Mini Project Report

on

Enhancing Obstacle Avoidance for Autonomous Vehicles

Submitted by

Nithish Chouti 21bcs074

Ratnesh Kherudkar 21bcs091

Shreyansh Tiwari 21bcs114

Udayini Vedantham 21bcs130

Under the guidance of

Dr. Prabhu Prasad

Assistant Professor, Computer Science and Engineering

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF INFORMATION TECHNOLOGY DHARWAD

30/04/2024

Certificate

This is to certify that the project, entitled **Enhancing Obstacle Avoidance in Autonomous Navigation**, is a bonafide record of the Mini Project coursework presented by the students whose names are given below during 2023-2024 in partial fulfilment of the requirements of the degree of Bachelor of Technology in Computer Science and Engineering.

Roll No	Names of Students
21*bcs*074	*NithishChouti*
21*bcs*091	*RatneshKherudkar*
21*bcs*114	*ShreyanshTiwari*
21*bcs*130	*UdayiniVedantham*

Dr.PrabhuPrasad
(Project Supervisor)

Contents

List of Figures

1 Introduction

Recent years have seen tremendous progress in the field of indoor autonomous navigation, which has significant implications for industrial uses. Using autonomous robotic systems to navigate complex interior office-like situations is one of the main areas of focus. These robots are being used in industrial settings more and more because they provide a practical way to reduce the high expenses of labor and other overhead related to manual material handling. Research suggests that implementing self-driving robotic trucks may save a plant's material handling costs by as much as 30

The capacity of autonomous robots to function in areas that could be dangerous for human workers is one of the main benefits of using them in industrial settings. This reduces the need for direct human intervention in risky scenarios. This feature adds to improved operational efficiency in addition to improving worker safety.

Numerous significant issues, including mapping, localization, and course planning, have been handled in the field of autonomous navigation through a variety of creative techniques. The idea, design, and modeling of a mobile robot that can autonomously navigate to predetermined locations while visually identifying and avoiding stationary impediments are the main topics of this study. The Robot Operating System (ROS) architecture is utilized in the creation of Simultaneous Localization and Mapping (SLAM) techniques along with sophisticated path planning algorithms.

Programmers may easily incorporate pre-existing solutions and tackle complex problems with ROS's sturdy architecture, which spares them from having to delve too deeply into the nuances of hardware. Because of its modular architecture, it is easier to link dissimilar parts to create a unified system that functions well. Additionally, ROS offers a networked environment in which nodes—representing different functionalities—communicate with a central hub, facilitating smooth robotic system coordination and control.

1.1 What is Autonomous Navigation?

Autonomous navigation is a critical component of autonomous vehicles, enabling them to navigate through complex environments without human intervention. Obstacle avoidance is a crucial aspect of autonomous navigation, ensuring the safety and efficiency of the vehicle's

movement. We focus on the use of ROS (Robot Operating System) and Turtlebot3 for practical testing and simulation testing, respectively, to enhance obstacle avoidance in autonomous navigation.

1.2 Robot Operating System (ROS)

ROS is a widely used open-source software framework for building and programming robots [1]. ROS's growing popularity in the robotics scientific community can be attributed to the addition of simulation support with Gazebo and visualization tools such as Rviz and Rqt graph [2]. To understand how the different features of ROS can be used in a basic remote-control application, a holistic approach is needed [3]. Turtlebot3 is a popular robot platform that is widely used for research and development in robotics [4]. The combination of ROS and Turtlebot3 provides a robust platform for testing and evaluating autonomous navigation systems, including obstacle avoidance

1.3 Sensors and Path Planning

.

Path planning is a critical component of autonomous navigation, enabling the vehicle to plan a safe and efficient path to its destination. Classical global path-planning methods include the Dijkstra algorithm [5], A* algorithm [6], Ant Colony Optimization (ACO) [7], Particle Swarm Optimization (PSO) [8], Rapidly Exploring Random Tree (RRT) [9] and so on. Among these algorithms, the A* algorithm is considered one of the most efficient algorithms for solving the shortest paths in static maps due to its search efficiency [10]. DWA algorithm (Dynamic Window Approach) is a fast path planning algorithm for robot navigation and mobile intelligences [11]. The basic algorithm idea is: firstly, to obtain the current state of the robot, such as: position, velocity, direction, etc.; secondly, to define the dynamic window, to determine its angular and linear velocity ranges to compose its maximum movable range, called the dynamic window; and subsequently, to generate the optimal obstacle avoidance motion trajectory [12][13][14].

1.3.1 LiDAR and Sensor Integration

LiDAR (Light Detection and Ranging) is a crucial sensor for autonomous navigation, providing accurate and high-resolution 3D data about the surrounding environment [14]. LiDAR data can be used for obstacle detection, mapping, and localization, which are essential for autonomous navigation. "Autonomous Obstacle Avoidance Vehicle using LIDAR and an Embedded System" The car just has one LIDAR sensor, allowing it to navigate in low-light conditions safely. Raspberry Pi and LIDAR sensor are the two main hardware components of this application [15].

1.3.2 Simulation Testing

Gazebo is a popular simulation platform that is widely used for testing and evaluating autonomous navigation systems [16]. Gazebo provides a realistic simulation environment that allows for testing and evaluation of autonomous navigation systems in a controlled and safe manner.

In a recent study, Jiang et al. [17] proposed a deep learning-based obstacle avoidance algorithm for autonomous vehicles using ROS and Turtlebot3. The algorithm was tested in both simulation and real-world environments, demonstrating its effectiveness in avoiding obstacles and navigating through complex environments.

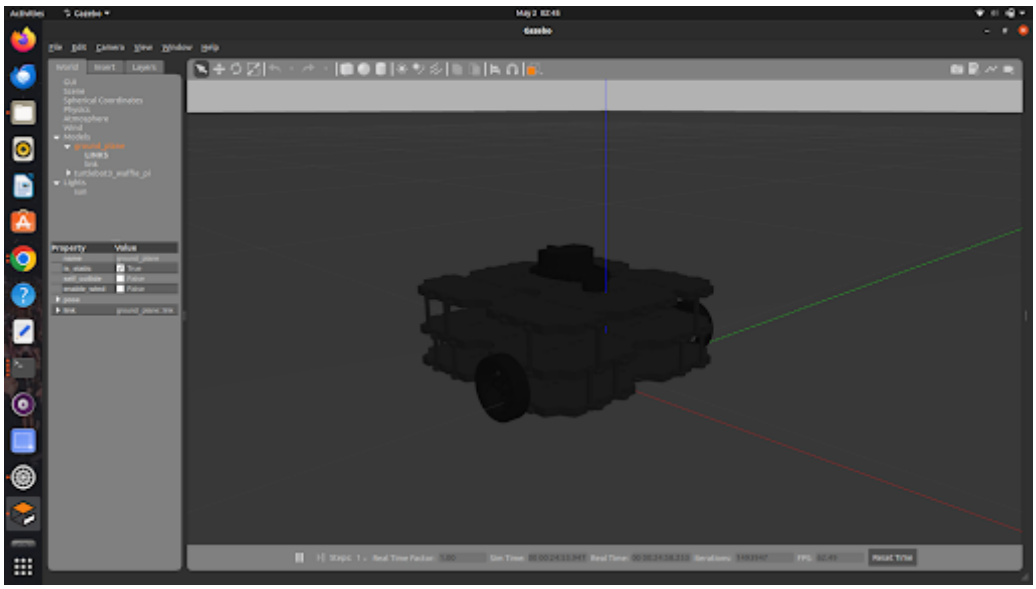

Figure 1. Gazebo Environment

Another study by Gao et al. [18] focused on the integration of LiDAR and visual sensors for improved obstacle detection and avoidance in autonomous navigation. The authors developed a multi-sensor fusion algorithm that was tested on a Turtlebot3 platform, showing improved performance compared to using a single sensor.

In conclusion, obstacle avoidance is a critical component of autonomous navigation, and the use of ROS and Turtlebot3 provides a robust platform for testing and evaluating autonomous navigation systems. The A* algorithm and DWA planner are two popular path planning algorithms that are particularly effective in finding the shortest path to a goal while avoiding obstacles. LiDAR is a crucial sensor for autonomous navigation, providing accurate and high-resolution 3D data about the surrounding environment. Simulation testing with Gazebo provides a realistic and controlled environment for testing and evaluating autonomous navigation systems.

2 Related Work

2.1 Literature Review

Autonomous navigation is a critical component of autonomous vehicles, enabling them to navigate through complex environments without human intervention. Obstacle avoidance is a crucial aspect of autonomous navigation, ensuring the safety and efficiency of the vehicle's movement. We focus on the use of ROS (Robot Operating System) and Turtlebot3 for practical testing and simulation testing, respectively, to enhance obstacle avoidance in autonomous navigation.

ROS is a widely used open-source software framework for building and programming robots [1]. ROS's growing popularity in the robotics scientific community can be attributed to the addition of simulation support with Gazebo and visualization tools such as Rviz and Rqt graph [2][3]. Turtlebot3 is a popular robot platform that is widely used for research and development in robotics [4]. The combination of ROS and Turtlebot3 provides a robust platform for testing and evaluating autonomous navigation systems, including obstacle avoidance.

Path planning is a critical component of autonomous navigation, enabling the vehicle to plan a safe and efficient path to its destination. Classical global path-planning methods include the Dijkstra algorithm [5], A* algorithm [6], Ant Colony Optimization (ACO) [7], Particle Swarm Optimization (PSO) [8], Rapidly Exploring Random Tree (RRT) [9], and others [10][11][13][14]. Among these algorithms, the A* algorithm is considered one of the most efficient algorithms for solving the shortest paths in static maps due to its search efficiency. The Dynamic Window Approach (DWA) algorithm is a fast path planning algorithm for robot navigation and mobile intelligences [11][12][13][14].

LiDAR (Light Detection and Ranging) is a crucial sensor for autonomous navigation, providing accurate and high-resolution 3D data about the surrounding environment [15]. LiDAR data can be used for obstacle detection, mapping, and localization, which are essential for autonomous navigation. Gazebo is a popular simulation platform that is widely used for testing and evaluating autonomous navigation systems [16].

In a recent study, Jiang et al. proposed a deep learning-based obstacle avoidance algorithm for autonomous vehicles using ROS and Turtlebot3 [17]. The algorithm was tested in both simulation and real-world environments, demonstrating its effectiveness in avoiding obstacles

and navigating through complex environments.

Another study by Gao et al. focused on the integration of LiDAR and visual sensors for improved obstacle detection and avoidance in autonomous navigation [18]. The authors developed a multi-sensor fusion algorithm that was tested on a Turtlebot3 platform, showing improved performance compared to using a single sensor.

In conclusion, obstacle avoidance is a critical component of autonomous navigation, and the use of ROS and Turtlebot3 provides a robust platform for testing and evaluating autonomous navigation systems.

3 Methodology

3.1 Setting Up the ROS Environment

Setting up the ROS environment involves several steps to ensure that the ROS (Robot Operating System) framework is properly installed and configured on the Ubuntu 20.04 PC. Initially, we install ROS1 Noetic, the latest version, ensuring compatibility with our system. Once installed, we proceed to install the Turtlebot3 Packages, essential for interfacing with the Turtlebot3 robot platform. Additionally, configuring the network is crucial, where we add the IP address of the PC to the `.bashrc` file and define it as the ROS Master and Host, establishing a communication link between the PC and the Turtlebot3.

3.2 Setting Up the Turtlebot

Setting up the Turtlebot involves a series of meticulous steps to ensure that the robot is ready for operation. Firstly, we prepare the microSD card by utilizing a card reader. Then, we download and burn the TurtleBot3 SBC Image to the card, ensuring that we choose the correct image file based on our hardware and ROS version (Noetic). Once the image is burned, we resize the partition and configure the WiFi Network Setting, ensuring seamless communication between the Turtlebot and the PC.

3.2.1 Bring Up

The 'Bring Up' phase involves initializing the Turtlebot for operation. This includes making an SSH connection to the Turtlebot from the remote PC, granting access to its functionalities. We then execute the bringup command `roslaunch turtlebot3_bringup turtlebot3_robot.launch`, which calibrates the Turtlebot for various functions such as teleoperation and mapping/navigation.

3.2.2 Teleoperation

Teleoperation enables remote control of the Turtlebot using various controllers. By utilizing the teleop function, we can maneuver the Turtlebot using keyboard inputs. Executing the command

`roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch` initiates the teleoperation interface, allowing us to navigate the Turtlebot in the desired direction.

3.2.3 Topic Monitor

The Topic Monitor serves as a tool to calculate bandwidth and message rate passing through topics. By executing the `rqt` command, we can access the Topic Monitor and monitor the desired topics, providing valuable insights into the communication flow within the ROS ecosystem.

3.2.4 RVIZ

RVIZ, a ROS graphical interface, plays a crucial role in visualizing various aspects of the Turtlebot's operation. By leveraging plugins for different topics, RVIZ allows us to visualize mapping and navigation algorithms effectively. With RVIZ, we can simulate and display intricate details of the environment and the Turtlebot's interaction within it.

3.3 SLAM

SLAM (Simultaneous Localization and Mapping) is a fundamental technique for mapping an unknown environment while simultaneously estimating the robot's location within it. To initiate SLAM, we execute the command `roslaunch turtlebot3_slam turtlebot3_slam.launch`. As the Turtlebot traverses the environment, its LIDAR sensor captures data, enabling the generation of a two-dimensional Occupancy Grid Map (OGM). We can save the map using `rosrun map_server map_saver -f ~/map`, ensuring that the environment is accurately represented for subsequent navigation tasks.

3.4 Navigation

Navigation enables the Turtlebot to autonomously move from its current position to a designated goal pose on the map. We launch the navigation node using the command `roslaunch turtlebot3_navigation turtlebot3_navigation.launch map_file:=~/map.yaml`. This enables the Turtlebot to utilize the map generated through SLAM for navigation purposes, facilitating seamless movement within the environment.

4 Navigation and Algorithms

4.1 Dynamic Window Approach (DWA)

The Dynamic Window Approach (DWA) stands as a cornerstone in path planning within the robotics domain, esteemed for its adeptness in accommodating dynamic constraints like a robot's maximum velocity, acceleration, and turning radius. At its core, DWA revolves around the concept of a "dynamic window," delineating the range of viable velocity commands a robot can execute in the short term, given its current state and environmental conditions. This algorithm aims to guide the robot towards its goal while navigating around obstacles and adhering to its dynamic limitations.

The commencement of DWA lies in state estimation, where the algorithm gauges the robot's current state encompassing parameters such as position, orientation, and velocity. Leveraging this known state information, DWA then calculates the dynamic window, which outlines the permissible velocities the robot can adopt based on its maximum capabilities and the constraints posed by its surroundings. Within this dynamic window, DWA generates multiple potential trajectories or paths that the robot could follow. Each trajectory undergoes rigorous evaluation based on criteria like obstacle proximity, distance from the goal, and path smoothness.

An integral part of DWA is the evaluation of trajectories using a designated cost function. This function plays a pivotal role in assigning scores to trajectories, penalizing those that pose risks such as proximity to obstacles, deviation from the goal, or necessitating abrupt velocity changes. Through this evaluation process, DWA identifies the trajectory with the lowest computed cost, thereby determining the optimal velocity command for the robot to execute under the prevailing circumstances.

In essence, the Dynamic Window Approach combines state estimation, dynamic window calculation, trajectory generation, cost function evaluation, and optimal velocity command selection into a cohesive framework aimed at steering robots effectively amidst dynamic constraints and diverse environmental challenges encountered in real-world scenarios. Its significance in robotics research and applications underscores its practical utility and theoretical robustness in facilitating intelligent and adaptive robot navigation.

4.1.1 Local Planning with DWA

Local planning in robotics is a critical aspect of enabling autonomous navigation and effective interaction with dynamic environments. One prominent approach in this domain is the Dynamic Window Approach (DWA), which employs the dynamic window concept as a cornerstone for generating feasible velocity commands for a robot within a short time window. At its core, the dynamic window represents the range of permissible velocity commands that the robot can execute, factoring in its current state encompassing position, orientation, and velocity, as well as dynamic constraints like maximum velocity, acceleration, and turning radius. This dynamic window is continuously updated at each time step, ensuring that planned velocities align harmoniously with the robot's physical capabilities and safety constraints within its immediate surroundings.

A pivotal step within the DWA framework is the feasibility check, where candidate velocity commands are meticulously evaluated within the dynamic window. This evaluation encompasses considerations such as collision avoidance, smooth motion trajectories, and adherence to dynamic constraints. By rigorously assessing the viability of velocity commands, this phase ensures that only feasible commands are considered for trajectory generation.

The trajectory generation process within DWA is multifaceted, aiming to produce multiple potential trajectories within the dynamic window by varying linear and angular velocities. Each trajectory undergoes comprehensive evaluation based on a sophisticated cost function that takes into account various factors such as obstacle proximity, distance to the goal, motion smoothness, and path deviation. This intricate evaluation process facilitates the selection of trajectories that not only navigate the robot towards its goal but also prioritize safety, efficiency, and smooth motion.

An integral aspect of DWA is its robust obstacle avoidance mechanism. This mechanism involves predicting potential collisions along each trajectory and dynamically adjusting or selecting new trajectories from the dynamic window to ensure safe navigation and obstacle avoidance. By taking a reactive approach, DWA enables the robot to navigate around obstacles in real time, maintaining a balance between progress towards the goal and safe traversal through the environment.

Moreover, DWA exhibits a remarkable capability for reactive and adaptive planning, which is crucial for navigating in dynamic and unpredictable environments. By leveraging live sensor

data such as lidar and cameras, DWA continuously updates the dynamic window and recalculates feasible trajectories based on the evolving environment. This adaptive behavior allows the robot to respond effectively to changes such as moving obstacles or dynamic terrain, ensuring reliable navigation even in challenging scenarios.

4.1.2 Global Planning with DWA

Global planning using the Dynamic Window Approach (DWA) represents a significant advancement in robotic navigation systems, extending its capabilities beyond local path planning to encompass goal-directed motion and optimization. At the core of this approach is the meticulous initialization of the robot's goal position, setting the stage for a comprehensive global planning process. The initial step involves establishing a rudimentary path, typically a straight line or an initial guess, from the robot's current location to the designated goal. This preliminary path serves as a foundational framework for the subsequent phases of path refinement and optimization.

In the realm of global planning, DWA excels in its ability to iteratively refine trajectories through a dynamic window approach. This methodology involves generating and evaluating potential trajectories while considering dynamic constraints, obstacle avoidance strategies, and optimization criteria. The trajectory refinement process is pivotal, as it fine-tunes the initial path to navigate complex environments effectively.

Central to this refinement is the optimization criteria, which drive the planner's decisions towards minimizing travel distance, circumventing known obstacles, and ensuring seamless transitions between different path segments.

Through the integration of goal-directed motion and optimization within its global planning framework, DWA embodies a sophisticated approach to robotic navigation. By combining meticulous goal initialization, path refinement techniques, and optimization strategies, DWA facilitates efficient and intelligent decision-making for robots operating in diverse and challenging environments. This comprehensive approach not only enhances navigation accuracy but also contributes significantly to the overall performance and adaptability of robotic systems in real-world scenarios.

4.1.3 Local-Global Coordination

The coordination between local and global planning components within the Dynamic Window Approach (DWA) forms a synergistic framework, where each aspect complements the other to achieve optimal navigation outcomes. Local planning plays a crucial role in immediate obstacle avoidance and dynamic adjustments, ensuring that the robot can navigate through its environment efficiently and safely. On the other hand, global planning focuses on long-term path optimization and goal-directed guidance, providing the strategic framework necessary for the robot's journey. This seamless coordination between local and global planning facilitates smooth transitions between different phases of navigation, maintaining stability and performance throughout the robot's trajectory.

One of DWA's notable strengths lies in its real-time adaptation capabilities, particularly in handling dynamic environments. The system excels in responding to real-time changes, such as moving obstacles or evolving goals, ensuring that the robot can navigate effectively in scenarios requiring constant responsiveness. This adaptability is achieved through continuous updates and trajectory evaluations, allowing DWA to refine its navigation strategies iteratively. Over time, this iterative process leads to improved performance and adaptability as the system learns from its experiences, making it well-suited for dynamic and unpredictable environments.

In terms of robustness and safety, DWA adopts a comprehensive approach to risk mitigation. By integrating obstacle avoidance techniques, trajectory evaluation methods, and dynamic adjustments, the system enhances overall navigation safety. It minimizes collision risks and ensures smooth, controlled motion, thereby mitigating potential hazards during navigation. Additionally, DWA incorporates fail-safe mechanisms such as emergency braking, alternative path selection, or dynamic replanning to handle unexpected situations effectively. These fail-safe measures further contribute to maintaining safe navigation outcomes, even in challenging or unpredictable scenarios.

4.2 A* (A-Star) Algorithm

Implementing the A* (A-Star) algorithm as the global planner in autonomous navigation necessitates a fundamental shift in approach compared to the traditional use of the Dynamic Window Approach (DWA). A* is renowned for its efficacy in finding the shortest path between a

start and goal node within a graph or grid-based map. This algorithm iteratively explores nodes in the search space, prioritizing those with lower estimated costs to reach the goal. Key to its operation is the heuristic function, which estimates the cost from each node to the goal, guiding the search efficiently while considering the cost of reaching each node. For A* to guarantee optimal pathfinding, certain conditions, such as employing an admissible heuristic, must be met.

In terms of implementation, A* operates on a graph representation of the environment where nodes signify positions or waypoints, and edges represent potential transitions. This graph is typically derived from the environmental map, with nodes strategically placed to facilitate effective navigation. Choosing an appropriate heuristic function is crucial; it typically estimates distances or times from nodes to the goal, with options like Euclidean or Manhattan distances, or customized metrics based on specific requirements.

The A* algorithm is then integrated as the pathfinding engine within the global planner module. Starting from the robot's current position (the start node), it systematically explores neighboring nodes based on their estimated costs, maintaining open and closed sets of nodes and prioritizing those with lower total costs. This generated global path, from start to goal, serves as a high-level trajectory that must be integrated seamlessly with the local planner, such as the DWA local planner, to ensure real-time obstacle avoidance and smooth execution. The advantages of employing A* as the global planner are substantial. It guarantees optimal pathfinding, optimizing the entire global path while accounting for obstacles, terrain, and other factors, thereby creating a comprehensive navigation strategy. A* demonstrates robustness in handling complex environments, including obstacles, varied terrain, and multiple waypoints, offering reliability even in challenging scenarios. Its flexibility allows for heuristic function and graph representation customization, making it adaptable to diverse environments, navigation objectives, and robot capabilities.

However, there are several challenges and considerations to address. A* can be computationally demanding, particularly in extensive environments with intricate graphs, necessitating efficient data structures and heuristic optimizations for real-time performance. The accuracy of the heuristic function profoundly influences A*'s performance and optimality, necessitating careful selection and tuning. Integration with the local planner demands synchronization of trajectory planning, obstacle avoidance, and dynamic adjustments to ensure seamless navigation. Additionally, strategies for dynamic environment handling, such as dynamic replanning, are

crucial to accommodate changes in the environment, moving obstacles, or deviations from the planned global path.

In conclusion, replacing the DWA global planner with A* offers significant advantages in terms of optimal pathfinding, global optimization, and robustness in complex environments. However, successful implementation requires addressing challenges related to computational complexity, heuristic accuracy, integration with local planning, and dynamic environment handling to achieve efficient and reliable autonomous navigation, making it a compelling area for further research and development in autonomous systems.

4.3 Navigation using DWA and A* Algorithms

The Navigation enables a robot to move from the current pose to the designated goal pose on the map by using the map, robot's encoder, IMU sensor, and distance sensor [19]. By launching the navigation node, RViz, the visualization tool opens up and the map which was saved earlier is used for reference. Using the LiDAR sensor, the turtlebot creates a heatmap around it for a radius of 1-2 meters which can be changed in the parameters and this heatmap appears on top of the map saved and identifies any obstacles.

4.3.1 How Navigation works

First step to do after this would be setting the initial pose and direction of the robot with respect to the map saved for it to recognise where to start from (Figure 2) and next would be giving it a goal position (Figure 3) for reaching which the robot can create a path to traverse through.

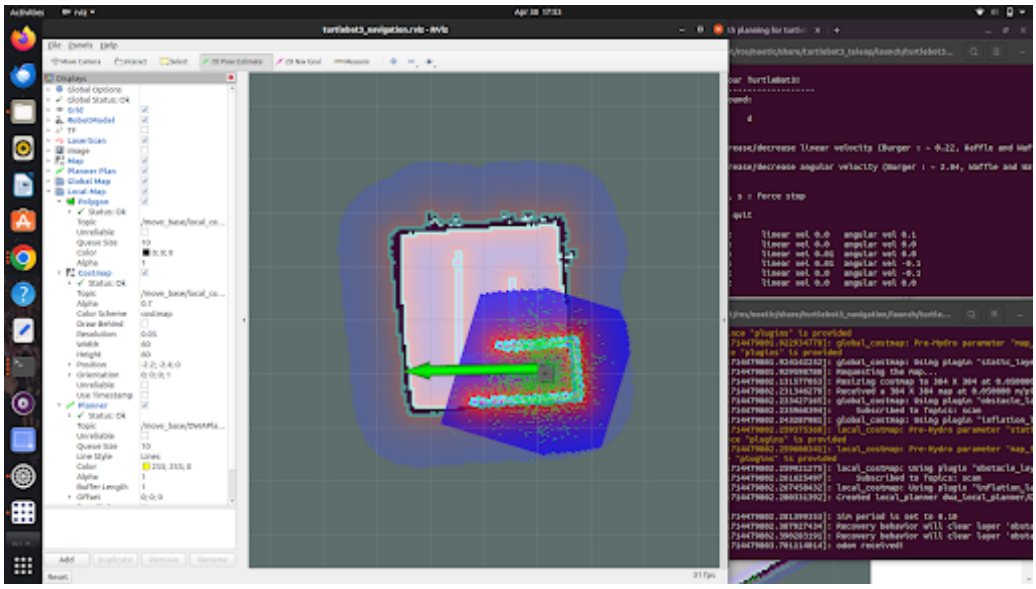

Figure 2. Setting the initial position/source of the bot

After setting the goal position, the robot will execute the path planning algorithm and create a path from it's current position to the goal position. There's a global path which is created by the global path planning algorithm based on just the initial and goal positions and there is a local path which is decided on the obstacles nearby if there's any.

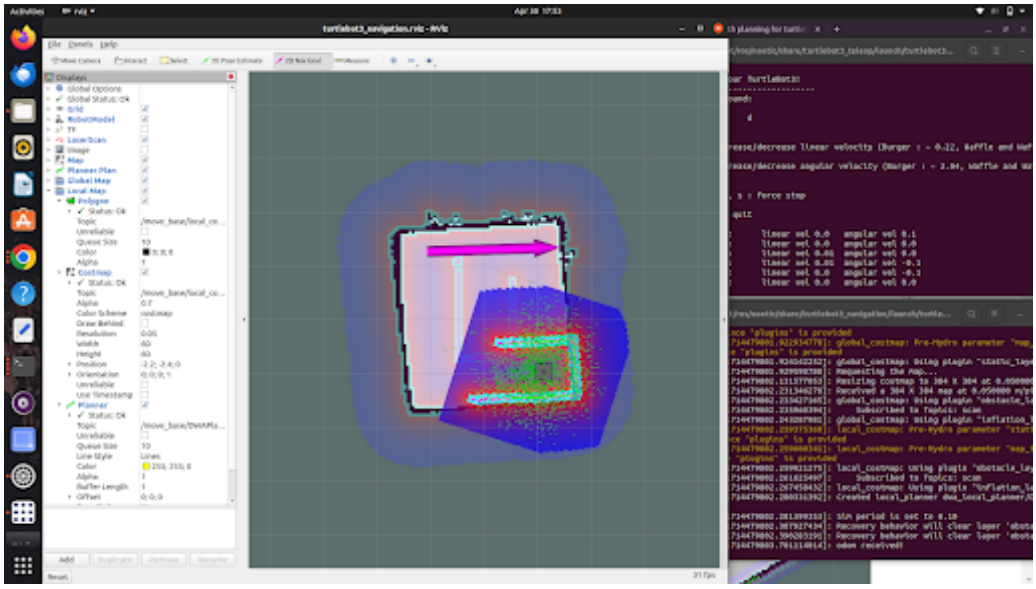

Figure 3. Setting the final position/destination of the bot

4.3.2 DWA Planner for Navigation

As explained in 4.1 about DWA Planner, the path is created by iteratively evaluating and selecting feasible velocity commands based on the robot's current state and sensor observations.

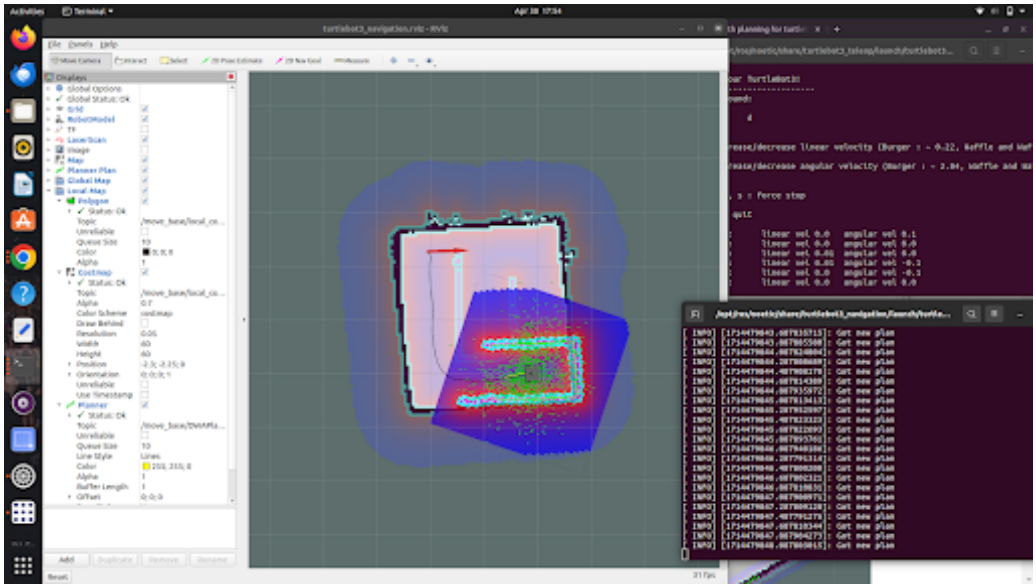

Figure 4. Path created by DWA Global and Local Algorithms

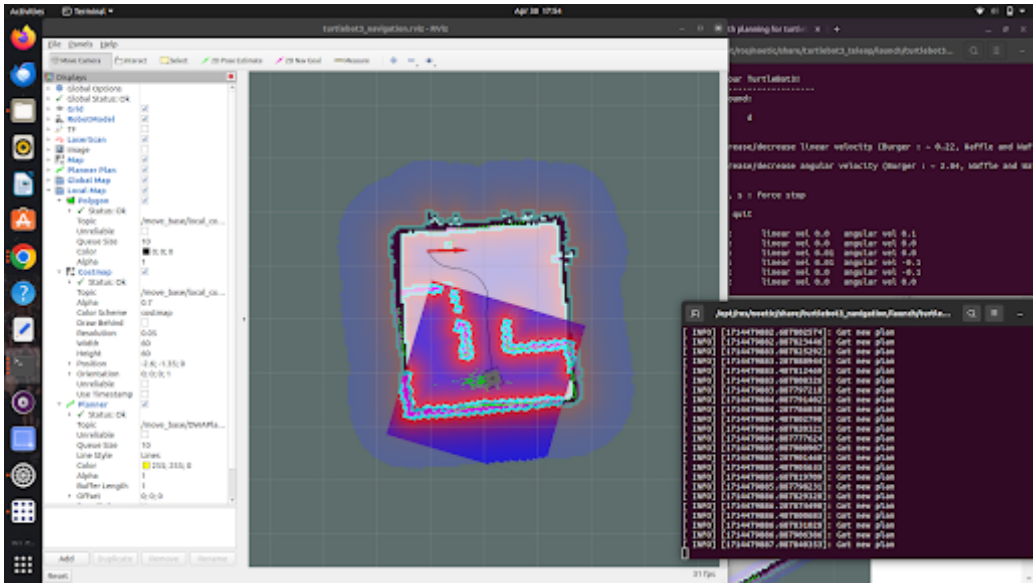

Figure 5. Updation of the path after obstacle detection

In the above figures, black line indicates the global path and the small yellow line near to the robot indicates the local path and when new obstacles are detected by traversing, the global path gets updated accordingly by avoiding the obstacles efficiently.

Also, within the ROS navigation stack, parameters are configured across multiple components, including the global and local planners, costmaps, sensor fusion algorithms, and trajectory controllers. For instance, parameters related to the global planner, such as the inflation radius of obstacles in the costmap and the resolution of the grid map, directly impact the generation of global trajectories and the robot's ability to plan paths that avoid collisions and optimize traversal efficiency [21].

Similarly, parameters within the local planner, such as the maximum and minimum velocities, acceleration limits, and obstacle avoidance thresholds, dictate the robot's reactive behavior in navigating through dynamic environments and avoiding obstacles encountered in real-time. Fine-tuning these parameters is crucial to achieving smooth, collision-free navigation while adhering to the robot's kinematic constraints and safety requirements [22].

4.3.3 A* algorithm for Navigation

The above descriptions were for DWA planners and we have implemented another algorithm which enhances the obstacle avoidance and creates paths better which is A* algorithm.

The A* algorithm, renowned for its efficiency in finding the shortest path between two points on a graph, presents a robust solution for navigating through complex environments while avoiding obstacles. In our case, the grids in the map are considered nodes and are traversed through by A* algorithm to find the goal position.

The path turns green when the goal is found and the robot reaches the goal through the created path. As we can see in the second terminal of Fig. 7, it shows the total execution time is around 14.5 seconds for traveling around 3.5 meters of distance.

By using the 2D navigation tool application given in the Gazebo platform, we can set the destination for the turtlebot to move without any collision. This 2D navigation tool also points the red color arrow mark as the destination is given and the bot will move to the destination. Also, in this the turtlebot moves in between the obstacles to reach the destination. The obstacles are visualized in the color of green and the boundaries of the environment are in black color as shown above [20].

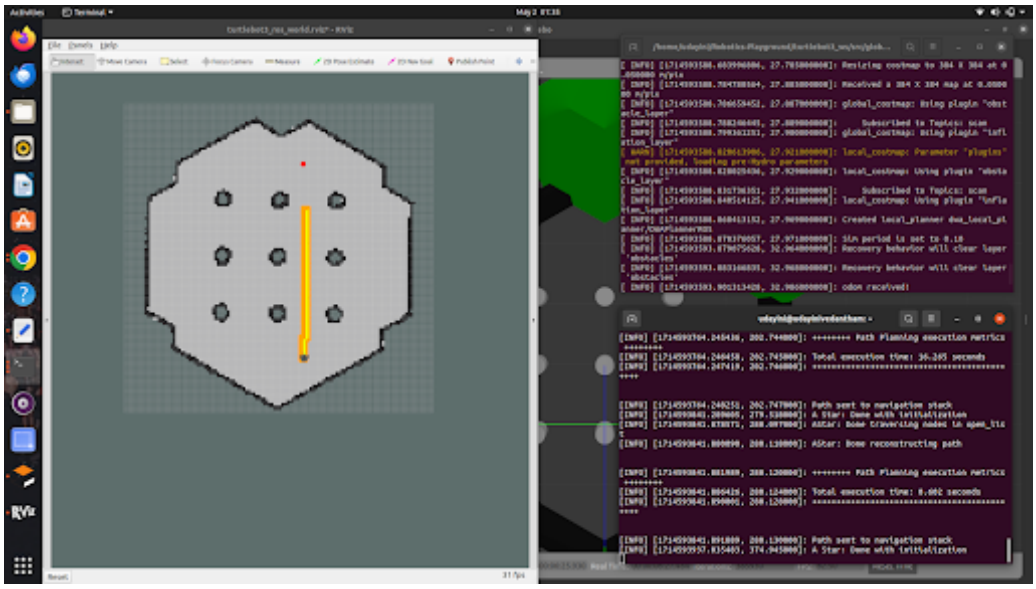

Figure 6. A* Algorithm finding a path from source to destination

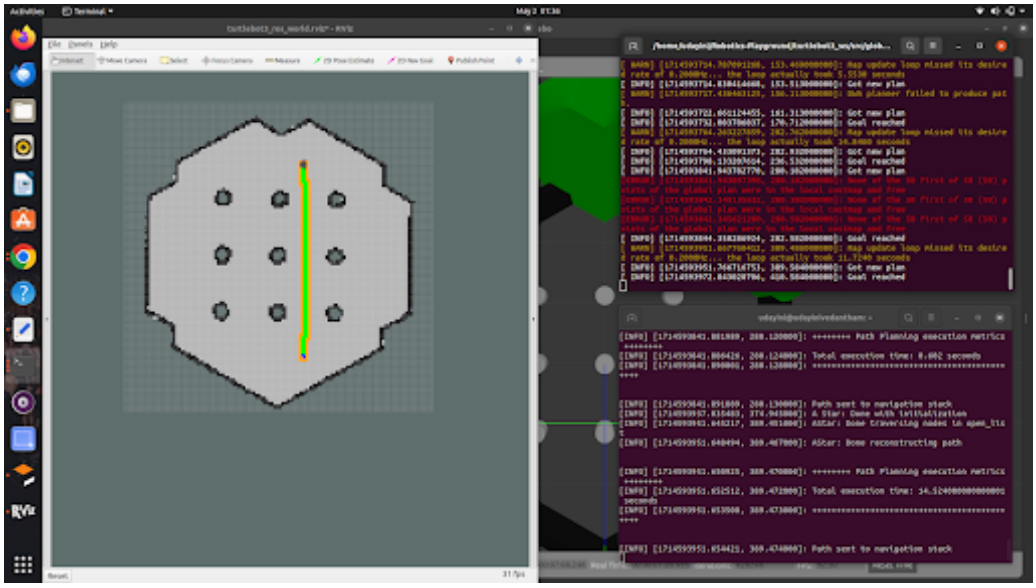

Figure 7. Goal reached

5 Conclusion

One notable advantage of utilizing the A* algorithm lies in its ability to systematically explore the search space, prioritizing paths that are less likely to encounter obstacles. By leveraging a heuristic function to estimate the cost of reaching the goal from each explored node, A* intelligently guides the robot towards its destination while dynamically adjusting its trajectory to avoid potential obstacles encountered along the way.

In contrast, DWA planners primarily focus on local navigation by iteratively sampling and evaluating velocity commands based on the robot's kinematic constraints and sensor readings. While DWA planners are effective in navigating dynamic environments in real-time, they may struggle to handle intricate obstacle configurations or plan globally optimized paths.

In the experiments conducted within the Gazebo simulation environment, the comparative analysis between A* and DWA planners showcases the distinct advantages of A* in navigating through cluttered environments with narrow passages, dynamic obstacles, and complex structures. Through quantitative metrics such as path length, traversal time, and collision avoidance rates, we can use the A* algorithm in achieving safe and efficient navigation trajectories.

6 References

1. Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., ... & Ng, A. Y. (2009). ROS: an open-source Robot Operating System. In ICRA workshop on open source software (Vol. 3, No. 3.2, p. 5).

2. S. Gobhinath, K. Anandapoorani, K. Anitha, D. D. Sri, and R. DivyaDharshini, "Simultaneous Localization and Mapping [SLAM] of Robotic Operating System for Mobile Robots," in 2021 7th International Conference on Advanced Computing and Communication Systems (ICACCS), IEEE, Mar. 2021, pp. 577–580, doi: 10.1109/ICACCS51430.2021.9441758.

3. K. Li and H. Tu, "Design and Implementation of Autonomous Mobility Algorithm for Home Service Robot Based on Turtlebot," in 2021 IEEE 5th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC), IEEE, Oct. 2021, pp. 1095–1099, doi: 10.1109/ITNEC52019.2021.9587138.

4. Heo, J., Choi, J., & Kim, J. (2019). Turtlebot3: A Low-Cost, Open-Source Robot for Learning and Research. IEEE Robotics & Automation Magazine, 26(3), 103-112.

5. Dijkstra, E.W. A note on two problems in connexion with graphs. In Edsger Wybe Dijkstra: His Life, Work, and Legacy; ACM: New York, NY, USA, 2022; pp. 287–290.

6. Hart, P.E.; Nilsson, N.J.; Raphael, B. A formal basis for the heuristic determination of minimum cost paths. IEEE Trans. Syst. Sci. Cybern. 1968, 4, 100–107.

7. Dorigo, M.; Gambardella, L.M. Ant colony system: A cooperative learning approach to the traveling salesman problem. IEEE Trans. Evol. Comput. 1997, 1, 53–66.

8. Kennedy, J.; Eberhart, R. Particle swarm optimization. In Proceedings of the ICNN'95-International Conference on Neural Networks, Perth, WA, Australia, 27 November–1 December 1995; Volume 4, pp. 1942–1948.

9. LaValle, S.M.; Kuffner, J.J., Jr. Randomized kinodynamic planning. Int. J. Robot. Res. 2001, 20, 378–400.

10. Ajeil, F.H.; Ibraheem, I.K.; Sahib, M.A.; Humaidi, A.J. Multi-objective path planning of an autonomous mobile robot using hybrid PSO-MFB optimization algorithm. Appl. Soft Comput. 2020, 89, 106076.

11. WU Feilong, GUO Shiyong. Fusion of improved A* and dynamic window method for AGV dynamic path planning[J]. Science, Technology and Engineering, 2020,20(30):12452-12459

12. WU Yi, OU Mingmin, DUAN Liwei. Research on robot path planning based on improved A* algorithm and dynamic window method[J]. Industrial Control Computer, 2020,33(10) 67-70.

13. CHENG Legend, HAO Xiangyang, LI Jiansheng, et al. Global dynamic path planning by integrating improved A* algorithm and dynamic window method[J]. Journal of Xi'an Jiaotong University, 2017,51(11):137-143.

14. Y. X. Wang, Y. Y. Tian, X. Li, et al. Adaptive DWA algorithm for traversing dense obstacles[J]. Control and Decision Making, 2019, 34(5): 927-936.

15. Himmelsbach, M., M¨uller, A., L¨uttel, T., & Hecker, F. (2010). LIDAR-based 3D Object Perception. In Proceedings of 1st International Workshop on Cognition for Technical Systems.

16. Koenig, N., & Howard, A. (2004). Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator. In Proceedings of the 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2004) (Vol. 3, pp. 2149-2154).

17. Jiang, Z., Zhao, Y., Gao, Y., & Jiang, G. (2022). Deep Learning-Based Obstacle Avoidance for Autonomous Navigation Using ROS and Turtlebot3. Sensors, 22(4), 1573.

18. Gao, Y., Zhao, Y., Jiang, Z., & Jiang, G. (2021). Obstacle Avoidance for Autonomous Navigation Using Multi-Sensor Fusion. Sensors, 21(6), 2138.

19. https://emanual.robotis.com/docs/en/platform/turtlebot3/navigation/run-navigation-nodes

20. https://ijisrt.com/assets/upload/files/IJISRT22JUN067(1).pdf

21. Zheng, K. (2021). ROS Navigation Tuning Guide. In: Koubaa, A. (eds) Robot Operating System (ROS). Studies in Computational Intelligence, vol 962. Springer, Cham. https://doi.org/10.1007/978-3-030-75472-36

22. B. Cybulski, A. Wegierska and G. Granosik, "Accuracy comparison of navigation local planners on ROS-based mobile robot," 2019 12th International Workshop on Robot Motion and Control (RoMoCo), Poznan, Poland, 2019, pp. 104-111