, pp. 104032-104057, Nov. 2020, doi.org/10.1016/j.engappai.2020.104032.

[6] X. Wang, X. Ma, and Z. Li, "Research on SLAM and path planning method of inspection robot in complex scenarios," Electronics, vol. 12, no. 10, pp. 2178–2178, May 2023, doi.org/10.3390/electronics12102178.

[7] Y. Xia, H. Wu, L. Zhu, W. Qi, S. Zhang, and J. Zhu, "A multi-sensor fusion framework with tight coupling for precise positioning and optimization," Signal Processing, vol. 217, pp. 109343–109343, Apr. 2024, doi.org/10.1016/j.sigpro.2023.109343.

[8] X. Zhang, J. Lai, D. Xu, H. Li, and M. Fu, "2D Lidar-based SLAM and path planning for indoor rescue using mobile robots," Journal of Advanced Transportation, vol. 2020, pp. 1–14, Nov. 2020, doi.org/10.1155/2020/8867937.

[9] T.-V. Dang, "Research and design of a path planning using an improved RRT* algorithm for an autonomous mobile robot," *MM Science Journal*, vol. 2023, no. 3, Oct. 2023, doi.org/10.17973/mmsj.2023_10_2023051.

[10] L. Liu *et al.*, "Path Planning for Smart Car Based on Dijkstra Algorithm and Dynamic Window Approach," *Wireless Communications and Mobile Computing*, vol. 2021, pp. 1–12, Feb. 2021, doi: https://doi.org/10.1155/2021/8881684.

[12]D. Rachmawati and L. Gustin, "Analysis of Dijkstra's Algorithm and A* Algorithm in Shortest Path Problem," *Journal of Physics: Conference Series*, vol. 1566, p. 012061, Jun. 2020, doi: https://doi.org/10.1088/1742-6596/1566/1/012061.

[13]A. Candra, M. A. Budiman, and K. Hartanto, "Dijkstra's and A-Star in Finding the Shortest Path: a Tutorial," *IEEE Xplore*, Jul. 01, 2020. https://ieeexplore.ieee.org/document/9190342

[14] S. Eddine Hadji, S. Kazi, T. Howe Hing, and M. S. Mohamed Ali, "A Review: Simultaneous Localization and Mapping Algorithms," *Jurnal Teknologi*, vol. 73, no. 2, Mar. 2015, doi: https://doi.org/10.11113/jt.v73.4188.