Design and Development of a Path Planning Scheme for Two-wheeled Mobile Robots with Obstacle Avoidance Properties

A Design Lab Project Report Submitted
in Partial Fulfillment of the Requirements
for the Degree of

Bachelor of Technology

by

Raveena Kumari and Abhishek Gupta
(210108039 and 210108065)

under the guidance of

Dr. Parijat Bhowmick

to the

DEPARTMENT OF ELECTRONICS AND ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI
GUWAHATI - 781039, ASSAM

CERTIFICATE

*This is to certify that the work contained in this report entitled "**Design and Development of a Path Planning Scheme for Two-wheeled Mobile Robots with Obstacle Avoidance Properties**" is a bonafide work of **Raveena Kumari and Abhishek Gupta** (**Roll No. 210108039 and 210108065**), carried out in the Department of Electronics and Electrical Engineering, Indian Institute of Technology Guwahati under my supervision and that it has not been submitted elsewhere for a degree.*

Supervisor: **Dr. Parijat Bhowmick**

Assistant Professor,

April, 2024

Dept of Electronics & Electrical Engineering,

Guwahati.

Indian Institute of Technology Guwahati, Assam.

Contents

Chapter 1

Introduction

1.1 Background of the topic

Two-wheeled mobile robots, often referred to as differential drive robots, are characterized by their simple yet effective locomotion mechanism consisting of two powered wheels with independent motors. This configuration allows for efficient maneuverability and versatility in navigating diverse terrains. However, the challenge lies in devising algorithms and control strategies that enable these robots to navigate autonomously in dynamic environments without colliding with obstacles.

The autonomous design of a safe, collision-free path that takes the least amount of time and travels from a beginning point to an end point is known as mobile robot path planning. This work presents a thorough review of strategies for path planning of mobile robots. First, based on the proficiency with environmental data, path planning is divided into two categories: global path planning and local path planning. Path evaluation and environment modelling techniques are incorporated in the global path planning. The grid approach, topology method, geometric feature method, and mixed representation method are some of the techniques used in environment modelling. We introduce the sensors typically employed in the detection environment, such as laser radar and visual sensor, in the local path planning.

1.2 State of the Art

Based on the type of environment the robot operates in, mobile robot path planning can be broadly divided into two classes. The path-planning strategy and methods utilised to

successfully guide the robot are heavily influenced by these classifications. This is shown in Figure 1:

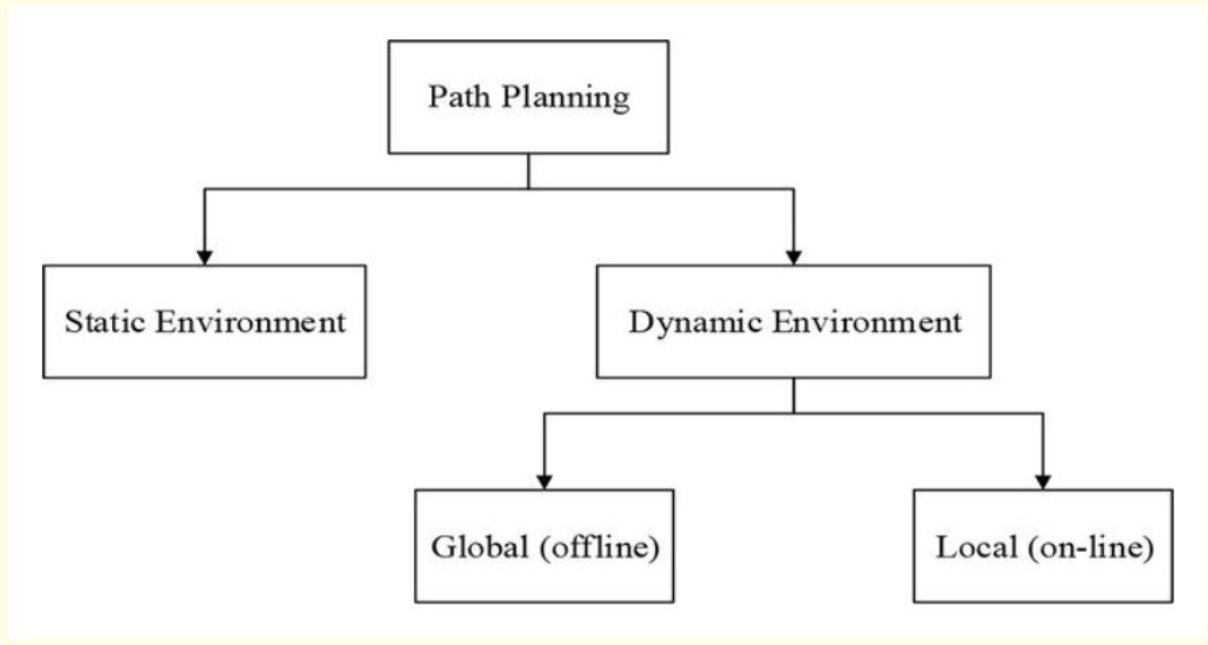

Fig. 1.1 Classification of mobile robot path planning

Robot route planning in a static environment: this type of path planning only includes stationary obstacles in the surrounding area;

Path planning in a dynamic environment: this type of path planning takes into account both stationary and moving environmental impediments.

Global path planning and local path planning are two further subcategories of the dynamic path planning.Global route planning, which is often referred to as off-line path planning, is the procedure wherein the robot knows its whole surroundings, including the locations of obstacles, before it begins to move.

However, online or local path planning deals with scenarios in which the robot does not know the surroundings or the locations of obstacles before it begins to move. The mobile robot in this scenario uses its sensors to continuously sense its surroundings. It creates a map that faithfully depicts the environment's structure using the sensor data.

To tackle this difficult issue, researchers have looked into a range of methods in the literature for mobile robot path planning. These methods can be roughly divided into four groups: hybrid-based methods, heuristics, metaheuristics, and classical (traditional) methods[1].

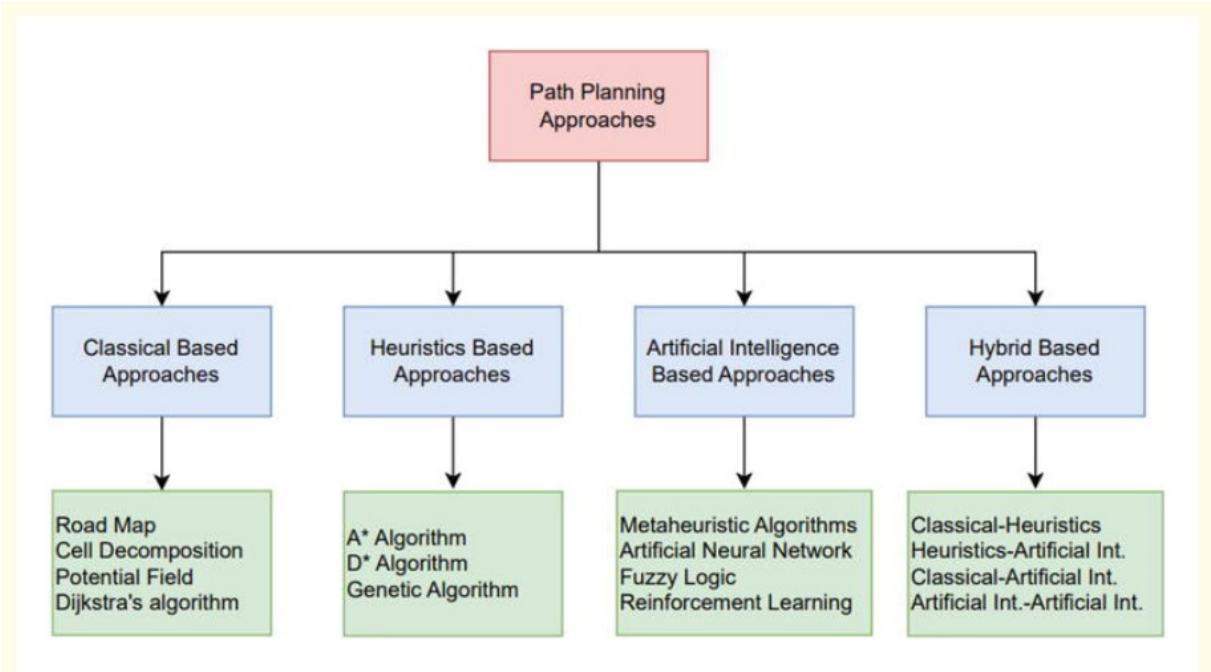

Fig. 1.2 Path-planning approaches.

1.2.1 Classical (conventional) approaches:

These approaches usually entail the use of algorithms that are derived from conventional ways of problem-solving. Road map, cell breakdown, potential field, and sub-goal network techniques are a few examples. These techniques can successfully locate the best or almost best courses for mobile robots in simple contexts with relatively small search regions[2].

1.2.2 Heuristic approaches:

Heuristic methods make use of domain-specific information or guidelines to effectively direct the search for a workable solution. They are appropriate for real-time applications since they forgo optimality and completeness in favour of computational speed. Dijkstra's algorithm, A* (A-star), D* (D-star), and the genetic algorithm are a few examples.

1.2.3 Artificial Intelligence approaches:

The drawbacks of the conventional methods include trapping in local minima, limited applicability to 2D, and high computation costs [8, 10, 38]. Furthermore, path planning is a hard task with non-deterministic polynomial time. For this reason, metaheuristic search strategies are superior approaches to solve this kind of issue.

1.2.4 Hybrid based Approaches:

Hybrid-based approaches for path planning of two-wheeled mobile robots combine elements of both discrete and continuous methods to generate feasible and efficient trajectories in dynamic environments. These approaches leverage the strengths of discrete algorithms, such as graph search techniques, and continuous methods, such as potential field-based or sampling-based algorithms, to overcome the limitations of each approach individually[2].

1.3 Motivation

The motivation behind our project stems from the increasing demand for autonomous navigation systems in various applications, ranging from warehouse logistics to urban transportation. Two-wheeled mobile robots offer unique advantages in terms of maneuverability, compactness, and energy efficiency, making them ideal candidates for tasks such as delivery, surveillance, and exploration.

Our project seeks to leverage state-of-the-art algorithms and sensor technologies to enable robots to navigate complex environments autonomously while ensuring safety, efficiency, and adaptability.Ultimately, our goal is to contribute to the advancement of autonomous robotics technology, paving the way for the seamless integration of two-wheeled mobile robots into various domains, including smart cities, industrial automation, and beyond

1.4 Objectives

Our project aims to develop a tailored path planning algorithm for two-wheeled mobile robots, focusing on optimizing navigation efficiency and obstacle avoidance capabilities. Specifically, we aim to integrate sophisticated obstacle detection mechanisms using sensor data from LiDAR, cameras, and IMUs to generate dynamic obstacle maps and plan collision-free trajectories. Additionally, we seek to enhance the algorithm's adaptability to dynamic environments by implementing real-time decision-making and optimization techniques to generate trajectories that minimize path length, conserve energy, and ensure smooth motion. Through extensive simulation studies and real-world experiments, we aim to validate the algorithm's effectiveness, scalability, and computational efficiency, contributing to the advancement of autonomous robotics technology and facilitating its practical deployment in various applications [3].

Chapter 2

Problem Formulation

2.1 Problem Statement

The title of project is "**Design and Development of a Path Planning Scheme for Two-wheeled Mobile Robots with Obstacle Avoidance Properties**." In this project, our primary goal is to design and implement a path planning scheme that enables the robot to navigate autonomously while avoiding obstacles in the path. There are two parts. The first part is to optimize the existing algorithms of path planning scheme and implement it on the physical turtlebot. Secondly, incorporate obstacle avoidance properties that can come while moving in the trajectory to reach the goal point.

This directly implies two behaviors that mobile robot should possess:

1. Go To Goal: Traversing the through the shortest route between the starting point and goal point.
2. Obstacle Avoidance: Avoiding obstacles while traversing towards the goal point and updating the trajectory accordingly.

2.2 Modelling

Prior to researching the APF algorithm, we studied path planners such as Dijkstra's and A-star algorithms. Since these two algorithms are the most basic navigation planners, we are experimenting with them. As a result, we evaluated them in various goal and obstacle locations and made comments regarding their effectiveness.

2.2.1 Kinematic Model Used:

The most fundamental study of the behaviour of mechanical systems is kinematics. In order to construct mobile robots that are suitable for a given task and to create control software for a mobile robot hardware instance, it is necessary to comprehend the mechanical behaviour of the robot in mobile robotics.

Unicycle Model : The unicycle model simplifies the motion of a robot to that of a unicycle, where the robot can move in two dimensions: forward/backward motion along its longitudinal axis and rotation about a vertical axis (yaw). This model is useful for describing the motion of differential-drive robots, which have two independently controllable wheels.

The state of the robot in the unicycle model is typically represented by its position (x, y) and its orientation (theta). The robot's velocity is represented by linear velocity (v) in the direction it's facing and angular velocity (omega) representing its rotational speed.
The kinematics of the unicycle model are described by simple differential equations:

$$\dot{x} = v \cdot \cos(\theta) \tag{2.1}$$

$$\dot{y} = v \cdot \sin(\theta) \tag{2.2}$$

$$\dot{\theta} = \omega \tag{2.3}$$

where:

$$\begin{aligned} &\dot{x} \text{ is the rate of change of the robot's position in the } x\text{-direction,} \\ &\dot{y} \text{ is the rate of change of the robot's position in the } y\text{-direction,} \\ &\dot{\theta} \text{ is the rate of change of the robot's orientation,} \\ &v \text{ is the linear velocity of the robot,} \\ &\omega \text{ is the angular velocity of the robot.} \end{aligned} \tag{2.4}$$

2.2.2 Dijkstra algorithm:

A key graph theory algorithm called Dijkstra's algorithm is used to determine the shortest path between nodes in a weighted graph. It ensures that a source node will always have the shortest path to every other node in the graph. A robot's floor space can be divided into multiple blocks as it moves in a known-sized environment. Every block in this space can be represented by a pair of coordinates by giving one of its corners a coordinate axis.Depending on whether a block has an impediment or is empty, it can be given a binary value. The initial cell is designated as 0. Next, we use the corresponding edge weight to assign a value to each neighbouring node. We also keep track of a matrix containing the parent cells of

every node as we proceed. We can retrace the path using the least total of the edge weights from the starting node once we have reached the destination. While exploring every cell in the workspace is not necessary in this case, we still need to investigate every cell that is close by in order to get to the objective [3].

```
for each node n in the graph
    n.distance = infinity

create empty list

start.distance = 0, add start to list

while list not empty
    current = node in the list with the smallest distance, remove current
from list
    for each node n that is adjacent to current
        if n.distance > current.distance + length  of edge from n to
current
          n.distance = current.distance + length of edge from n to current
          n.parent = current
           add n to back of the list if there isn't already
```

Fig. 2.1 Pseudocode for Dijkstra's Algorithm

2.2.3 A*(star) algorithm:

Dijkstra's Algorithm is modified and optimised for a single destination by A*. While Dijkstra's Algorithm is effective at determining the shortest path, it wastes time looking in unproductive paths. Greedy Best First Search looks at interesting places, but it might not always find the easiest route. The A* algorithm creates pathways to a single site by utilising both the estimated distance to the target and the actual distance from the start. Paths that appear to be getting closer to the objective are given priority [**?**].
The best-first search algorithm finds the path that results in the least amount of cost—that is, the least amount of distance travelled or the shortest amount of time needed to reach the desired location—by looking over all of the potential routes to the destination. In terms of weighted graphs, A* is defined as follows: it begins at a particular node in the graph and builds a tree of pathways from it, expanding the paths one at a time until one of the paths finishes at the goal node or goal position. A* uses this method to expand paths that are already less expensive:

$$f(n) = g(n) + h(n) \tag{2.5}$$

where,

- n = the last node on the path
- f(n) = total estimated cost of path through node n
- g(n) = cost so far to reach node n
- h(n) = estimated cost from n to goal. This is the heuristic part of the cost function.

The heuristic used in the algorithm is tailored to the specific problem at hand. To ensure that the algorithm accurately finds the shortest path, the heuristic function must be admissible, meaning it should never overestimate the true cost of reaching the nearest goal node. The heuristic function can be calculated in various ways:

- **Manhattan Distance:** In this approach, the heuristic function $h(n)$ is determined by tallying the cumulative count of horizontal and vertical movements required to reach the target square from the current one. This method disregards any obstacles encountered along the path and excludes diagonal movements from consideration.

$$h = |x_{start} - x_{destination}| + |y_{start} - y_{destination}| \tag{2.6}$$

- **Euclidean Distance Heuristic:**The Euclidean heuristic offers slightly greater accuracy compared to its Manhattan counterpart. When both are applied simultaneously to the same maze, the Euclidean path finder tends to favor paths along straight lines, reflecting a more precise estimation of distance. However, this accuracy comes at the cost of increased computational time, as the Euclidean method necessitates exploration of a larger area to identify the optimal path.

$$h = \sqrt{(x_{start} - x_{destination})^2 + (y_{start} - y_{destination})^2} \tag{2.7}$$

- If h(n) = 0, A* becomes Dijkstra's algorithm

The algorithms mentioned operate on the premise of precise sensing and accurate state prediction. However, real-world sensor data often contains inherent noise, making it impossible to obtain a perfect understanding of the robot's location and its surroundings. Hence, the use of probabilistic models such as the Kalman Filter becomes essential for reasonably estimating system states [3].

2.2.4 Artificial Field Potential(apf) algorithm:

Artificial Potential Fields (APF) is a popular technique in robotics and path planning, designed to navigate autonomous agents or robots through complex environments while avoiding obstacles and reaching target destinations. APF is inspired by the concept of potential fields in physics, where objects exert forces on each other based on their positions and properties. Similarly, in APF, obstacles generate repulsive forces, while goals generate attractive forces, guiding the agent towards its target.

At the heart of APF is the notion of potential functions. Each point in the environment is assigned a scalar value representing its potential, which influences the motion of the agent. Typically, obstacles are associated with high potential values, creating repulsive fields around them. As the agent approaches an obstacle, the repulsive force increases, pushing the agent away and preventing collisions. Conversely, the goal location is assigned a low potential value, creating an attractive field that draws the agent towards it.

Equation (1) shows the calculation for the gravitational field generated by the target point on the vehicle.

$$U_{att}(q) = \frac{1}{2} k_{att} \rho^2(q, q_g)$$

Fig. 2.2

where $U_{\text{att}}(q)$ is the gravitational field, k_{att} is the gain of the gravitational field, and $\rho(q, q_g)$ represents the Euclidean distance between the car and the target. The magnitude of gravity is the negative derivative of the gravitational field on the distance between the vehicle and the target point.

The magnitude of gravity is the negative derivative of the gravitational field on the distance between the vehicle and the target point.

$$F_{att}(q) = -\nabla U_{att}(q) = -k_{att} \rho(q, q_g) = -k_{att}(q - q_g)$$

Fig. 2.3

The repulsive field function can be calculated using Equation (3).

where $U_{\text{rep}}(q)$ is the size of the repulsive field, k_{rep} is the gain in the repulsive field, $\rho(q, q_g)$ represents the Euclidean distance between the car and the obstacle, and ρ_o is the range of repulsion influence of the obstacle.

$$U_{rep}(q) = \begin{cases} 0, & \rho(q, q_g) > \rho_0 \\ \frac{1}{2} k_{rep} [\frac{1}{\rho(q,q_0)} - \frac{1}{\rho_0}], & 0 \le \rho(q, q_g) < \rho_0 \end{cases}$$

Fig. 2.4

The magnitude of repulsion is the negative derivative of the repulsion field on the distance between vehicles and obstacles.

$$F_{rep}(q) = -\nabla U_{rep}(q) = \begin{cases} 0, & \rho(q, q_g) > \rho_0 \\ k_{rep} [\frac{1}{\rho(q,q_0)} - \frac{1}{\rho_0}] \times \frac{1}{\rho^2(q,q_0)}, & 0 \le \rho(q, q_g) < \rho_0 \end{cases}$$

Fig. 2.5

Based on the above analysis, in the traditional APF algorithm, the total potential field and resultant force on the intelligent vehicle are shown in Equations (5) and (6), respectively [4].

$$U_{total} = U_{att}(q) + \sum_{i=1}^{n} U_{rep}(q)$$

$$F_{total} = F_{att}(q) + \sum_{i=1}^{n} F_{rep}(q)$$

Fig. 2.6

2.3 Analysis

A* and Dijkstra's algorithm are frequently employed in the process of planning paths for mobile robots with two wheels. Node and arc information are handled using node-based optimum algorithms. Nevertheless, this type of approach can only produce the optimal outcome constrained by the environment's representation, since the nodes and arcs only offer a partial structure of the configuration space. These algorithms are one-way search techniques. It is unable to provide multipath for the fleet. The temporal complexity of Dijkstra's algorithm

for real-time implementation is ($O(n^2)$), where n is the number of nodes. A* lowers the complexity to allow for on-line implementation [5].

Dijkstra's algorithm is praised for its simplicity and versatility, making it easy to implement across various environments. However, its drawback lies in its high time complexity, which can become a limiting factor in larger or more complex scenarios. It primarily addresses static threats, but its efficiency diminishes when faced with dynamic environments or changing conditions. On the other hand, an alternative method labeled as A* exhibits a fast searching ability despite carrying a heavy time burden. This characteristic enables its online implementation, making it suitable for real-time applications. Like Dijkstra's algorithm, it also deals with static threats and grapples with the issue of nonsmoothness, which can affect the optimality of the generated paths [2].

Over the past ten years, a lot of research has been done on the potential field approach for autonomous mobile robot path planning. The potential field method's core idea is to create an artificial potential field in the robot's workspace, which attracts the robot to its target position and repels it from impediments. This method's rapid convergence, elegant mathematics, and simplicity make it very appealing. It does, however, have certain intrinsic restrictions. In, a methodical critique of the underlying issues supported by mathematical analysis was provided. It contains the following points:

- oscillations in restricted routes
- oscillations in the presence of obstacles
- no passage between closely spaced obstacles
- trap situations resulting from local minima [6]

Overall, the choice between Dijkstra's algorithm and method A* depends on the specific requirements and constraints of the given scenario, balancing factors such as time complexity, searching ability, and adaptability to dynamic environments[5].

Chapter 3

Controller Design

In this chapter, the attention turns to control, specifically the creation of a controller enabling a car-like robot, to follow a predefined path, which can be adjusted dynamically. The path is defined by a series of (x, y) coordinates. The focus is solely on controlling the vehicle's steering, while other aspects of its motion remain unaffected. Any autonomous land vehicle necessitates a robust, dependable, and precise path-following system to ensure the successful attainment of motion objectives set by path planning and obstacle avoidance modules. To fulfill this role, a diverse array of control strategies can be employed, spanning from intricately detailed approaches encompassing all aspects of vehicle performance[7].

3.1 Usign PID

To enable a vehicle to follow a continuous curvature path accurately a control technique must be developed that allows the control of the three variables that comprise vehicle posture:

- Position, (x, y) - the vehicle must be on the path
- Orientation, θ - the vehicle must be facing in the correct direction
- Curvature, κ - the vehicle's steering angle must match the curvature of the path

For the control system in this project, the desired posture must be attained by controlling only the vehicle's steering angle ϕ [7]. These are two key features common to all such techniques, as listed below:

- The path is specified as a series of (x,y) points joined by line segments. This is effectively the same as the path specification output by the path planning module.
- A'look ahead distance' is defined. The controller makes control decisions based upon the vehicle's current configuration relative to some goal point further along the path

The steering controller utilizes a PID feedback control law, which operates on an error-based strategy. In this approach, the control action is determined by observing the behavior of the error, defined as the disparity between the steering setpoint and the measured steering angle. This control method can be expressed mathematically using the following equation[8].

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt} \tag{3.1}$$

where $u(t)$ is the output (control) signal, e(t) is the error term, and K_p, K_i and K_d are the proportional, integral and derivative gains, respectively. Such a controller is tuned by varying these three PID gains.

However, to implement such a control law in terms of discrete time intervals, the 'velocity'(or 'difference') form of the PID equation is required

$$u[k] = K_p e[k] + K_i \sum_{i=0}^{k} e[i] + K_d(e[k] - e[k-1]) \tag{3.2}$$

3.2 Using State Feedback

State-space techniques are central to contemporary control theory, operating under the premise that even intricate nonlinear systems exhibit linear behavior in the vicinity of the point of linearization. After linearizing the system to establish a state-space representation, a variety of controllers can be designed using multivariable linear control theory.

As an illustration, employing a state-space representation of a vehicle's kinematics enables the utilization of feedforward or model-based predictive controllers. These controllers leverage accurate predictions of the vehicle's future states derived from its current configuration and established dynamic behavior. Consequently, control decisions can be made based on anticipated future states of the vehicle, reducing reliance solely on error-based techniques typical in feedback control and allowing actions to be initiated proactively [7].

Chapter 4

Simulation Case Studies

We turn our attention to comprehending the ROS environment needed to run Gazebo in order to grasp the practical aspect of this project. This required configuring Rviz, Turtlebot, and Gazebo in ROS. Additionally, we learnt how to establish nodes, publish or subscribe to topics, and compile Python executables into packages. With the use of this information, we were able to construct a world in Gazebo for Turtlebot to navigate and teleoperate it to produce maps. We were able to complete a thorough analysis of these techniques and re-implement the A-star algorithm in Python thanks to this map[3].

4.1 Problem Description

As the mentioned title of the project, we have aimed to simulate a similar objective of path planning with obstacle avoidance properties for turtlebot in the virtual environment of Gazebo with the help of ROS.

But results of the code on the simulation and on a physical bot is completely different. Other factors need to be considered while running the code on the physical turtlebot.

4.2 Control Objectives

The control objectives of path planning algorithms implemented for TurtleBot in Gazebo encompass several key aims. Firstly, the algorithms strive to efficiently navigate the TurtleBot through its environment, avoiding obstacles while adhering to specified path constraints. Additionally, the control strategies aim to optimize the TurtleBot's trajectory, ensuring smooth and stable motion throughout its operation. Moreover, these algorithms prioritize robustness, aiming to maintain consistent performance in the face of uncertainties such as sensor noise or environmental variations. Overall, the control objectives of path planning algorithms for TurtleBot on Gazebo revolve around achieving reliable, agile, and obstacle-free navigation in simulated environments.

Holonomic refers to the correspondence between the controllable and total degrees of freedom within a robot's motion system. When the controllable degrees of freedom match the total degrees of freedom, the robot is classified as holonomic. An example of this is a robot equipped with castor wheels or omniwheels, where it can move freely in any direction. In such cases, the controllable degrees of freedom align precisely with the total degrees of freedom, enabling versatile and unrestricted movement. [1].

Designing a controller for path planning while considering the holonomic drive of the TurtleBot involves several key considerations. Firstly, the controller should take advantage of the TurtleBot's omni-directional movement capabilities, allowing it to navigate through complex environments efficiently. This entails developing control algorithms that can generate trajectories incorporating lateral movements, rotations, and translations seamlessly.

Secondly, the controller should prioritize real-time responsiveness to dynamic changes in the environment. This requires implementing algorithms that can quickly adapt the TurtleBot's path in response to new obstacles or changes in the desired trajectory.

Additionally, the controller should aim for smooth and stable motion to ensure the TurtleBot's movements are precise and predictable. This involves designing control strategies that minimize jerky or abrupt motions, particularly during direction changes or maneuvers.

Furthermore, the controller should integrate obstacle avoidance mechanisms to ensure the TurtleBot can navigate safely through cluttered environments. This may involve incorporating sensors and perception algorithms to detect obstacles and dynamically adjust the planned path accordingly.

Overall, the design of the controller for path planning with a holonomic drive TurtleBot should focus on leveraging its omni-directional capabilities, ensuring real-time responsiveness, achieving smooth and stable motion, and integrating obstacle avoidance mechanisms for safe navigation [9].

4.3 Simulation Results

We implemented the Dijkstra's algorithm as well as A* algorithm. for simulation purpose we used gazebo and rviz software's. We are using **TurtleBot3 Simulation Package** to run the simulation. We are also using **ROS Navigation Stack**.

Dijkstra's Algorithm : Dijkstra's algorithm is used to determine the shortest path between nodes in a weighted graph. It is named for the Dutch computer scientist Edsger W. Dijkstra because, if the graph has no negative edge weights, it ensures the shortest path from a source node to every other node in the graph.

Here we are attaching 3 pictures as results. In the first and second picture as we can see

the algorithm is searching for all the nodes between the start point and goal point and then it finds the shortest path between the two points. In the 3rd picture we can see that the robot is moving towards the goal position along the trajectory generated by algorithm.

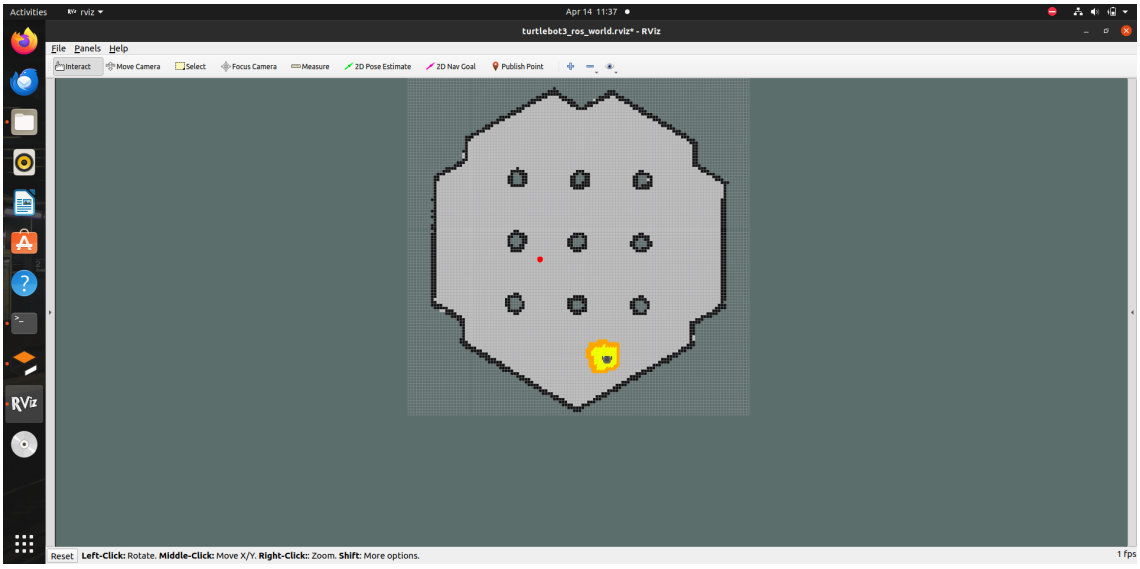

Fig. 4.1

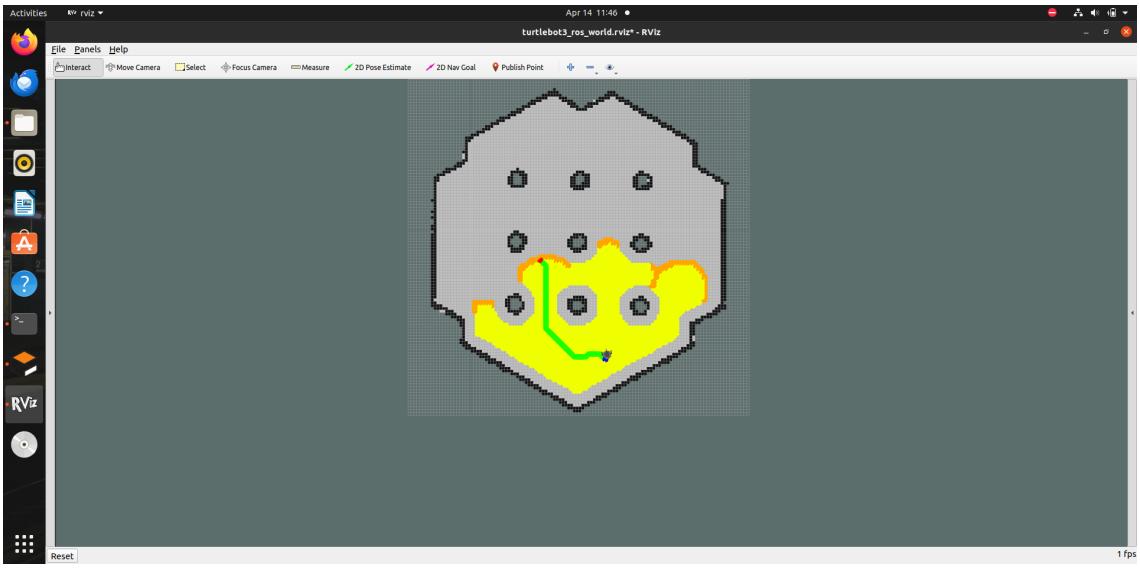

Fig. 4.2

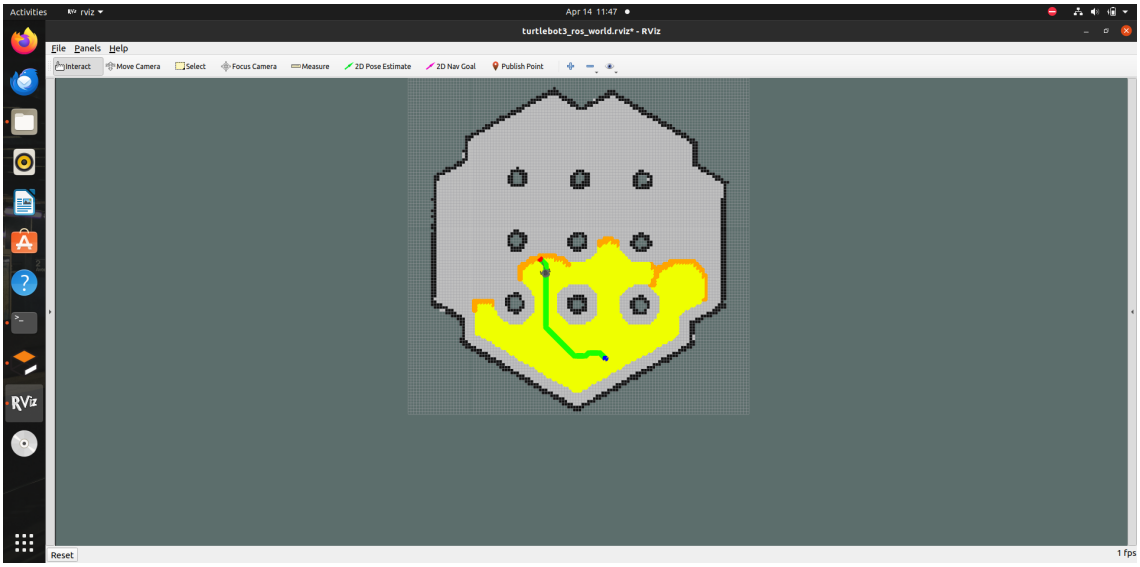
Activities
rviz
Apr 14 11:47
turtlebot3_ros_world.rviz* - RViz
File Panels Help
Interact
Move Camera
Select
Focus Camera
Measure
2D Pose Estimate
2D Nav Goal
Publish Point
Reset
1 fps

Fig. 4.3

A*(star) Algorithm : It's a combination of Dijkstra's algorithm and a heuristic search, which guides the algorithm towards the goal efficiently.This heuristic guides the search towards the goal, potentially making A* more efficient by exploring promising paths first.

In the first and second picture as we can see the algorithm is searching only for the nodes that contributes to the promising shortest path between the start point and goal point and then it finds the shortest path between the two points. In the 3rd picture we can see that the robot is moving towards the goal position along the trajectory generated by algorithm.

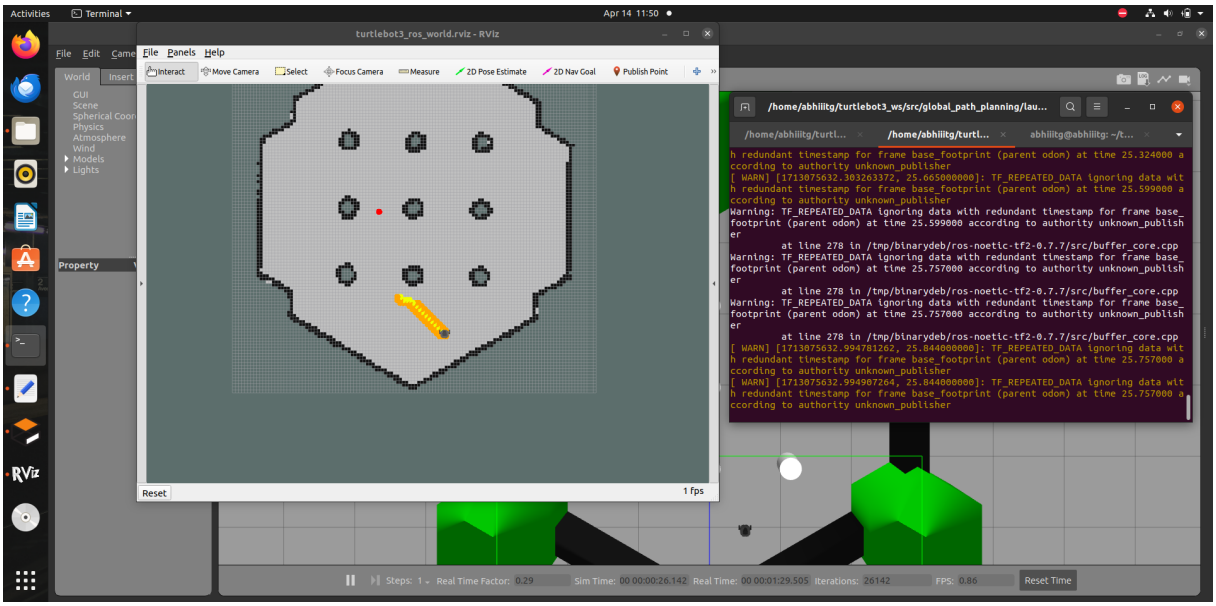

Fig. 4.4

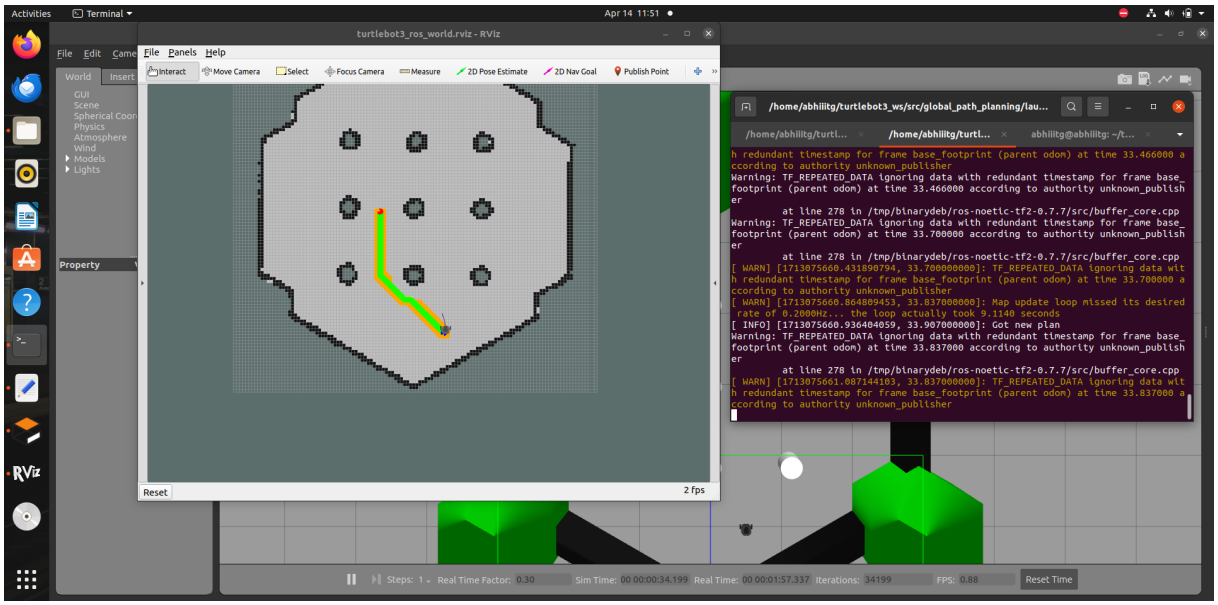
Activities
Terminal
Apr 14 11:51
turtlebot3_ros_world.rviz - RViz
File Panels Help
Interact Move Camera Select Focus Camera Measure 2D Pose Estimate 2D Nav Goal Publish Point
Reset
2 fps
/home/abhilitg/turtlebot3_ws/src/global_path_planning/lau...
Reset Time

Fig. 4.5

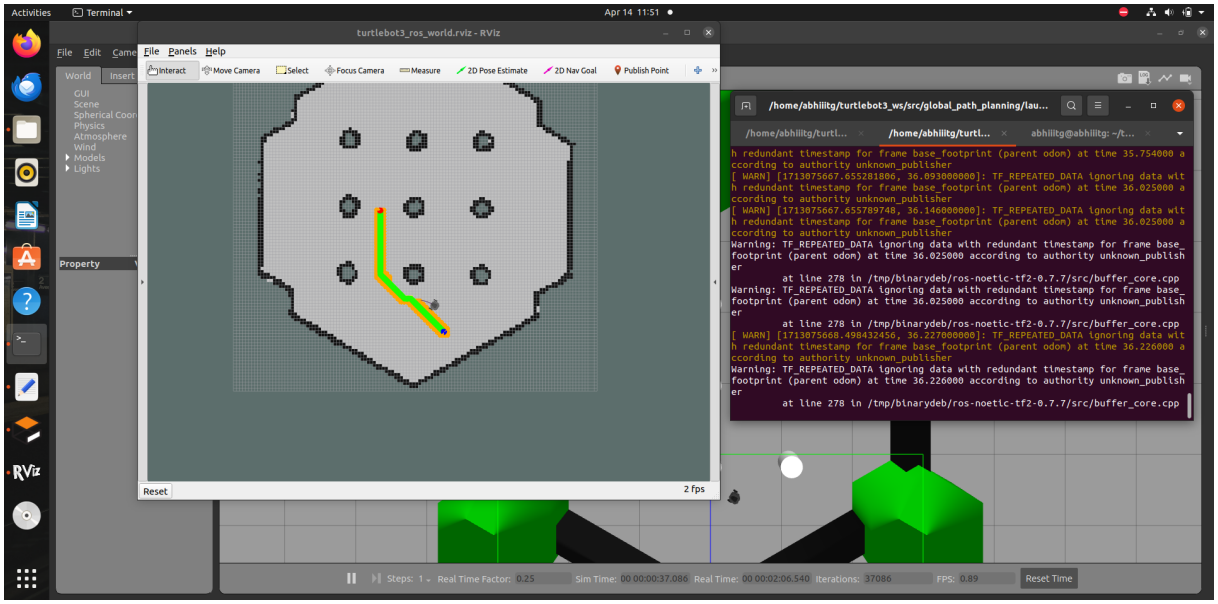
Activities
Terminal
Apr 14 11:51
turtlebot3_ros_world.rviz - RViz
File Panels Help
Interact Move Camera Select Focus Camera Measure 2D Pose Estimate 2D Nav Goal Publish Point
Reset
2 fps
/home/abhilitg/turtlebot3_ws/src/global_path_planning/lau...
Reset Time

Fig. 4.6

Chapter 5

Experimental Validation

The experimental validation of the aforementioned project involves transitioning from simulation to real-world implementation on a physical TurtleBot platform. While the simulation environment provides a valuable testing ground, running the same code on a physical robot presents several challenges. One major challenge is the discrepancy between the simulated environment and the real world, leading to differences in robot dynamics, sensor characteristics, and environmental conditions. This disparity can result in unexpected behaviors and performance discrepancies when deploying the code on the physical TurtleBot.

To address these challenges and ensure successful deployment on the physical robot, several adjustments may be necessary. Firstly, calibrating and fine-tuning the robot's sensors, such as lidar or cameras, is essential to accurately perceive the environment and detect obstacles. Additionally, modifications to the control algorithms may be required to account for discrepancies between simulated and real-world robot dynamics. This may involve adjusting parameters or implementing compensatory strategies to improve performance and robustness in the physical environment[5].

Furthermore, the code may need to be optimized for real-time execution on the TurtleBot's hardware platform, considering computational constraints and processing limitations. This may entail streamlining algorithms, reducing computational complexity, or leveraging hardware acceleration techniques to ensure timely and responsive operation.

Overall, the experimental validation process involves iteratively testing and refining the code on the physical TurtleBot, addressing discrepancies between simulation and reality, optimizing performance, and ensuring robustness and reliability in real-world scenarios. By carefully addressing these challenges and making necessary adjustments, it is possible to successfully deploy the path planning scheme on the physical TurtleBot platform and validate its effectiveness in navigating autonomously while avoiding obstacles.

Chapter 6

Conclusion and Future Work

In conclusion, the experimental validation of the path planning scheme on the physical TurtleBot platform has revealed performance discrepancies compared to the simulated environment. While the simulation results align with the desired outcomes, the real-world implementation encounters challenges related to differences in robot dynamics, sensor characteristics, and environmental conditions. Despite these discrepancies, the path planning scheme demonstrates promise and effectiveness in navigating autonomously while avoiding obstacles.

For future work, there is a clear need to address the performance gaps observed during physical experimentation. This includes further refining and optimizing the algorithm and code to enhance robustness and reliability in real-world scenarios. Additionally, improvements to sensor calibration, control algorithms, and trajectory optimization techniques may be explored to better align the behavior of the physical robot with simulated expectations. Moreover, leveraging advanced sensor technologies, such as depth cameras or advanced lidar systems, could enhance perception and obstacle avoidance capabilities, further improving the overall performance of the path planning scheme on the physical TurtleBot.

Overall, by continuing to refine and improve the algorithm and codebase, as well as incorporating advancements in sensor technology and control strategies, the path planning scheme holds promise for achieving more accurate and reliable navigation on the physical TurtleBot platform in future iterations of the project.

References

[1] J. Wang and D. Herath, *How to Move? Control, Navigation and Path Planning for Mobile Robots*, pp. 205–238. Singapore: Springer Nature Singapore, 2022.

[2] Z. Haruna, M. B. Mu'azu, A. Umar, and G. O. Ufuoma, "Path planning algorithms for mobile robots: A survey," in *Motion Planning for Dynamic Agents*, IntechOpen, 2023.

[3] H. Nemlekar, R. Khajuriwal, and N. Shah, "Robot navigation and path planning," *Dept. Elect. Eng., Worcester Polytechnic Institute, Worcester, MA*, 2018.

[4] Y. Zhang, K. Liu, F. Gao, and F. Zhao, "Research on path planning and path tracking control of autonomous vehicles based on improved apf and smc," *Sensors*, vol. 23, no. 18, 2023.

[5] L. Yang, J. Qi, D. Song, J. Xiao, J. Han, Y. Xia, *et al.*, "Survey of robot 3d path planning algorithms," *Journal of Control Science and Engineering*, vol. 2016, 2016.

[6] S. S. Ge and Y. J. Cui, "New potential functions for mobile robot path planning," *IEEE Transactions on robotics and automation*, vol. 16, no. 5, pp. 615–620, 2000.

[7] M. J. Barton, "Controller development and implementation for path planning and following in an autonomous urban vehicle," *Undergraduate thesis, University of Sydney*, 2001.

[8] D. Mishra, R. S. Yadav, and K. K. Agrawal, "Kinematic modelling and emulation of robot for traversing over the pipeline in the refinery," *Microsystem Technologies*, vol. 26, no. 3, pp. 1011–1020, 2020.

[9] S. Coenen, M. M. Steinbuch, M. van de Molengraft, J. Lunenburg, and G. Maus, "Motion planning for mobile robots—a guide," *Control Systems Technology*, vol. 79, 2012.