Title	Adaptive dynamic path planning design for improving safety and efficiency in mobile robot navigation
Sub Title	
Author	鄒, 皓若(Zou, Haoruo) 西村, 秀和(Nishimura, Hidekazu)
Publisher	慶應義塾大学大学院システムデザイン・マネジメント研究科
Publication year	2025
Jtitle	
JaLC DOI	
Abstract	
Notes	修士学位論文. 2025年度システムエンジニアリング学 第580号
Genre	Thesis or Dissertation
URL	https://koara.lib.keio.ac.jp/xoonips/modules/xoonips/detail.php?koara_id=KO40002001-00002025-0003

Master's Dissertation 2025

Adaptive Dynamic Path Planning Design for Improving Safety and Efficiency in Mobile Robot Navigation

Haoruo Zou
（Student ID Number：82334528）

Supervisor　Nishimura Hidekazu

September 2025

Graduate School of System Design and Management,
Keio University
Major in System Design and Management

SUMMARY OF MASTER'S DISSERTATION

Student Identification Number	82334528	Name	Haoruo Zou

Title

Adaptive Dynamic Path Planning Design for Improving Safety and Efficiency in Mobile Robot Navigation

Abstract

With the growing deployment of mobile robots in industry, logistics, and public services, there is an increasing demand for autonomous systems that can operate safely and efficiently in complex and dynamic environments. One of the key challenges is enabling robots to navigate in real time through dynamic scenarios, avoiding unexpected obstacles and maintaining smooth trajectories. Traditional path planning methods usually rely on static environmental maps and lack the responsiveness and adaptability to cope with real-time obstacle changes and uncertain environments.

This study addresses these challenges by designing an adaptive path planning system to improve the safety and execution efficiency of mobile robots. The proposed navigation framework integrates three core functions that work collaboratively: environment sensing through onboard sensors, global path generation based on a static map, and real-time path adjustment using a dynamically updated local cost map. These components are closely connected—sensor data continuously updates the local map, the global planner sets the overall route, and the local planner refines the trajectory in real time to avoid obstacles and ensure smooth motion.

The proposed planning framework is implemented and validated in a simulation environment constructed using the Robot Operating System (ROS). Comparative experiments with traditional static path planning strategies demonstrate significant improvements in path smoothness, obstacle avoidance success rate, and real-time responsiveness, demonstrating the system's adaptability and robustness in dynamic scenarios.

In summary, this study proposes a real-time adaptive path planning system that integrates global and local planning strategies, and enhances the handling of dynamic obstacles through an improved local path adjustment mechanism. However, when facing dense or fast-moving obstacles, the system's responsiveness and adaptability deteriorate in highly dynamic environments. Based on this observation, future work will focus on introducing obstacle motion prediction models and multi-robot collaboration mechanisms to improve the system's performance and applicability in complex real-world environments.

Keywords (5 words)

Mobile Robots, Dynamic Path Planning, Autonomous Navigation, Obstacle Avoidance, Robot Operating System (ROS).

Table of Contents

Table of figures

Chapter 1 Introduction

1.1 Research Background

The deployment of mobile robots has experienced significant growth across various industries, including manufacturing, logistics, and public services[1]. According to the International Federation of Robotics, the global average robot density in factories reached a record 162 units per 10,000 employees in 2023, more than doubling over the past seven years[2].This trend indicates a growing demand for automation systems that improve efficiency and safety.

Concurrently, the global mobile robots market has expanded rapidly[3], growing from $21.23 billion in 2023 to $25.81 billion in 2024, with a projected compound annual growth rate (CAGR) of 21.6%[4] . This growth is driven by advancements in artificial intelligence, sensor technologies[5], and the rising demand for automation in sectors such as logistics and healthcare.

However, despite continuous technological advancements, mobile robots still face many challenges[6] in path planning in dynamic environments. Traditional path planning methods such as A* and Dijkstra algorithms mostly rely on static environment maps and lack the ability to respond in real time to sudden changes in the environment (e.g., dynamic obstacles or unexpected situations)[7]. Such methods perform well in structured and predictable environments, but are prone to path inefficiencies, slow

response and even safety accidents in real dynamic scenarios[8].

In recent years, more and more research has focused on path planning strategies with adaptive capabilities to cope with dynamic changes in the environment[9][10]. For example, hybrid methods that integrate global and local planning have been shown to significantly improve navigation performance[11]. Meanwhile, data-driven based approaches such as reinforcement learning are also being explored for enhancing the robustness and adaptivity of path planning systems[12].

Based on the above background, it is particularly important to develop a dynamic path planning framework that combines global path generation with real-time local path adjustment[13]. This type of system is capable of dynamically adjusting the travelling path of a mobile robot based on continuously updated environmental information, thus achieving safe and efficient navigation in complex and dynamic environments.

1.2 Previous Research

Research on autonomous navigation for mobile robots has made substantial progress over the past two decades[14]. Key components such as environmental perception, localization, and path planning have been widely studied[15]. Among these, path planning plays a crucial role in enabling robots to reach a target location safely and

efficiently in both static and dynamic environments[16]. This section reviews existing work in global path planning, local path planning, SLAM technologies, and integrated approaches[17].

1.2.1 Global Path Planning Techniques

Global path planning is responsible for generating an initial route from the robot's start position to the goal based on a known map. Classical algorithms such as Dijkstra, A*, and D*-Lite have been widely applied due to their deterministic nature and guaranteed optimality on static maps[18]. A* in particular is favored for its balance between efficiency and accuracy using heuristic functions. However, these methods assume a static environment and struggle to adapt in real-time to unforeseen changes, such as dynamic obstacles or temporary barriers[19][20].

Recent research has attempted to improve global planning efficiency through heuristic optimization, hierarchical abstraction, or graph pruning. Yet, these improvements still rely heavily on accurate prior map information, limiting their robustness in un[21]structured or partially known environments[22].

1.2.2 Local Path Planning Techniques

Local path planning focuses on real-time obstacle avoidance and motion feasibility within the robot's immediate surroundings. Algorithms like Dynamic

Window Approach (DWA)[23], Artificial Potential Fields (APF), and Elastic Bands dynamically adjust the robot's trajectory based on sensor input and environmental changes[24]. DWA, for example, samples candidate trajectories based on the robot's kinematics and evaluates them using a cost function that considers safety, goal alignment, and speed.

While local planners offer high reactivity to dynamic environments, they may suffer from local minima, oscillation, or path instability, especially in cluttered spaces. Furthermore, without global guidance, they may fail to reach the target efficiently or even become trapped[25].

1.2.3 SLAM and Map Construction

Simultaneous Localization and Mapping (SLAM) enables robots to build a map of the environment while estimating their own position[26]. Algorithms such as Gmapping (based on particle filters), Hector SLAM, and Cartographer have become widely used in robotics platforms[27]. These mapping frameworks provide occupancy grids or costmaps that serve as inputs for planning algorithms. For localization, Adaptive Monte Carlo Localization (AMCL) remains a popular method for estimating the robot's pose on a known static map[28].

Despite the advancements, SLAM outputs can still suffer from drift, noise, or incomplete coverage in large or dynamic spaces[29][30]. Therefore, relying on SLAM

data alone for navigation may introduce uncertainty in planning performance.

1.2.4 Hybrid and Adaptive Path Planning Approaches

In recent years, integrated approaches that combine global and local planning strategies have gained traction[31][32]. These methods leverage the long-range efficiency of global planners and the real-time adaptability of local planners. Some studies have explored dynamic replanning frameworks where global routes are updated periodically or on-demand in response to environmental changes[33].

Additionally, research has begun exploring adaptive path planning algorithms, where parameters such as trajectory evaluation weights or costmap thresholds are dynamically tuned based on context (e.g., obstacle density, speed requirements)[34]. These adaptive methods aim to improve flexibility and safety but often require careful parameter design and extensive simulation to validate[35].

While these existing works have advanced the state of mobile robot navigation, challenges remain in achieving robust performance in highly dynamic, partially observable, or multi-agent environments. Therefore, this thesis aims to enhance path planning adaptability and reliability by optimizing the fusion of static and dynamic environment representations for mobile robots.

1.3 Research Purpose and Thesis Structure

Research Purpose

The primary purpose of this research is to design and implement a path planning system for autonomous mobile robots using the Robot Operating System (ROS) as the development platform. In recent years, the demand for safe, reliable, and efficient navigation systems has grown rapidly, especially in the context of autonomous vehicles and service robots operating in dynamic environments. To meet this demand, path planning must be capable of not only generating collision-free trajectories but also adapting to changing surroundings in real time.

This thesis aims to explore the integration of static and dynamic map information to enhance planning accuracy, and to evaluate the feasibility and performance of combining global and local path planning techniques in a simulation environment. Through simulation-based experiments, the research investigates the effectiveness of these techniques in typical indoor and semi-structured environments, validating their potential for real-world deployment.

Thesis Structure

This thesis is organized into five chapters as follows:

- Chapter 1 presents the background and motivation for robotic path planning, followed by a literature review of relevant techniques, and concludes with the

statement of research purpose and thesis organization.

- Chapter 2 describes the processes of map construction and robot localization. It explains SLAM implementation based on the particle filter and Gmapping algorithm, the AMCL-based localization approach, and the generation of static and cost maps.

- Chapter 3 focuses on the architecture of the proposed path planning system. It includes requirement analysis, system modeling using SysML, and behavioral modeling to describe the robot's navigation workflow.

- Chapter 4 discusses the design strategies for global and local path planning. It analyzes the principles and workflows of each approach, describes the robot's motion model, and explains how velocity sampling and trajectory evaluation are used to identify the optimal path.

- Chapter 5 conducts simulation experiments to evaluate the proposed methods. It first introduces the ROS simulation environment and tools, then presents simulation results from five stages including mapping, localization, and path planning in both 2D and 3D environments.

- Chapter 6 concludes the research by summarizing key findings and limitations. It also provides suggestions for future enhancements and real-world application prospects.

Chapter 2 Map Construction and Localization for Mobile Robots

2.1 Implementation of SLAM(Simultaneous Localization and Mapping)

Simultaneous Localization and Mapping (SLAM) refers to the process by which a robot, operating in an unknown environment and without prior knowledge of its own position, constructs a map of its surroundings while simultaneously determining its location within that map for autonomous navigation.

From this definition, it is clear that SLAM focuses on two core tasks: localization and map building. Commonly used SLAM algorithms include Hector, Cartographer, and Gmapping. In this study, a virtual simulation platform is used to perform path planning in an indoor environment, and the advantages and disadvantages of these algorithms are compared. Among them, Gmapping demonstrates notable advantages in small-scale indoor scenarios, offering low computational cost and high mapping accuracy. The algorithm effectively utilizes data from odometry to improve pose estimation, thereby reducing the dependency on high-frequency laser scan data.

2.1.1 Principle of Particle Filter

The main idea of the particle filter algorithm is to represent the posterior probability density by the sum of the weights of multiple random particles. Since the

integral process can be approximated by dividing the integration region into small parts and summing them up, the result of the integral can be estimated accordingly.

In a particle filter, there are two key models: the system model and the measurement model. These two models are used to predict the internal state of the system.

Assuming the state equation of the system is known, it can be described as follows (Equation 2-1):

$$x_k = f(x_{k-1}, u_{k-1}) \tag{2-1}$$

In this equation:

x_k——the current state of the system;

f——the state transition function;

u——the process noise

The measurement equation of the system is shown as follows (Equation 2-2):

$$y_k = h(x_k, v_k) \tag{2-2}$$

Where:

y_k——the measurement data at time step k;

h——the measurement function;

v——the observation (measurement) noise.

The posterior probability can be derived through the prediction and update steps. The prediction step utilizes the system model, while the update step uses the latest observed data to revise the estimated posterior distribution.

The prediction step computes the prior probability of the current state based on the state transition function and the probability distribution of the previous state. Assuming the system follows a first-order Markov process—i.e., the current state depends only on the previous state—the prior can be expressed as:

$$p(x_k|y_{1:k-1}) = \int p(x_k, x_{k-1}|y_{1:k-1})\, dx_{k-1}$$

$$= \int p(x_k|x_{k-1}, y_{1:k-1}) p(x_{k-1}|y_{1:k-1})\, dx_{k-1}$$

$$= \int p(x_k|x_{k-1}) p(x_{k-1}|y_{1:k-1}) dx_{k-1} \quad (2\text{-}3)$$

Here, $p(x_k|x_{k-1})$ is determined by the known state transition function.

The update step incorporates the current measurement to obtain the posterior probability:

$$p(x_k|y_{1:k}) = \frac{p(y_k|x_k) p(x_k|y_{1:k-1})}{p(y_k|y_{1:k-1})} \quad (2\text{-}4)$$

$$p(y_k|y_{1:k-1}) = \int p(y_k|x_k) p(x_k|y_{1:k-1}) dx_k \quad (2\text{-}5)$$

In this equation, $p(y_k|x_k)$ is the likelihood function, and $p(x_k|y_{1:k-1})$ is the

prior obtained from the prediction step. Once the posterior distribution at time k is obtained, it can be passed to the next prediction cycle, thus forming a recursive estimation process.

However, applying the above recursive steps directly in nonlinear and non-Gaussian systems is challenging due to the integration involved. Therefore, the Monte Carlo method is used to approximate the distribution.

Since the exact posterior cannot be obtained analytically, importance sampling is applied to estimate the posterior distribution. Although the posterior can be approximated in this way, the computation of weights for each particle is computationally expensive and inefficient. To address this issue, Sequential Importance Sampling (SIS) is introduced, which estimates both the posterior density and the weight function in a recursive manner.

Nevertheless, SIS tends to suffer from degeneracy, where many particles have negligible weights, leading to high computational cost. To alleviate this problem, the concept of Effective Sample Size (ESS) is introduced. Resampling is triggered only when the ESS falls below a predefined threshold. This strategy helps to reduce degeneracy and improve the overall efficiency of the particle filter.

2.1.2 Principle of Gmapping Algorithm

The Gmapping algorithm computes the joint posterior probability density

function of a robot's trajectory and the map given sensor observations and odometry readings. Specifically, let

$z_{1:t} = z_1, z_2, ... z_t$ be the sequence of observations obtained via the measurement model,

$u_{1:t-1} = u_1, u_2, ... u_{t-1}$ be the odometry inputs,

$x_{1:t} = x_1, x_2, ... x_t$ be the robot's trajectory,

m: be the environment map.

The joint posterior can be written as:

$$p(x_{1:t}, m | z_{1:t}, u_{1:t-1})$$

Through Rao–Blackwellized Particle Filtering (RBPF), this factorizes into two terms:

$$p(x_{1:t}, m | z_{1:t}, u_{1:t-1}) = p(m | x_{1:t}, z_{1:t}) \times p(x_{1:t} | z_{1:t}, u_{1:t-1}) \quad (2\text{-}6)$$

By separating the map and trajectory distributions, the algorithm reduces the dimensionality of the estimation problem and significantly lowers computational cost.

- Trajectory estimation.

Each particle represents a hypothesis of the robot's path. At every time step, particles are propagated according to the motion model (from u_{t-1}) and re-weighted by comparing predicted observations with actual measurements using an inverse sensor model (e.g., beam-based or endpoint-based).

- Map update.

Once weighted trajectories are available, the algorithm updates an occupancy grid map m by integrating the observations z_t along each particle's path, incrementally refining the representation of free and occupied space.

Because robust performance typically requires hundreds to thousands of particles, the computational burden can be substantial—especially on robots with limited onboard processing power. To alleviate this, Gmapping employs an adaptive resampling strategy:

A judgment criterion (often based on the effective sample size, ESS) is introduced.

Resampling is triggered only when ESS falls below a predefined threshold, preventing unnecessary particle replacement and reducing both particle impoverishment and overall computation.

Gmapping excels at building high-resolution maps in small- to medium-scale environments. However, its reliance on large particle sets and the lack of explicit loop-closure detection make it less suitable for very large or highly looping trajectories, where accumulated drift can degrade map consistency. Integrating pose-graph optimization or scan-matching-based loop closure modules is a common extension to overcome these limitations.

2.2 Robot Localization with AMCL

To achieve autonomous navigation, the robot needs to determine its own location on a map as well as the desired destination, which implies that both the environmental map and the autonomous vehicle's position must be accurately identified. In this research, the Adaptive Monte Carlo Localization (AMCL) algorithm is employed for localization. AMCL utilizes a particle filter approach to track and estimate the robot's pose on a known map.

The Robot Operating System (ROS) has a unique mechanism called TF (transform), which allows developers to track multiple coordinate frames simultaneously and facilitates convenient transformations between these frames. At the beginning of localization using AMCL, distance measurements are acquired from a LiDAR sensor. Given that the LiDAR sensor is fixed at a known position on the robot, the measured distances from LiDAR to obstacles can be directly translated into distances from the autonomous vehicle itself to these obstacles.

Subsequently, a TF transformation is performed between the robot's base coordinate frame /base_link and the odometry coordinate frame /odom. This transformation primarily relies on odometry data. Finally, the map coordinate frame /map is transformed into the odometry frame /odom, a process implemented through the stdr_amcl node. The TF transformation process is illustrated in Figure 2-1.

When running the stdr_amcl package, the particle filter-based localization approach provides pose information relating the /base_link frame directly to the /map frame. However, since the parent node of /base_link is already /odom, stdr_amcl instead publishes the transform from /map to /odom. This approach also provides an additional benefit—it allows the system to directly measure and accumulate odometry errors overtime.

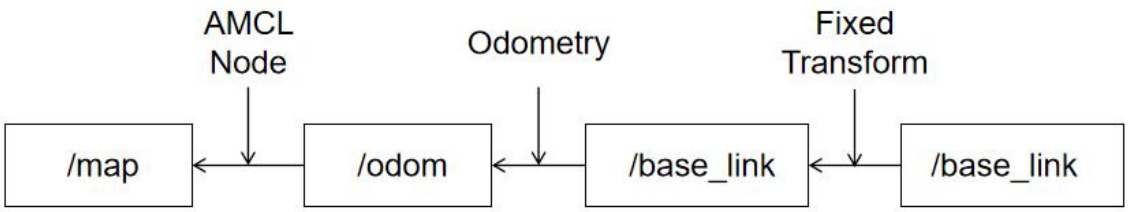


Figure 2- 1 TF Transformation Diagram

In practice, this TF tree tightly couples global consistency with local responsiveness. By publishing the /map → /odom transform, AMCL continuously corrects odometry drift using map-based pose estimates. Downstream nodes—such as path planners and obstacle avoidance modules—simply listen to /odom → /base_link for real-time motion and to /map → /odom for drift-corrected positioning, without needing to fuse data themselves. The fixed /base_link → /laser_frame transform ensures all LiDAR scans align correctly, enabling precise scan-to-map matching.

Moreover, because /base_link remains a child of /odom, the difference

between the raw odometry transform and the map-corrected transform directly reflects accumulated odometry error. Monitoring this discrepancy in real time aids in diagnosing sensor degradation or wheel slippage, and can trigger adaptive resampling thresholds in AMCL to maintain robust localization under challenging conditions.

2.3 Static Map Construction

Static map construction is implemented using the stdr_gmapping software package, which conveniently creates corresponding two-dimensional grid maps on the ROS platform. During the map-building process, it subscribes to odometry, LiDAR, and TF topic nodes to obtain data and subsequently publishes the constructed map. The parameters of the static map can be adjusted within the stdr_gmapping function package, allowing the construction of various maps by modifying relevant parameters.

Before running the code, it is necessary to adjust certain parameters within the source code, such as changing the map's topic name. This is because, in the STDR simulation environment, the /map topic is already used to publish the simulated map from STDR. If not remapped, this could cause conflicts and prevent the STDR-generated map from functioning correctly. The data transmission process is illustrated in Figure 2-2.

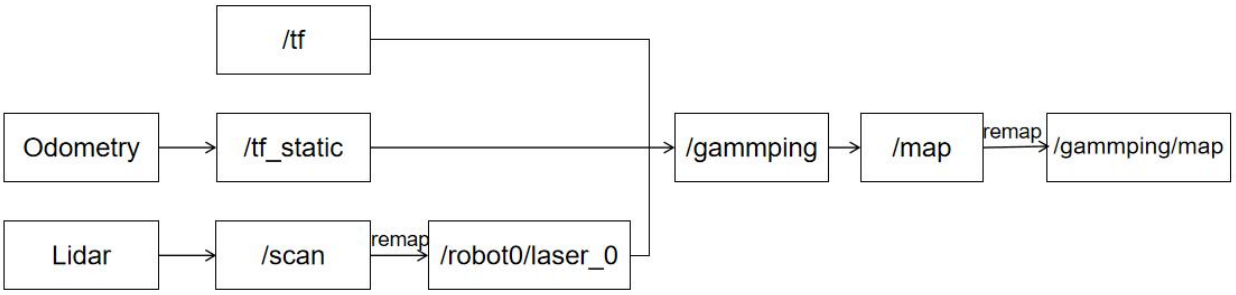


Figure 2- 2 Gmapping Data Transmission Diagram

To obtain a high-quality yet efficient grid map, several key parameters must be balanced. The resolution parameter sets each cell's side length: higher resolution captures finer environmental details but also increases memory usage and the time required to update the grid. Bounding the map with xmin, ymin, xmax and ymax limits the coverage area to only the region of interest, reducing computation and improving real-time performance. Gmapping fuses successive LiDAR scans using a log-odds representation and applies the occ_thresh and free_thresh thresholds to decide when a cell becomes occupied or free, thereby enhancing robustness against sensor noise. The update_rate and particles parameters control how often the map is refreshed and how many particles contribute to each update: raising these values improves map consistency but places a heavier load on the CPU.

Once mapping is complete, the map_saver node can export the grid as a PGM

image alongside a YAML metadata file. In future sessions, loading this saved map via the map_server node allows navigation and planning modules to run without re-invoking Gmapping. In environments containing moving objects, static mapping may inadvertently record transient obstacles as permanent; to address this, one may pre-filter incoming scans—discarding points with abnormally high variance—or apply morphological post-processing to the saved PGM to eliminate spurious artefacts.

In simulation contexts, abundant computational resources permit more aggressive settings, such as higher resolution and larger particle counts. On physical robots, however, parameters like map_update_interval, maxRange, and sigma (the sensor noise model) should be tuned carefully to match available CPU capacity and real-time requirements, ensuring the mapping process does not overwhelm the system. By combining these parameter-tuning strategies with appropriate pre- and post-processing, one can produce high-quality, reusable static maps that form a robust foundation for downstream autonomous navigation tasks.

2.4 Cost Map Generation

While SLAM produces a static occupancy grid, real-world navigation environments often feature the sudden appearance or disappearance of obstacles.

Because a static map cannot reflect such changes in real time, successful autonomous navigation requires a dynamic representation—namely, the cost map. In our framework, cost maps are divided into a global cost map for high-level path planning and a local cost map for reactive obstacle avoidance.

Creating a cost map involves layering additional information atop the static map. The base layer is the occupancy grid generated by SLAM, which is first written into the global cost map. Above this sits the obstacle layer, which records the current positions of all detected obstacles. Finally, the inflation layer adds a safety buffer around each obstacle: during initialization, the inflation radius is set to be at least equal to the robot's radius, thereby ensuring that planned trajectories maintain sufficient clearance to prevent collisions. Together, these layers form a continuously updated cost map that reflects both the static environment and any dynamic changes.

All parameters governing cost map construction—such as layer update rates, obstacle map thresholds, and inflation radius—can be configured within the stdr_move_base ROS package, allowing users to tailor the responsiveness and granularity of the cost map to their specific robot and operating conditions

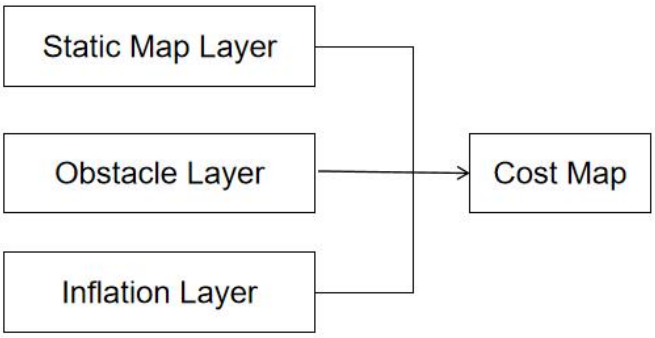


Figure 2- 3 Cost Map Construction Diagram

This diagram illustrates how the cost map is built by layering dynamic information on top of a static occupancy grid. First, the Static Map Layer provides the baseline occupancy grid generated by SLAM. Next, the Obstacle Layer overlays real-time obstacle detections (e.g., from LiDAR) onto that grid. Finally, the Inflation Layer adds a safety buffer around each obstacle—its radius is set to at least the robot's radius—so that any planned trajectory maintains sufficient clearance. The three layers are then fused into the final Cost Map, which drives both global and local path planning by reflecting both the static environment and dynamic changes in real time.

Evaluation Criteria: Definition of Cost and Effectiveness

In autonomous navigation systems, particularly those involving dynamic environments, the evaluation of planning performance must consider both cost and effectiveness. These two aspects form the core of the trajectory evaluation mechanism, guiding the selection of optimal motion commands during local path planning. In this

section, we define these concepts in the context of mobile robot navigation and explain how they are modeled, quantified, and balanced in the proposed system.

- **Definition of Cost**

The term cost refers to the quantified penalty or traversal difficulty associated with a specific trajectory, grid cell, or region within the robot's environment. In this study, cost is used in two key contexts:

Global costmap: constructed from the static map; represents fixed environmental structures (e.g., walls).

Local costmap: updated in real time with sensor input; reflects dynamic obstacles and transient hazards.

Each trajectory evaluated by the local planner is associated with a trajectory cost, which is calculated based on several contributing factors:

Distance cost (C_{dist}): favors paths that progress toward the goal.

Obstacle cost(C_{obs}): penalizes trajectories that come close to obstacles.

Velocity cost(C_{vel}): penalizes commands with lower translational velocity, to discourage inefficient motion.

These are integrated into a weighted cost function:

$$C_{total} = \alpha \cdot C_{dist} + \beta \cdot C_{obs} + \gamma \cdot C_{vel} \quad (2\text{-}7)$$

where α, β, γ are designer-defined weights, and all terms are normalized to allow fair comparison.

The costmap representation ensures that high-cost areas (e.g., near obstacles or dead ends) are automatically avoided during local planning, while the cost function ensures trajectory-level decisions optimize safety and efficiency simultaneously.

- **Definition of Effectiveness**

While cost emphasizes penalty avoidance, effectiveness evaluates the outcome quality of the navigation behavior. An effective path planning system should enable the robot to complete its navigation task with high reliability, smoothness, and responsiveness.

Effectiveness in this study is assessed using the following criteria:

- Goal-reaching success rate: the percentage of runs where the robot successfully reaches the goal.
- Path smoothness: the angular deviation between consecutive motion commands; smoother paths are desirable for energy efficiency and mechanical stability.
- Time to goal: total time taken to complete the task.
- Obstacle avoidance success rate: whether the robot collides with any dynamic

obstacle during execution.

These indicators are collected during simulation experiments and compared across different planning strategies to demonstrate the superiority of the proposed adaptive system.

- **Quantification of Evaluation Metrics**

Each metric is computed using the following definitions:

Path Length (L):

$$L = \sum_{i=1}^{n-1} \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2} \quad (2\text{-}8)$$

Smoothness (S):

$$\mathrm{S} = \sum_{i=2}^{n-1} |\theta_{i+1} - \theta_i| \quad (2\text{-}9)$$

Success Rate (R):

$$\mathrm{R} = \frac{Successful\ Runs}{\text{Total Runs}} \times 100\% \quad (2\text{-}10)$$

Average Time (T):

$$\mathrm{T} = \frac{1}{\mathrm{N}} \sum_{i=1}^{N} \left(t_{goal}^{(i)} - t_{start}^{(i)} \right) \quad (2\text{-}11)$$

These metrics allow for consistent and objective comparison between static and adaptive planning approaches.

- **Balancing Cost and Effectiveness**

In practice, there is often a trade-off between minimizing cost and maximizing effectiveness. For example, the shortest path may pass dangerously close to obstacles, increasing collision risk. Conversely, the safest path may be unnecessarily long or slow.

The proposed system addresses this by dynamically balancing the cost terms in the local planner's scoring function using adjustable weights. These weights can be tuned depending on the mission objective:

- In dense environments, higher weight is assigned to obstacle cost.
- In open spaces, more priority is given to speed and efficiency.

This flexible mechanism enables the planner to adapt its behavior according to context, achieving a balanced compromise between safety and task completion.

To conclude, cost provides a framework for penalizing undesirable paths, while effectiveness evaluates the quality of actual navigation outcomes. The integration of these two aspects in the local planner's evaluation mechanism enables the robot to plan trajectories that are both safe and goal-directed. The effectiveness of this approach is validated through simulation experiments presented in Chapter 5.

Chapter 3 Architecture Definition of Path Planning

In this chapter, I apply a rigorous Model-Based Systems Engineering (MBSE) methodology to define the architecture of our Adaptive Dynamic Path Planning System. I begin by eliciting and formalizing stakeholder needs, quality attributes, and environmental constraints into a structured requirements model. These requirements are then translated into fully traced SysML use-case diagrams (Section 3.1), which serve as the authoritative source for system functionality and acceptance criteria. Building on this foundation, I decompose the system into a static component architecture using SysML Block Definition and Internal Block Diagrams (Section 3.2), thereby establishing clear interfaces, data flows, and allocation of responsibilities. Finally, I capture the end-to-end planning logic and runtime behaviors with SysML activity and state-machine diagrams (Section 3.3), enabling early verification and validation through model-in-the-loop simulations and parametric checks. By weaving together requirements management, use-case modeling, structural decomposition, and behavioral specification within an MBSE tool-chain, I ensure traceability from high-level goals down to implementation artifacts, support automated consistency checking, and provide a robust blueprint for subsequent development, integration, and testing.

3.1 Requirements Analysis and Use Cases

Our Model-Based Systems Engineering (MBSE) process begins with a systematic requirements analysis to capture all stakeholder needs, performance objectives, and environmental constraints. I elicited functional requirements—such as the ability to receive and validate high-level navigation goals, compute collision-free global routes, refine trajectories locally around dynamic obstacles, and hand off executable paths to the motion controller. Non-functional requirements were defined to guarantee real-time performance (planning latency ≤ 100 ms), enforce safety margins (clearance ≥ 1.2 × robot radius), and bound resource usage (CPU utilization < 40 %). Interface requirements specify strongly typed ROS messages (PoseStamped, LaserScan, PathMsg) and TF transforms to ensure coordinate consistency. Environmental constraints document expected obstacle densities (up to 0.1 obstacles/m^2), map resolutions (0.05 m grid), and sensor characteristics (LiDAR range ≥ 10 m, update rate ≥ 10 Hz). Each requirement was formalized in a SysML requirements repository with a unique identifier, rationale, and acceptance test criterion, establishing full traceability for downstream design and verification activities.

From this requirements foundation, I distilled the system's core functionality into a single, mission-critical use case—Generate Adaptive Path—which integrates three sub-functions: Global Path Generation, Local Path Refinement, and Dynamic

Re-planning. Figure 3-1 shows the corresponding SysML use-case diagram:

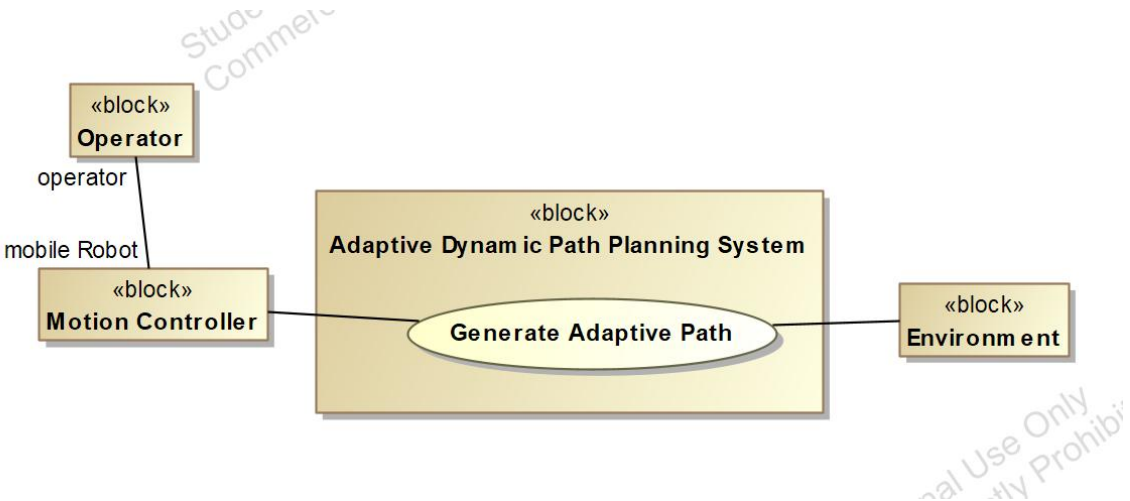


Figure 3- 1 Use Case Diagram

In the use-case diagram, three primary actors interact with the Generate Adaptive Path service. The Operator issues navigation objectives—such as target pose, arrival tolerance, and override commands—while the Environment continuously provides real-time obstacle information via LiDAR scans and semantic labels. The Motion Controller then executes the planner's output, converting published trajectories into low-level velocity and steering commands for the robot.

For Generate Adaptive Path, I defined the following conditions and flows. Preconditions require that the static occupancy map is loaded and validated, all TF transforms (/map → /odom → /base_link → /laser_frame) are initialized, and sensor streams (LiDAR and odometry) remain active and within operational bounds. Triggers include the arrival of a new navigation goal from the Operator or the detection of a

significant obstacle update entering the local planning window. The primary flow proceeds through six steps: accessing the static map, computing the global path using graph-search or sampling techniques, updating the local costmap with live obstacle data, generating a local path, adapting the path to account for newly detected obstacles, and finally publishing the trajectory to the Motion Controller. Post-conditions stipulate that a time-parameterized trajectory—meeting both the latency constraint (≤ 100 ms) and the clearance requirement (≥ 1.2 × robot radius)—is published and acknowledged by the Motion Controller before execution begins. Exception flows handle failure scenarios: if no valid path is found, the system initiates a safe-stop maneuver and notifies the Operator; if a sensor dropout occurs, it switches to a degraded planning mode, expands safety buffers, and attempts sensor re-initialization.

I validated this use case through scenario-based walkthroughs, covering nominal operations (steady progress to target), edge cases (sudden obstacle emergence), and fault conditions (sensor failures). Each scenario was mapped to specific requirements and use-case steps, ensuring that every requirement is exercised by at least one use case. This requirement-driven use-case model establishes the authoritative functional scope and acceptance criteria, providing a solid foundation for the structural and behavioral models in Sections 3.2 and 3.3.

3.2 Structural Modeling

Building on our use-case foundation, I constructed a SysML Block Definition Diagram (BDD) to establish the system's static software and hardware components. Figure 3-2 illustrates this top-level decomposition:

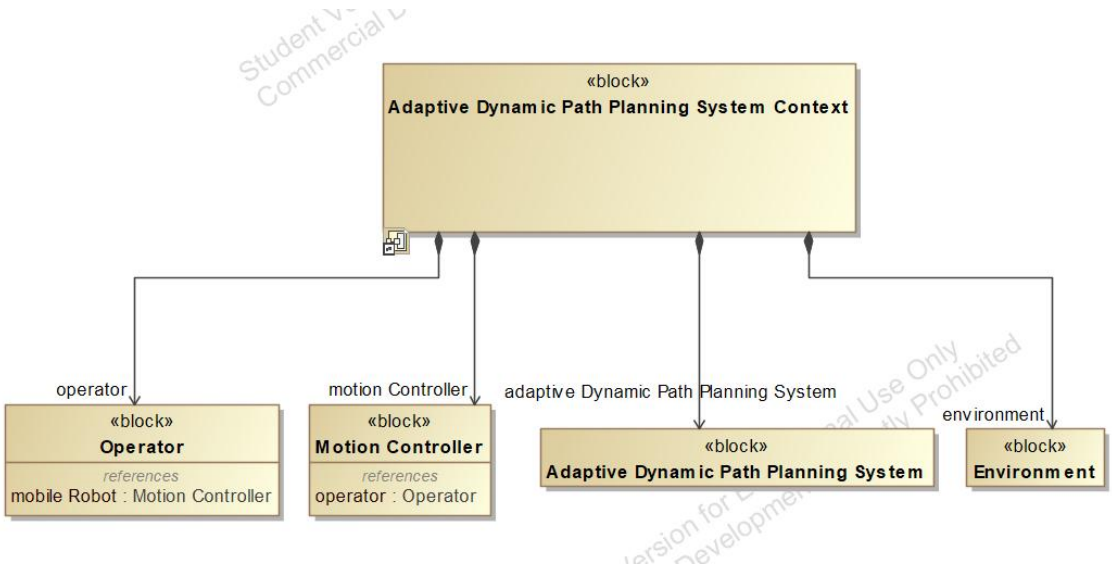


Figure 3- 2 Block Definition Diagram

Figure 3-2 shows the top-level BDD, where the Adaptive Dynamic Path Planning System Context block composes four principal sub-blocks:

- Operator, allocated to the "Send Navigation Goal" use case, encapsulates the human or supervisory interface.
- Environment, allocated to "Provide Obstacle Information," represents the world model and live sensor feeds.

- Adaptive Dynamic Path Planning System, allocated to “Generate Adaptive Path” and its constituent functions, contains the core path-planning algorithms.
- Motion Controller, allocated to “Execute Planned Path,” interfaces with robot actuators to follow computed trajectories.

Composition links (black diamonds) indicate a part-of relationship, while plain associations denote the direction of data or control flow. By mapping each block to specific use cases and requirements, the BDD guarantees that every requirement has a responsible component and supports independent evolution of subsystems.

Figure 3-3 presents the Internal Block Diagram of the Adaptive Dynamic Path Planning System, detailing its port-based interfaces to the Operator, Motion Controller, and Environment components:

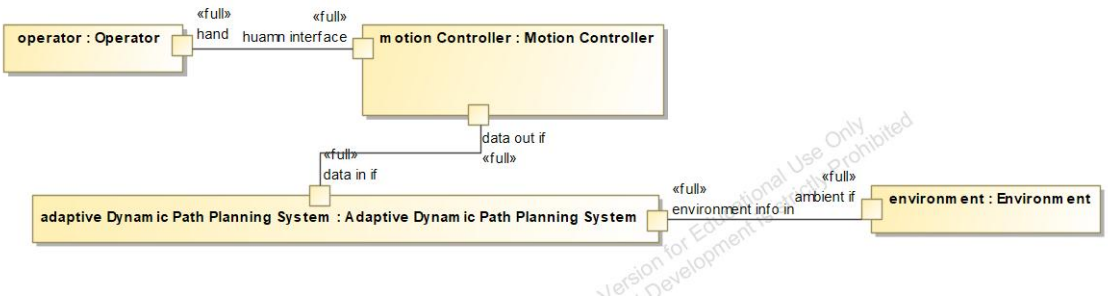


Figure 3- 3 Internal Block Diagram of the Adaptive Dynamic Path Planning System

- The hand port on the Planner connects to the Operator's human interface port, through which navigation goals, overrides, and cancellations are delivered.
- A pair of complementary ports—data in if and data out if—link the Planner to the Motion Controller. Trajectory messages (PathMsg) flow out of the Planner's data out if into the Controller's data in if, while execution feedback or updated goals travel back via the Controller's data out if into the Planner's data in if, enabling closed-loop re-planning.
- The environment info in port receives live obstacle and map data from the Environment's ambient if port, ingesting LiDAR scans, semantic labels, and dynamic obstacle updates into the planning algorithms.

Each port is strongly typed—using message definitions such as PoseStamped, LaserScan, and TrajectoryMsg—to enforce unambiguous data exchange and support automated interface verification. By explicitly modeling these interfaces, the IBD ensures a clean separation of concerns between human command, low-level actuation, and environmental perception, while providing a precise blueprint for code generation and integration testing within the ROS system.

3.3 Behavioral Modeling

As shown in the activity diagram (Figure 3-4), I capture the dynamic behavior

of the path-planning system as a single, end-to-end workflow to ensure every step—from goal issuance to execution—is explicit and traceable. The process begins when the Operator invokes the Send Navigation Goal action, handing the target pose to the Motion Controller. The Controller's Receive Function decouples human directives from the planning algorithm, acting as a lightweight validation and dispatch mechanism.

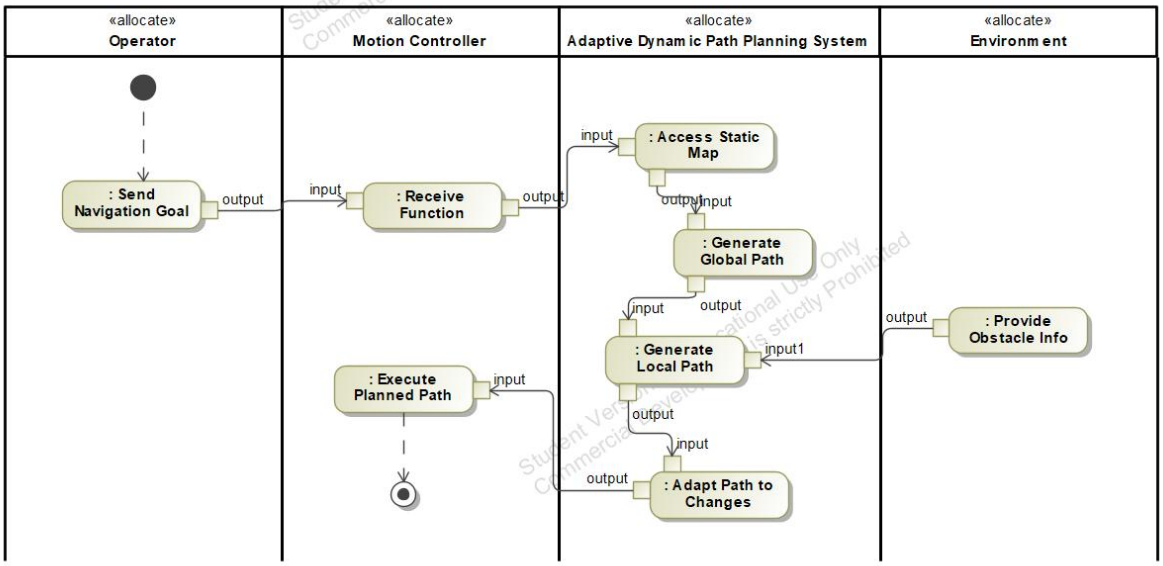


Figure 3- 4 Activity diagram

Once the goal reaches the Adaptive Dynamic Path Planning System, the planner first accesses the static map to load the latest occupancy grid. It then computes a Global Path on this static representation using graph-search or sampling-based techniques, guaranteeing a collision-free route under nominal conditions.

Meanwhile, the Environment module continuously provides obstacle

information (e.g., live LiDAR scans). These data feed into the Generate Local Path activity, where the local planner refines the global trajectory to avoid dynamic obstacles. The loop back from Adapt Path to Changes represents real-time re-planning: whenever new obstacle data arrive or the goal is updated, the system re-evaluates and adjusts the local path without restarting the entire global planning process.

Finally, the refined trajectory is sent back to the Motion Controller for Execute Planned Path, closing the loop with the robot's actuators. Entry and exit actions—such as sensor verification, map consistency checks, and latency monitoring—are implied by guard conditions on decision nodes. Exception flows (e.g., "No valid local path → trigger safe-stop" or "Map access failure → request map reload") are defined in the full model to ensure robust fault handling.

Chapter 4 Design of Path Planning Algorithms

4.1 Overview of Path Planning and Design Strategy

Global path planning in our system operates on the complete, static occupancy grid produced by SLAM and sensor fusion. Once the robot's current pose is localized on this map, the planner computes a fixed, collision-free route from the start to the goal position by traversing the known environment representation. This global trajectory provides the high-level guidance that directs the robot through large-scale spaces under nominal conditions.

As the robot follows this precomputed path, it inevitably encounters regions where obstacles have appeared, moved, or were not captured in the original map. To maintain safety and adaptability, I employ a real-time local planning layer that operates within a rolling window centered on the robot. In each control cycle, live sensor data—principally LiDAR scans—are fused into a multi-layer cost map. This cost map comprises the static occupancy layer, a dynamic obstacle layer, and an inflation layer that enforces a configurable safety buffer around each hazard. The local planner continuously re-evaluates and adjusts the trajectory within this window, sampling feasible motion commands and selecting those that minimize cost while respecting kinematic and clearance constraints. Because the environment is only partially known and constantly evolving, this local planning process demands both low latency and

substantial computational effort.

Our overall design strategy weaves these two layers into a cohesive framework defined by clear module boundaries, rigorously specified interfaces, and end-to-end timing guarantees. The global planner publishes its route via ROS topics, while the local planner subscribes to that route, TF transforms, and live obstacle feeds to update the cost map and compute motion commands. All data exchanges use strongly typed message definitions (PoseStamped, LaserScan, PathMsg), ensuring unambiguous contracts. Key parameters—such as cost-map resolution, inflation radius, and planning cycle frequency—are calibrated through extensive simulation and hardware-in-the-loop testing to verify that the local planner meets the 100 ms latency budget. Robustness is further enhanced by built-in fault-handling routines: if the local planner fails to find a valid trajectory, a safe-stop maneuver is executed and the Operator is alerted; sensor or map inconsistencies trigger automated re-initialization procedures. By combining a stable global route with a reactive local planner—and by enforcing strict interface and performance requirements—our architecture delivers both efficient long-range navigation and safe, responsive obstacle avoidance in complex, dynamic environments.

- **Coordination Between Global and Local Path Planning**

The path planning system adopted in this study follows a hierarchical

architecture composed of two complementary components: Global Path Planning and Local Path Planning. These components fulfill different roles and operate at different spatial and temporal resolutions, yet they must function in tight coordination to enable robust autonomous navigation.

Global Path Planning operates on the static map, which is typically generated using SLAM. It generates an initial path from the robot's start position to the goal location, avoiding known static obstacles. This path represents the long-term strategy for goal-reaching and is computed before the robot starts moving. However, it assumes a static environment and cannot react to dynamic changes such as moving people or newly introduced obstacles.

Local Path Planning, in contrast, functions at runtime and is responsible for short-term trajectory generation based on real-time sensor data. It continuously samples velocity commands and evaluates multiple candidate trajectories using a cost function that accounts for obstacle avoidance, goal direction, and velocity smoothness. This module uses a local costmap, which dynamically updates with detected obstacles, allowing the robot to re-plan its motion in real-time.

The coordination between the two modules is as follows:

- The global path provides a reference trajectory that helps the local planner remain goal-oriented, preventing it from becoming trapped in local minima or

deviating excessively.

- The local planner ensures safety and adaptability by modifying the motion commands when dynamic obstacles are encountered, even if such changes deviate temporarily from the global path.

This hierarchical division of labor ensures that the robot benefits from both global optimality and local reactivity, enabling reliable navigation in complex and dynamic environments. Figure 5-6 illustrates this interaction, where the robot follows the global path while dynamically adjusting its trajectory through local re-planning.

4.2 Global Path Planning Algorithms

Over decades of research, a variety of global path-planning techniques—such as Dijkstra's algorithm, A*, and RRT—have proven effective in diverse navigation contexts. In our framework, I adopt the A* algorithm as the backbone for high-level route computation. First introduced in 1968 by Peter Hart, Nils Nilsson, and Bertram Raphael, A* remains one of the most popular choices for graph-based and grid-based path finding, thanks to its balance of performance, reliability, and ease of integration.

Within our system, the A* planner operates on the static occupancy grid produced by SLAM, reading in obstacle information and map topology to generate an optimal, collision-free sequence of waypoints. We leverage open-source ROS

implementations—configuring grid resolution, movement costs, and heuristic weightings—to tune search speed and path smoothness for our vehicle's kinematic constraints. Because A* exhibits deterministic behavior and predictable resource usage, it runs efficiently on embedded onboard computers, ensuring that global route updates complete within the allotted planning window.

Moreover, A*'s design allows seamless coupling with our layered cost-map architecture: static obstacles, precomputed risk weights, and heuristic guidance merge naturally into a unified search space. In practice, this yields stable, repeatable global paths across both simulated and real-world environments. By basing our global planner on A*, we benefit from its mature tooling, extensive community support, and proven track record—providing a solid foundation upon which our dynamic, real-time local planning layer can safely and responsively navigate.

4.2.1 Principle of Global Path Planning Algorithm

As a heuristic search algorithm, A* uses heuristic information to guide the path-search process, thereby reducing the search space, lowering complexity, and decreasing computational load. A* is typically employed for global planning, and its core expression is:

$$F(n) = G(n) + H(n) \qquad (3\text{-}1)$$

where $G(n)$ is the accumulated cost from the start node to the current node n, and $H(n)$ is the heuristic estimate of the cost from n to the goal. The sum $F(n)$ thus reflects the total estimated cost of a path passing through nnn. In A*, the choice of $H(n)$ is critical—an admissible, consistent heuristic (one that never overestimates the true remaining cost) both guarantees optimality and dramatically reduces the number of nodes the algorithm must examine.

A* operates on three core data structures:

- Open List: a priority queue of frontier nodes, ordered by increasing $F(n)$, from which the next candidate is selected.
- Closed List: a record of nodes already expanded, preventing redundant work.
- Node Set: the complete set of grid cells (or graph vertices) under consideration.

At each iteration, A* removes the node with the lowest $F(n)$ from the open list, moves it to the closed list, and examines its neighbors. For each neighbor, the algorithm computes a tentative G cost (current node's G plus edge or traversal cost), updates H if necessary, and either inserts or reprioritizes the neighbor in the open list.

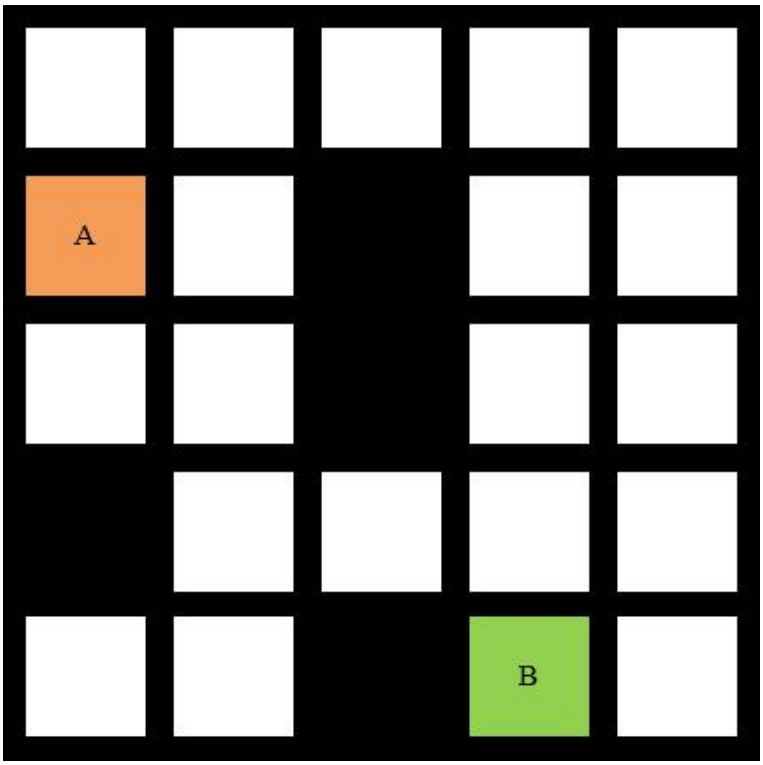


Figure 4- 1 Two-Dimensional Grid Map

Consider the two-dimensional grid of Figure 4-1, where each cell measures 10 units per side. The yellow cell marks the robot's start, the green cell marks the goal, and black cells denote obstacles. A* begins by placing the start node in the open list. Guided by the heuristic—often the straight-line (Euclidean) or Manhattan distance on a grid—it quickly homes in on a narrow corridor leading to the goal, exploring only those cells most likely to lie along the optimal path. Walls of obstacles cause local detours, but the algorithm's combined G and H evaluation ensures these detours remain minimal in both length and computational effort.

In practice, this approach finds the shortest feasible path on the grid with high efficiency, even in moderately large maps, making A* an ideal choice for our global

planning layer. By adjusting the grid resolution and heuristic weighting, we can further balance path optimality against planning speed to meet real-time requirements.

We begin by inserting node A and its eight surrounding neighbors into the open list, designating A as the parent of each. Node A is then removed from the open list and placed into the closed list, so any cell in the closed list will no longer be considered for expansion. In our cost model, horizontal and vertical moves incur a G(n) cost of 10, while diagonal moves incur a cost of 14. The heuristic H(n) is defined as the Manhattan distance (4-2) to the goal—i.e., the sum of the absolute differences in row and column indices between the current cell and the goal cell—scaled by 10 to match the cost units:

$$H(x) = |A_x - B_x| + |A_y - B_y| \tag{4-2}$$

$$H(x) = 10 \times (|A_x - B_x| + |A_y - B_y|)$$

Using this definition, we compute each neighbor's total cost $F(n) = G(n) + H(n)$ around A, as illustrated in Figure 4-2.

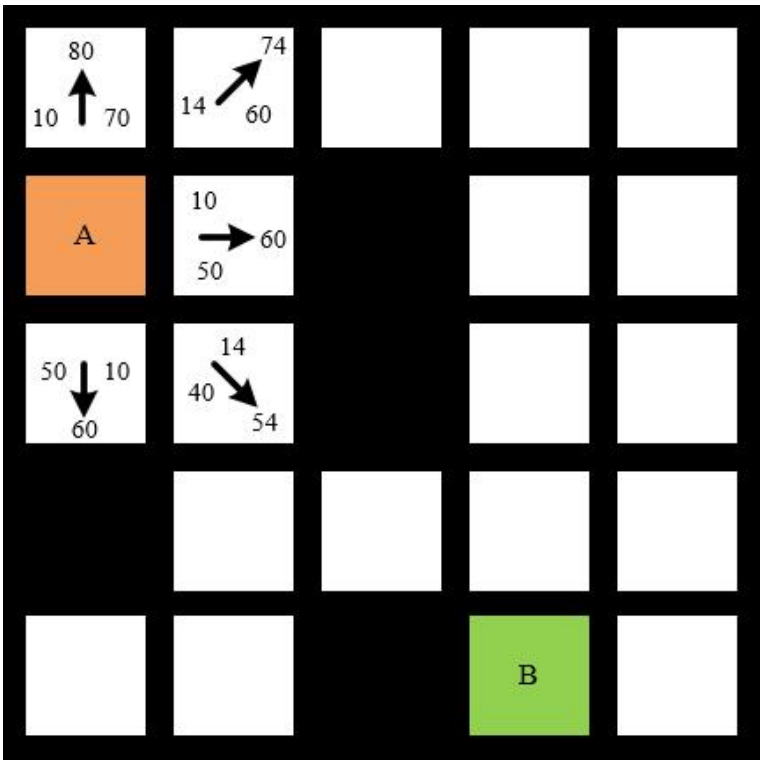


Figure 4- 2 Calculation of Grid Cell Cost Values

After evaluating all F-values in the open list, we select the neighbor with the lowest F—in this case, the cell diagonally down-right from A—as the next node to process. This node is moved from the open list to the closed list, and we then evaluate its own eight neighbors. Any neighbor that is an obstacle, out of bounds, or already in the closed list is skipped. For each remaining neighbor, if it already resides in the open list, we compute its tentative cost via the new parent and compare it with its existing cost: if the new cost is lower, we update the neighbor's parent pointer to the current node; otherwise, we leave it unchanged. If the neighbor is not yet in the open list, we insert it and set the current node as its parent.

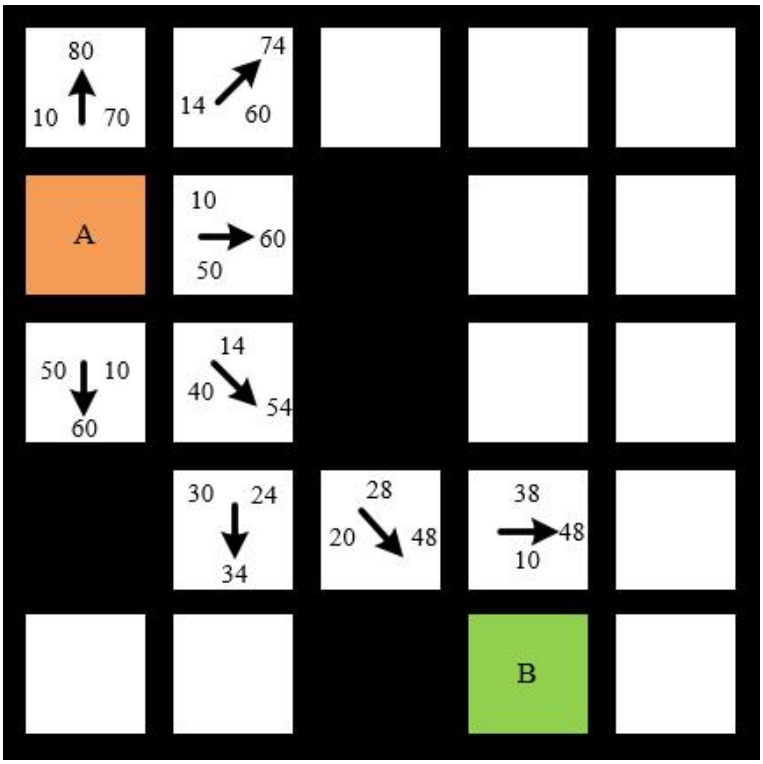


Figure 4- 3 Total Cost Values for All Grid Cells

This iterative expansion continues—always choosing the open-list node with the smallest F—until the goal cell itself is added to the open list, as shown in Figure 3-3. At that point, the optimal path is reconstructed by tracing backward from the goal through each node's parent pointer until the start node is reached. The resulting sequence of cells constitutes the planned route, as depicted in Figure 3-4. This procedure, guided by the Manhattan heuristic and our cost model, ensures that A* explores the most promising nodes first and produces an optimal collision-free path under static map conditions.

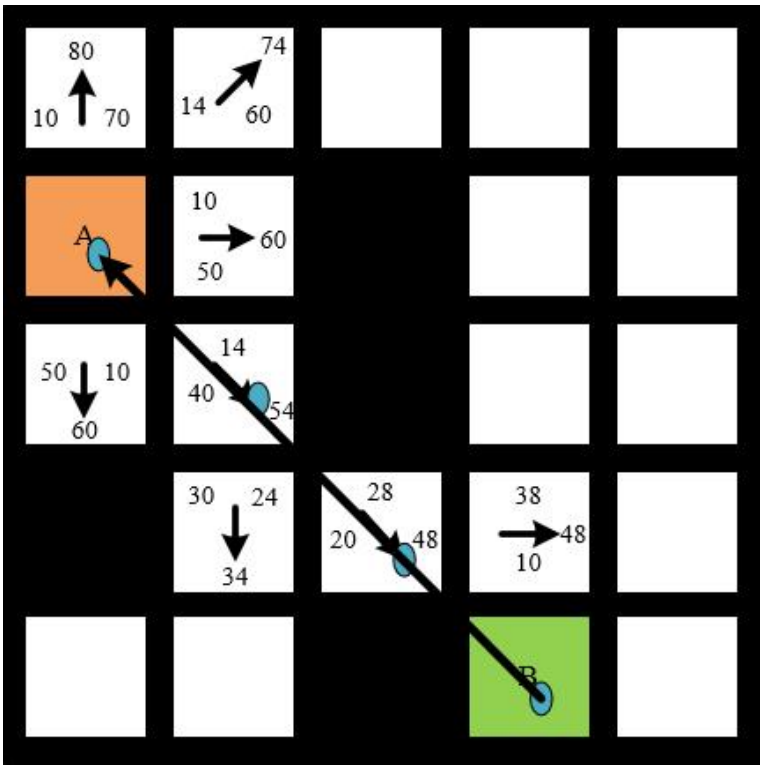


Figure 4- 4 Backward Path Diagram

4.2.2 Process of Global Path Planning Algorithm

In Figure 4-5, the A* search is laid out as a structured, decision-driven process with explicit checks and list-management steps. The algorithm proceeds as follows:

1. **Initialization**

At the top of the flowchart, we create two empty lists—Open and Closed—and immediately insert the start node into Open, initializing its total cost FF (typically $F = G + H = 0 + H$).

2. **Goal-Reached Check**

Before each expansion cycle, we first ask: “Has the goal node been moved into Closed?” If so, the search terminates successfully (path found).

3. **Exhaustion Check**

If the goal is not yet in Closed, we then check whether Open is empty. An empty Open list at this point indicates no viable path exists; the algorithm terminates in failure.

4. **Select and Promote Current Node**

When Open is nonempty, we scan its entries to find the node with the smallest F-value—this is our current node. We remove it from Open and add it to Closed, guaranteeing it will not be revisited.

5. **Neighbor Expansion**

From the current node, we examine all eight surrounding grid cells in turn:

- Skip Conditions: If a neighbor lies outside the grid bounds, is marked as an obstacle, or already resides in Closed, we ignore it.
- New Node Insertion: If the neighbor is not in Open, we compute its costs (G based on movement type: 10 for straight, 14 for diagonal; H via scaled Manhattan distance; and F=G+H, insert it into Open, and record the current node as its parent.
- Cost Refinement: If the neighbor already exists in Open, we compute a new tentative G′ via the current node and compare G′+H against its existing F. If the new cost is lower, we update the neighbor’s parent pointer to the current node and

adjust its F in the Open list priority queue.

6. **Loop Back**

After all neighbors are processed, control returns to step 2 (the goal-reached check), forming the main search loop. This cycle repeats, always promoting the most promising frontier node, until either the goal is closed or no nodes remain.

7. **Path Reconstruction**

Once the goal node appears in Closed (step 2), we exit the main loop and begin backtracking: starting from the goal, we follow each node's parent pointer in reverse until we reach the start node. This backward chain of grid cells constitutes the optimal path (Figure 4-4).

Throughout this process, the explicit decision diamonds and loop arrows in Figure 4-5 ensure that every branch—whether skipping invalid neighbors, inserting new ones, or refining existing costs—is handled deterministically. By tightly coupling the priority-driven expansion of Open with the exclusion semantics of Closed, A* guarantees both completeness (it finds a path if one exists) and optimality (it returns the least-cost path under static-map conditions). Because each iteration only examines a bounded subset of the grid (those nodes in Open), the algorithm remains efficient even on large maps, making it well suited for our global planning layer.

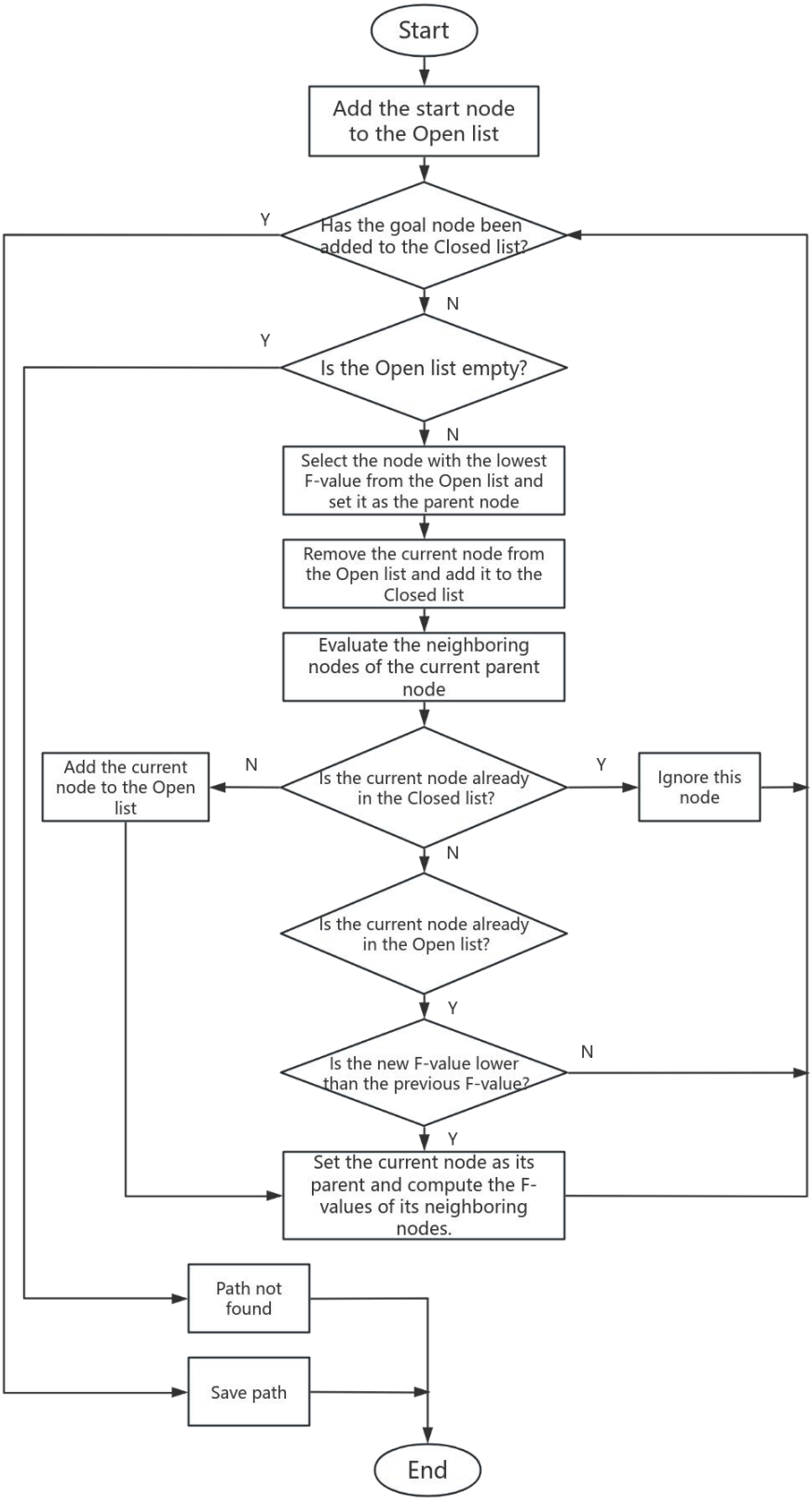


Figure 4- 5 A* Algorithm Flowchart

4.3 Local Path Planning Algorithms

Over decades of research into local path-planning methods, a variety of techniques—such as artificial potential fields and the Dynamic Window Approach (DWA)—have been proposed to enable safe, reactive navigation in dynamic environments. In this work, we adopt the rolling-window DWA for local path planning, a method first introduced in a 1997 IEEE publication. DWA is especially well suited to handling unpredictable obstacles: by sampling potential motion commands within a velocity-and-acceleration constrained window, it evaluates only those trajectories that are guaranteed to be dynamically feasible and collision-free.

One of DWA's primary strengths is its low computational burden. Because the algorithm limits its search to a short time horizon and considers only safe velocities and accelerations, the set of candidate trajectories remains small, allowing each control cycle to complete within strict real-time deadlines. This rolling-window framework continuously fuses fresh obstacle information—drawn from layered cost maps that combine static map data with live sensor readings—so that the robot can adjust its path on the fly as new hazards appear or disappear.

In practice, DWA balances three objectives within each planning interval: maintaining a safe clearance from obstacles, progressing toward the global goal, and optimizing the robot's forward velocity. By tuning parameters such as the sampling

resolution, control-cycle duration, and cost-function weights, designers can prioritize smoothness, speed, or safety depending on the application. DWA's responsiveness and respect for the platform's kinematic constraints make it an ideal local planner for resource-constrained robots operating in cluttered, fast-changing scenarios.

4.3.1 Principle of Local Path Planning Algorithm

Among the many local path-planning techniques, the ROS navigation stack implements the Dynamic Window Approach (DWA) for real-time trajectory refinement and obstacle avoidance. DWA operates by uniformly sampling the robot's admissible velocity space—bounded by its current translational and rotational velocities as well as its maximum acceleration and deceleration limits—and using each sampled velocity pair to predict a short-term motion trajectory on the local cost map.

For each candidate trajectory, a carefully designed evaluation (or scoring) function quantifies its suitability by combining multiple criteria:

- Clearance: the minimum distance from predicted path points to any obstacle, ensuring safety margins;
- Heading: the alignment of the trajectory's endpoint with the global goal direction, promoting progress toward the target;
- Velocity: the forward speed achieved, encouraging efficient travel.

By normalizing and weighting these metrics, DWA ranks all sampled trajectories and selects the one with the highest overall score as the next motion command. Because the sampling window is constrained to dynamically feasible velocities—those the robot can safely reach within one control cycle—the set of candidate paths remains small, enabling each planning iteration to complete within tight real-time deadlines (typically 10–20 Hz).

This rolling-window framework continuously integrates fresh obstacle data from the local cost map—layered with static map information, live LiDAR scans, and inflation buffers—so that the planner can react immediately to newly detected hazards without recomputing the entire global route. Compared to other local planners (e.g., artificial potential fields), DWA offers greater stability (avoiding oscillations and local minima) and predictable performance under dynamic conditions. Its respect for the robot's kinematic constraints and its low computational overhead make it an ideal choice for robust, responsive navigation in cluttered, unpredictable environments.

4.3.2 Process of Local Path Planning Algorithm

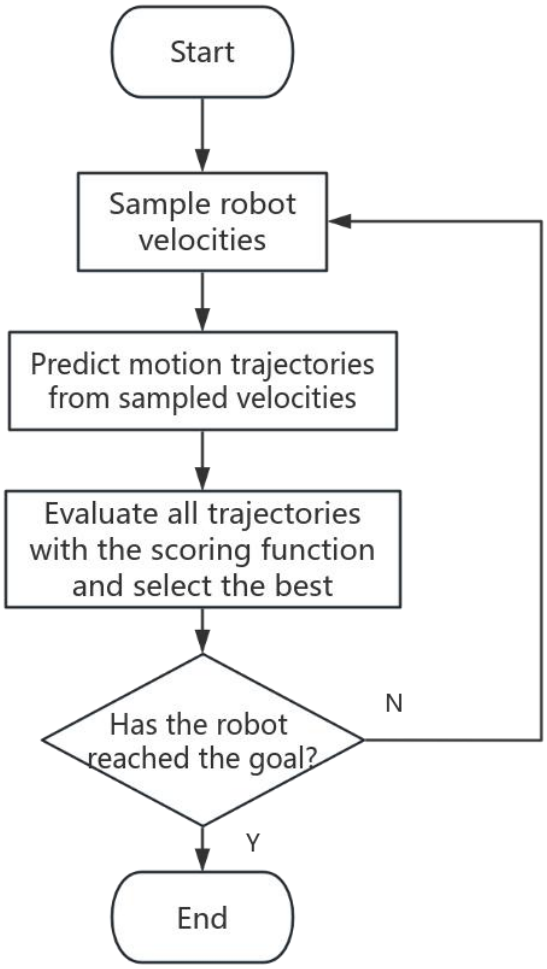


Figure 4- 6 DWA Algorithm Flowchart

Figure 3-6 illustrates the real-time control loop of the Dynamic Window Approach, which continuously refines the robot's trajectory based on live sensor data and kinematic constraints. The process unfolds as follows:

1. **Sampling Robot Velocities**

At the start of each cycle, the planner computes the robot's dynamic window by

intersecting its current velocities with its maximum acceleration and deceleration limits. Within this window, a finite set of linear and angular velocity pairs (v,ω)(v,\omega)(v,ω) is sampled. The sampling density and the number of candidates can be tuned to balance responsiveness against computational cost—denser sampling yields smoother paths but increases CPU load, while sparser sampling conserves resources at the expense of granularity.

2. **Predicting Motion Trajectories**

For each sampled velocity pair, the algorithm forward-simulates the robot's motion over a short time horizon Δt, using its discrete-time kinematic model. These predicted trajectories are overlaid onto the local cost map, which merges static map data with dynamic obstacle inflations. By simulating only a few timesteps ahead (typically 3–5), the planner limits computational effort while still capturing near-term collision risks and navigational progress.

3. **Scoring and Selecting the Optimal Path**

Each candidate trajectory is evaluated by a multi-criteria scoring function that combines:

- Obstacle Clearance: The minimum distance to any obstacle along the path, ensuring the robot maintains its safety buffer.
- Goal Alignment: How well the trajectory's endpoint points toward the global goal,

promoting efficient progress.

- Forward Velocity: The average or final speed achieved, encouraging faster traversal when safe.

Scores are normalized and weighted to reflect the application's priorities (e.g., safety vs. speed). The highest-scoring trajectory is chosen for execution, guaranteeing that at each step the robot moves along a path that best trades off these competing objectives.

4. **Goal-Reached and Exception Handling**

After selecting the optimal trajectory, the planner checks whether its endpoint falls within the goal region. If the goal is reached, the loop terminates and the robot stops or transitions to a subsequent task. If not, the planner proceeds to the next cycle. In cases where no collision-free trajectory exists—perhaps due to a sudden obstacle blocking all sampled paths—the algorithm triggers a safe-stop behavior and raises an alert to the Operator. Similarly, if sensor data fail or the local cost map becomes invalid, fallback routines reload the map or reset sensors before resuming planning.

5. **Real-Time Performance and Integration**

This planning loop typically runs at 10–20 Hz, allowing the robot to respond immediately to environmental changes. Its low computational overhead—achieved by constraining both the sampling window and the time horizon—makes DWA a natural

fit for embedded robotic platforms. In our ROS-based implementation, the DWA node subscribes to the global path topic, LiDAR scans, and TF transforms, then publishes velocity commands to the robot's driver interface. Diagnostic hooks measure execution time each cycle, ensuring the planner consistently meets its hard real-time deadline.

By iterating these steps continuously, DWA enables robust, collision-free navigation in cluttered and unpredictable environments, seamlessly integrating with the higher-level global planner to achieve both long-range efficiency and local reactive safety.

4.3.3 Velocity Sampling of Mobile Robot

Since this work is validated on a virtual simulation platform, we do not develop a new physical motion model; instead, we assume the robot follows a two-wheel differential-drive kinematic model. Once this model is established, the robot's future motion trajectories can be predicted directly from its commanded linear and angular velocities. The Dynamic Window Approach therefore begins by sampling a large set of (v, ω) pairs, but these samples must be constrained by the robot's own performance limits—this restriction is at the heart of DWA's dynamic window concept.

In our implementation, the mobile robot's forward speed v and rotational speed ω are bounded by minimum and maximum values, as expressed in equation (4-3):

$$V_m = \{v \in [v_{min}, v_{max}], \omega \in [\omega_{min}, \omega_{max}]\} \quad (4\text{-}3)$$

Where v_{min} and v_{max} are the minimum and maximum linear velocities, ω_{min} and ω_{max} are the minimum and maximum angular velocities.

These bounds reflect the robot's maximum achievable speeds as well as safety margins imposed by motor and chassis dynamics. During each control cycle, the planner uniformly samples velocities within these intervals—dividing each range into a fixed number of steps—to form the candidate set. This ensures that every sampled (v,ω) is both dynamically feasible (respecting acceleration and deceleration limits) and safe to execute within the upcoming time horizon. By limiting the search to this compact velocity window, the DWA algorithm maintains real-time performance while still exploring enough diversity in motion commands to navigate around obstacles and follow the global route effectively.

The robot's drive motors exhibit non-ideal behaviors—such as torque ripple, frictional losses, battery voltage sag, and control-loop latency—that directly influence how quickly the vehicle can accelerate or decelerate. To account for these real-world limitations, DWA constructs its dynamic window by intersecting the robot's static speed bounds with the acceleration-limited velocity interval achievable within a single control cycle Δt. This guarantees that every sampled (v,ω) pair is not only within the

robot's absolute speed limits but also reachable given its current motion state and motor capabilities. In effect, the dynamic window adapts at each time-step to the robot's instantaneous performance envelope, ensuring that only physically feasible trajectories enter the evaluation step. This acceleration-constrained sampling is expressed as:

$$V_d = \left\{ (v, \omega) \middle|_{\omega \in [\omega_c - \dot{\omega}_b \Delta t, \omega_c + \dot{\omega}_a \Delta t]}^{v \in [v_c - \dot{v}_b \Delta t, v_c + \dot{v}_a \Delta t]} \right\} \quad (4\text{-}4)$$

Where:

v_c、ω_c——the robot's current linear and angular velocities.;

$\dot{v}_b$、$\dot{v}_c$ —— the maximum linear deceleration and maximum linear acceleration.;

$\dot{\omega}_a$、$\dot{\omega}_c$ ——the maximum angular deceleration and maximum angular acceleration.

To guarantee that the vehicle can reliably avoid newly detected obstacles, the planner enforces a safety buffer around the robot's footprint. In practice, this means that when calculating candidate trajectories, we first determine the minimum predicted distance d_{min} to any obstacle along those paths. Given the control-cycle duration Δt and the robot's reaction time, the robot must not travel farther than this buffer in one cycle; otherwise, it risks breaching the safety margin before the next re-planning step can occur. As a result, the maximum allowable linear velocity v is dynamically capped

by:

$$V_a = \left\{(v,\omega)\middle| v \le \sqrt{2 \cdot dist(v,w) \cdot \dot{v}_b}, \omega \le \sqrt{2 \cdot dist(v,w) \cdot \dot{\omega}_b}\right\}) \qquad (4\text{-}5)$$

$$0 \le v \le \frac{d_{min}}{\Delta t}$$

where d_{min} is the closest distance from the planned trajectory to the nearest obstacle, and Δt is the duration of one planning-execution cycle.

By recomputing d_{min} at each iteration, the DWA algorithm automatically reduces the velocity limit when navigating tight or cluttered environments—thereby providing the robot with sufficient distance and time to decelerate or maneuver around hazards. In open spaces with few obstacles, d_{min} grows larger, allowing higher speeds and more efficient travel. This adaptive speed bound balances safety (maintaining the required buffer) against efficiency (maximizing speed when conditions permit). Moreover, by coupling this constraint with the acceleration-bounded dynamic window (Equation 4-4), the planner ensures that every sampled velocity is both feasible—respecting motor and chassis limits—and safe—respecting the real-time obstacle buffer. Continuous empirical tuning of Δt and safety-margin parameters across different scenarios further refines performance, yielding robust collision-free navigation under dynamic, unpredictable conditions.

4.3.4 Trajectory Evaluation Function

Once all candidate trajectories have been generated, each is scored by a unified evaluation function designed to balance three competing objectives—progress toward the goal, safety clearance, and forward speed—and then normalized to prevent any single metric from dominating. Concretely, for each sampled velocity pair (v, ω) we compute:

$$G(v, \omega) = \sigma(\alpha \cdot heading(v, \omega) + \beta \cdot dist(v, w) + \gamma \cdot velocity(v, w)) \quad (4\text{-}6)$$

$heading(v, \omega)$ ——measures the angular alignment between the trajectory's endpoint and the global goal, encouraging paths that maintain forward progress;

$dist(v, w)$ ——is the minimum Euclidean distance from any point on the predicted trajectory to the nearest obstacle, enforcing a safety buffer;

$velocity(v, w)$ ——captures the average or terminal forward speed of the trajectory, promoting efficient motion；

α、β、γ——are designer-tunable weights that reflect mission priorities (for example, setting $\alpha \gg \beta \gg \gamma$ in highly cluttered environments to prioritize safety over speed);；

σ——denotes a normalization operator, which may be as simple as dividing by the sum across all candidates or as sophisticated as applying a smooth nonlinearity (e.g,

a logistic map) to amplify top-scoring paths.

To ensure that heading, distance, and velocity terms are commensurate, we first normalize each raw metric across the full set of n candidates:

$$\text{Normal\_}h\text{ead(i)} = \frac{h\text{ead(i)}}{\sum_{\text{i=1}}^{\text{n}} h\text{ead(i)}} \quad (4\text{-}7)$$

$$\text{Normal\_dist(i)} = \frac{\text{dist(i)}}{\sum_{\text{i=1}}^{\text{n}} \text{dist(i)}} \quad (4\text{-}8)$$

$$\text{Normal\_velocity(i)} = \frac{\text{velocity(i)}}{\sum_{\text{i=1}}^{\text{n}} \text{velocity(i)}} \quad (4\text{-}9)$$

This normalization confines each component to the range [0,1], preventing skew from large variance in raw units and ensuring that the weighted sum fairly reflects the intended trade-offs. In practice, we tune α、β、γ through a combination of simulation-based parameter sweeps and hardware-in-the-loop experiments: for example, increasing β when operating in narrow corridors to widen the effective clearance margin, or boosting γ on long, obstacle-scarce segments to improve average speed.

Once weighted and normalized, the σ-processed score $G(v, \omega)$ provides a single scalar by which all trajectories can be ranked. The planner then selects the (v, ω) pair with the highest score, guaranteeing that each cycle's motion command optimally balances safety, goal alignment, and velocity under the current environmental and kinematic constraints. This selection mechanism is recomputed at

every control cycle(typically at 10–20 Hz)so that the robot continuously adapts to new obstacle data and evolving mission objectives.

By embedding this trajectory evaluation function within the DWA loop, our system not only achieves real-time responsiveness and collision-free safety, but also maintains predictable performance: the use of strongly typed ROS messages for cost-map and trajectory exchanges, combined with built-in diagnostic timers, confirms that each scoring-and-selection step stays within its allotted time budget. As a result, the robot can navigate complex, dynamic environments with a proven, quantitatively tunable mechanism for balancing speed, safety, and efficiency at every instant.

This chapter primarily explains path-planning algorithms from two directions. Global path planning mainly uses the A* algorithm: it employs a two-dimensional grid map as an example to demonstrate its theoretical foundation and draws a flowchart of the A* algorithm's principle. Local path planning mainly uses the DWA algorithm: it first analyzes the algorithm's principle, then draws a flowchart of that principle. The vehicle's velocities are sampled, motion trajectories are inferred from those velocities, an evaluation function is used to assess trajectory quality, and finally normalization is applied to make the trajectory function smoother.

Chapter 5 Simulation Experiments on Path Planning

5.1 Overview of the ROS Simulation Environment

Robot Operating System (ROS) is a flexible and powerful open-source software framework widely used in the field of robotics for building complex and robust robot applications. It acts not only as a middleware but also provides a full-fledged operating system for robot development. ROS integrates a variety of essential components such as terminal commands, toolkits, open-source code packages, and communication mechanisms that facilitate robot perception, planning, control, and simulation. Thanks to its modular and distributed architecture, ROS allows researchers and developers to divide a robotic system into independent functional units that can communicate and cooperate with one another, which is especially suitable for large-scale and multi-node robotics projects.

ROS is highly adaptable in terms of programming languages. It supports C++, Python, and even Java, enabling users from different technical backgrounds to contribute and integrate their work. The flexible programming environment and a rich ecosystem have led to a vibrant global community and a vast number of reusable packages. Additionally, ROS includes visualization and simulation tools such as Rviz for real-time graphical rendering and Gazebo for high-fidelity 3D physical simulation, allowing developers to validate their algorithms in virtual environments before

deploying them to real hardware.

From a system architecture perspective, ROS is composed of three core layers: the community layer, the filesystem layer, and the computational graph layer. The ROS Filesystem Layer (as shown in Figure 5-1) forms the foundation of code organization. The fundamental unit of the ROS filesystem is the package. A package may contain executable nodes, scripts, configuration files, launch files, as well as message definitions (msg), service types (srv), and metadata such as the package manifest. Message types are often used in topic-based communication to define the structure of data exchanged between nodes. Similarly, service types are used in service communication to define request-response message structures.

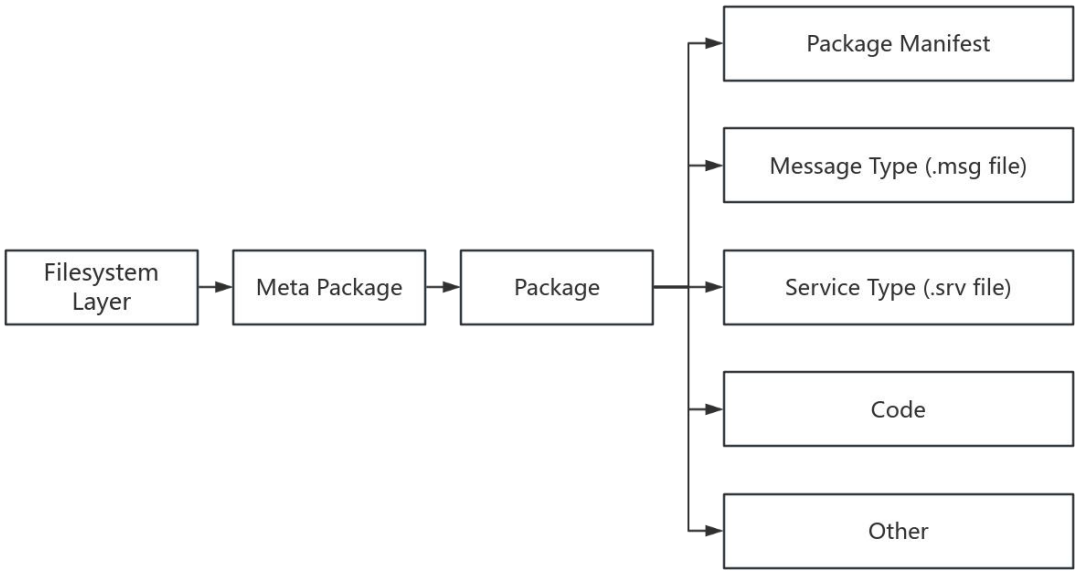


Figure 5- 1 ROS Filesystem Layer Structure

The ROS Computational Graph Layer (illustrated in Figure 5-2) is the core

runtime framework that handles data flow and execution logic. It represents how nodes interact during program execution. Each process in ROS is abstracted as a node, and these nodes communicate with each other through publishing or subscribing to topics, or calling services. The Master node acts as the registrar and central coordinator that tracks the presence and location of nodes and facilitates their interconnection. Additionally, a Parameter Server is available to store and manage configuration data that can be accessed globally by any node. Topics provide a one-way communication mechanism, suitable for continuous data streams like sensor readings, while services support synchronous two-way interactions, typically used for request-response patterns. It is important to note that within a single ROS network, each node must have a unique name to avoid communication conflicts.

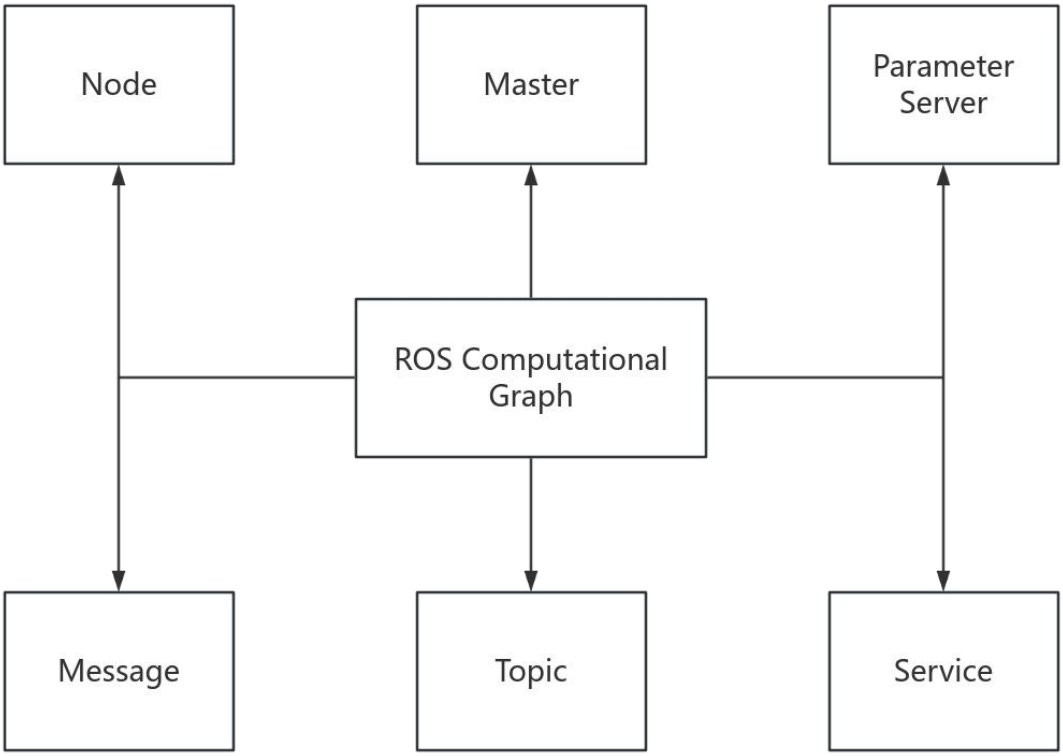


Figure 5- 2 ROS Computational Graph

By combining these two layers—filesystem and computational graph—ROS provides a comprehensive and highly structured environment for simulation and experimentation. This is particularly important in this research, where various simulation experiments are conducted using ROS tools to validate the performance of the proposed path planning algorithms under both static and dynamic conditions. The simulation environment lays the groundwork for subsequent experiments in mapping, localization, and adaptive path planning as detailed in the following sections.

5.1.1 STDR Simulator

STDR, short for Simple Two Dimensional Robot Simulator, is a lightweight ROS-compatible simulation platform designed specifically for simulating robot movement and sensor interactions in a 2D environment. Unlike high-complexity simulators like Gazebo, STDR focuses on simplicity and rapid deployment, making it highly suitable for validating fundamental navigation algorithms, testing sensor models, and conducting educational experiments in path planning.

This simulator supports not only single-robot simulations but also multi-robot coordination scenarios. Each robot in the STDR environment can be equipped with simulated sensors such as LiDAR, sonar, and odometry sources. The data generated from these sensors can be published using standard ROS communication mechanisms—namely, topics (/scan, /odom, etc.) and services—thereby ensuring seamless integration with other ROS packages.

In this study, STDR is used to simulate a mobile robot navigating within a two-dimensional environment. The robot is equipped with a virtual laser scanner to capture environmental data. As the robot moves, the collected data is processed and sent to the mapping algorithm. The mapping result, which forms the static occupancy grid, is visualized in real time using Rviz, ROS's built-in visualization tool.

The simulation process includes launching the STDR world, initializing robot

parameters, and issuing motion commands to the robot. During motion, Rviz provides a dynamic visual representation of both the robot's trajectory and the gradually constructed static map. This allows for immediate verification of sensor feedback and mapping performance.

STDR offers several advantages: it is lightweight, easy to configure, and well-integrated with core ROS functionalities. It serves as an effective tool in the early development stages of robot navigation systems, particularly when 3D environments are not required. In this experiment, STDR was utilized to validate the correctness and efficiency of the Gmapping algorithm for static map construction. The interface of the STDR simulation environment is shown in Figure 4-3.

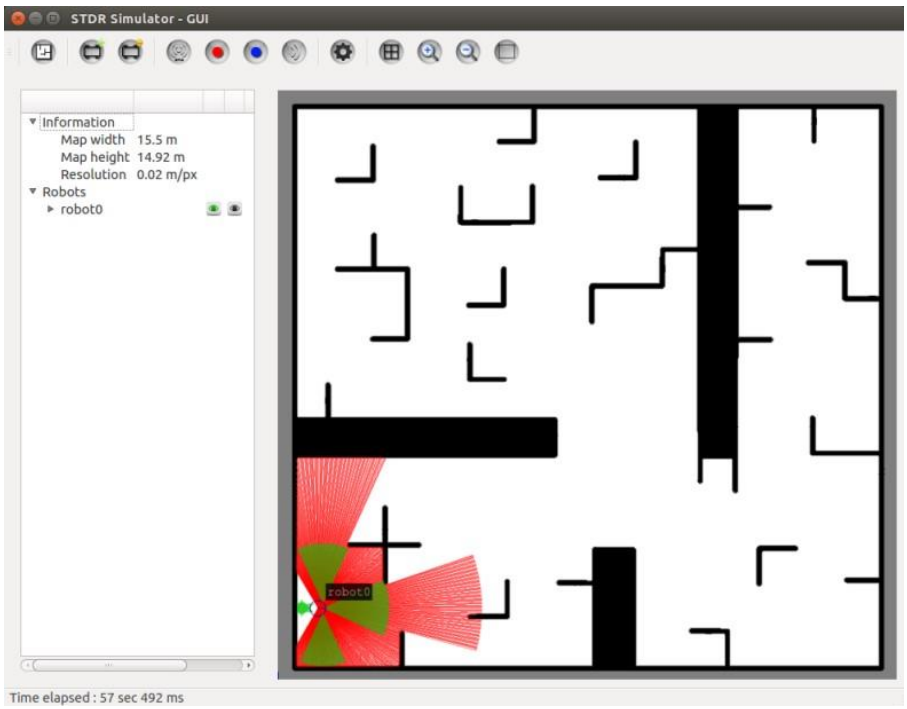


Figure 5- 3 STDR Simulation Interface

5.1.2 Rviz Visualization Tool

Rviz (short for ROS Visualization) is a powerful 3D visualization tool included in the ROS ecosystem. It serves as a critical interface for observing and interpreting sensor data, robot states, and environmental information in both real-time and recorded scenarios. Rviz plays an essential role in debugging, validating algorithms, and monitoring robotic behavior during simulation and actual deployment.

The user interface of Rviz is organized into several key sections. The left panel functions as the display configuration area. In this section, users can add various display elements such as laser scans, point clouds, robot models, map data, and navigation paths. For each element, properties like topic name, color, and style can be customized. As long as the desired ROS topics are being published, Rviz will subscribe to those topics and automatically render the corresponding data on the screen.

The central area of the interface is a real-time 3D rendering window, where visual representations of sensor data, robot geometry, coordinate frames, and planned trajectories are displayed. This visualization provides an intuitive and interactive way to inspect spatial relationships and motion behavior from multiple viewpoints, which is especially valuable when tuning navigation or localization algorithms.

On the right side, the global options panel allows users to configure general settings such as fixed frame (e.g., /map, /odom, or /base_link), background color,

frame rate, and visualization grid scale. These options help tailor the visualization environment to specific project needs.

In this research, Rviz is used in conjunction with other simulation tools such as STDR and Gazebo. It displays key outputs from the robot's sensors and planning system, including real-time LiDAR scans, AMCL localization results, generated costmaps, and the paths planned by the global and local planners. An example of the Rviz interface used in our experiment is shown in Figure 5-4.

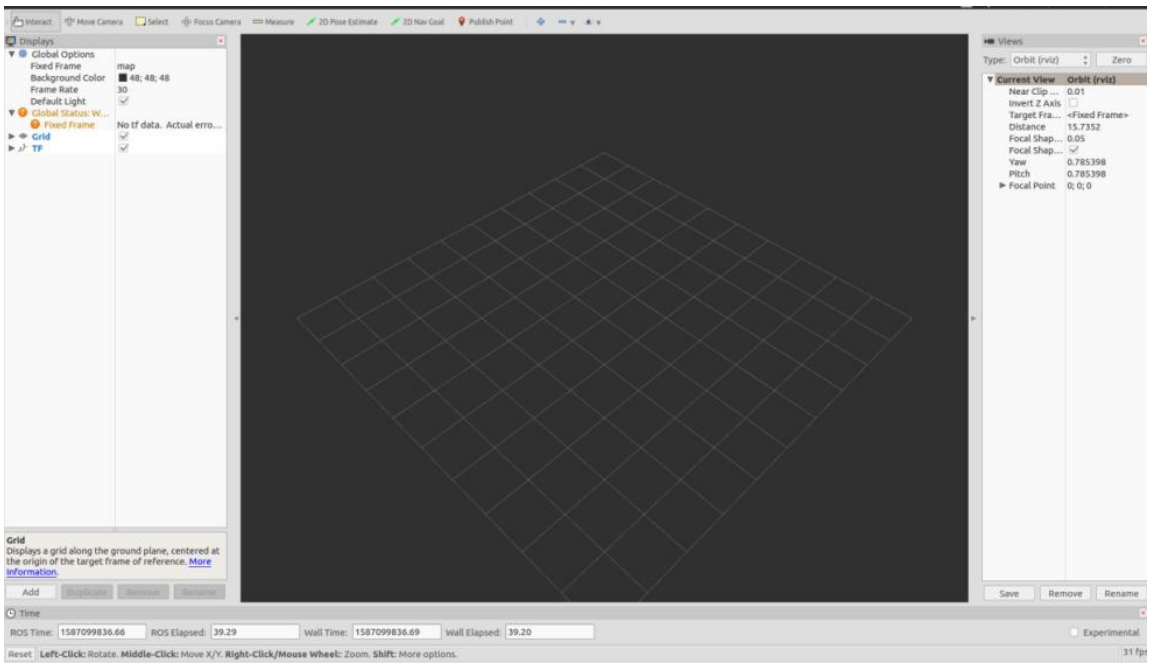


Figure 5- 4 Rviz Visualization Interface

5.1.3 Gazebo Simulator

Gazebo is a widely used 3D simulation platform for robotic systems, capable of replicating realistic environments and physics-based interactions. It provides a highly

customizable and extensible environment where virtual robots can be tested under various conditions. Gazebo enables users to import a rich library of models—such as buildings, terrains, and obstacles—and to assign physical properties to these models, including gravity, friction, and mass. This helps create environments that closely mimic real-world dynamics, making simulations more meaningful and effective.

One of Gazebo's key advantages is its extensive sensor library, which includes LiDARs, depth cameras, RGB-D sensors, GPS modules, and inertial measurement units (IMUs). These sensors can be attached to simulated robots and configured to replicate real sensing behaviors. As a result, Gazebo allows for accurate testing of perception, localization, and navigation algorithms. The collected sensor data is published as ROS topics, seamlessly integrating with other tools like Rviz for visualization or Move Base for path planning.

Unlike simple 2D simulations, 3D simulation in Gazebo provides a rich and immersive testing environment. Its value lies not only in visual realism but also in replicating complex physics interactions. For instance, in autonomous driving research, real-world road testing can be expensive, risky, or impractical. Gazebo fills this gap by simulating dynamic and unpredictable road conditions, enabling researchers to test autonomous navigation algorithms in a safe and controllable manner. Environmental factors such as uneven terrain, obstacles, sensor noise, or lighting conditions can be

replicated in simulation to validate system robustness.

Under the ROS framework, Gazebo can be launched using predefined launch files such as gazebo_ros empty_world, which opens a blank simulation environment. From there, users can import and arrange elements from the object library to construct customized worlds. These environments can include urban streets, indoor rooms, industrial layouts, or outdoor terrains—depending on the testing needs.

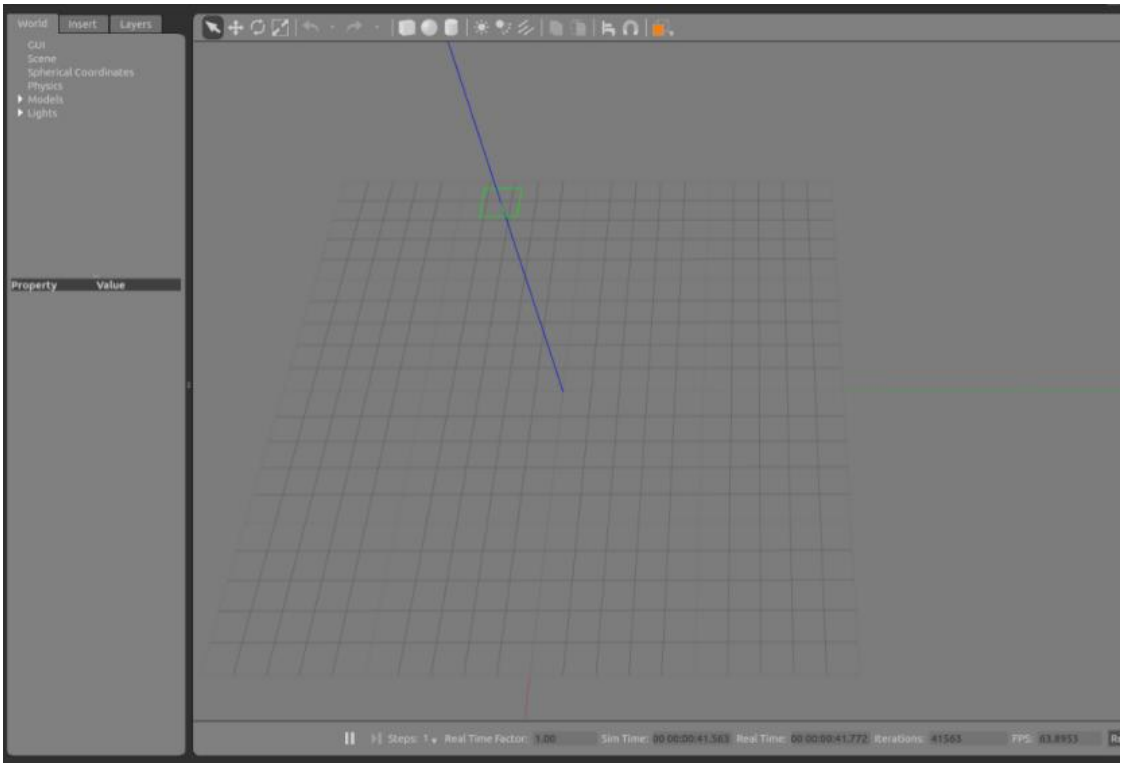


Figure 5- 5 Gazebo Simulation Interface

Figure 5-5 shows the user interface of the Gazebo simulator as used in this research. This setup was employed to evaluate the performance of our adaptive path planning algorithm under realistic 3D environmental conditions.

5.2 Simulation Results

To successfully implement path planning for autonomous robots on a simulation platform, it is essential to define a clear and structured system design strategy. Before diving into actual implementation, we need to thoroughly understand what components are necessary for a functional path planning system and how they interact with one another.

Based on literature review and practical considerations, we summarize that three core prerequisites must be satisfied to achieve effective path planning:

- A high-resolution static map that accurately represents the surrounding environment;
- A costmap that dynamically reflects environmental factors to assist in obstacle avoidance;
- A localization system that provides real-time positional information of the robot and the destination point on the map.

In this study, we implement path planning on a static occupancy grid map and an updated dynamic costmap. The local path planning component can actively perceive the robot's surrounding environment and adjust its trajectory accordingly. However, the static map generated at the beginning contains fixed obstacles and cannot adapt to dynamic changes in the environment. In real-world scenarios, especially for

autonomous vehicles navigating urban roads, dynamic obstacles such as pedestrians, cyclists, and moving vehicles are frequently encountered.

To better simulate such conditions and improve the realism of our experiments, we introduce a 3D simulation environment. During the simulated car's path planning process, dynamic obstacles are deliberately added along the path to emulate real-world challenges. This setup allows us to test the robustness and adaptability of the planning algorithm in complex scenarios, and to observe and evaluate the path planning results more comprehensively.

With the overall design objectives clarified, the next step involves a detailed understanding of each subsystem—such as mapping, localization, and path planning—and the tools required to implement and validate them. We employ Gmapping for building the static map, and the costmap layer for generating dynamic obstacle information. For localization, we use AMCL (Adaptive Monte Carlo Localization), which enables accurate tracking of the robot’s position. The global path is planned using an A*-based algorithm, while the local path is generated through the DWA (Dynamic Window Approach) algorithm.

The system is validated across various simulation tools including the STDR simulator, Rviz for real-time visualization, and Gazebo for immersive 3D simulation. Each of these platforms plays a specific role: STDR is used for lightweight testing,

Rviz provides real-time sensor feedback and mapping visualization, and Gazebo is used to replicate real-world physics and interactions.

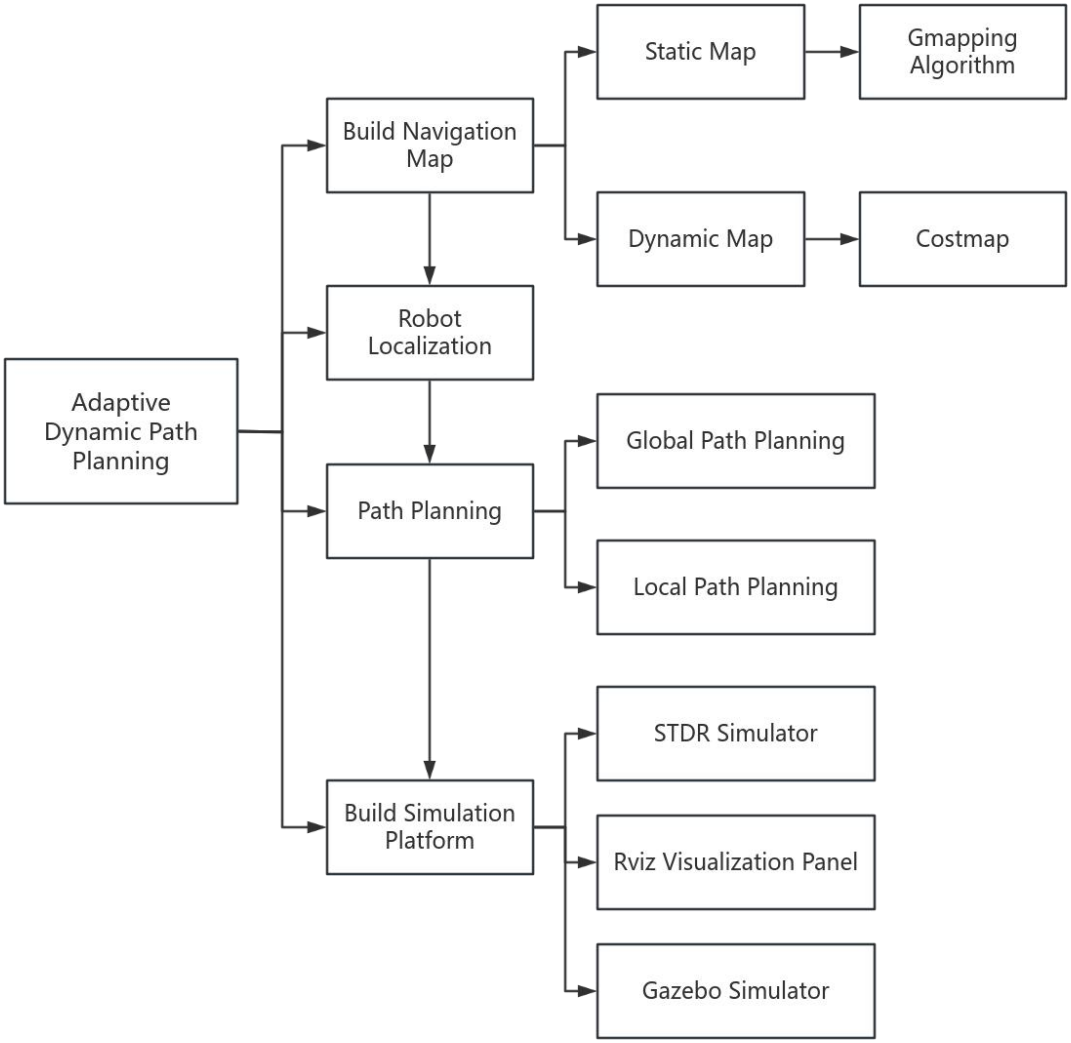


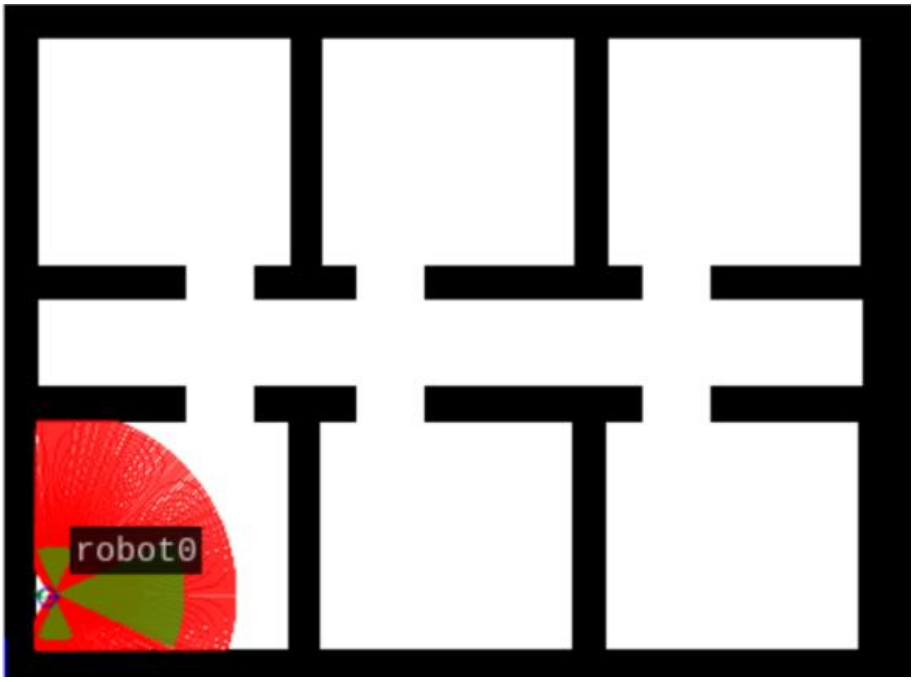
Figure 5- 6 Overall System Design Flowchart

The overall design flow of our system is illustrated in Figure 5-6, where each module from environment modeling to algorithm execution and system validation is systematically connected.

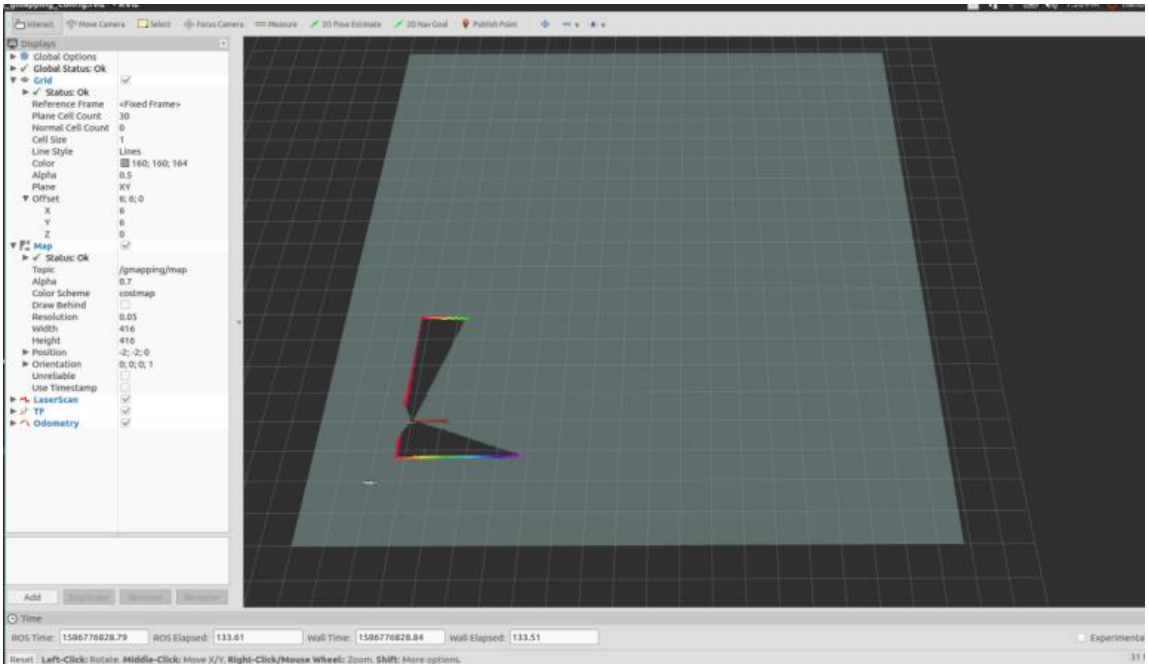
In Chapters 2 and 4, the theoretical foundations for achieving path planning were discussed in detail. This chapter focuses on the implementation aspect by establishing a virtual simulation platform. Within this simulated environment, experiments are conducted to analyze map construction, localization, and path planning. The simulation is divided into five stages: the first three stages focus on preparatory tasks for path planning, while the last two simulate path planning in different simulation platforms. Finally, the simulation results are analyzed to verify the feasibility and effectiveness of the proposed algorithms.

5.2.1 Gmapping Static Map Experiment

To construct a static occupancy grid map for the robot's navigation, the first step involves launching the stdr_gmapping.launch file, as illustrated in Figure 5-7. Figures (a) and (b) show the simulation interface in STDR and the visualization interface in Rviz respectively.



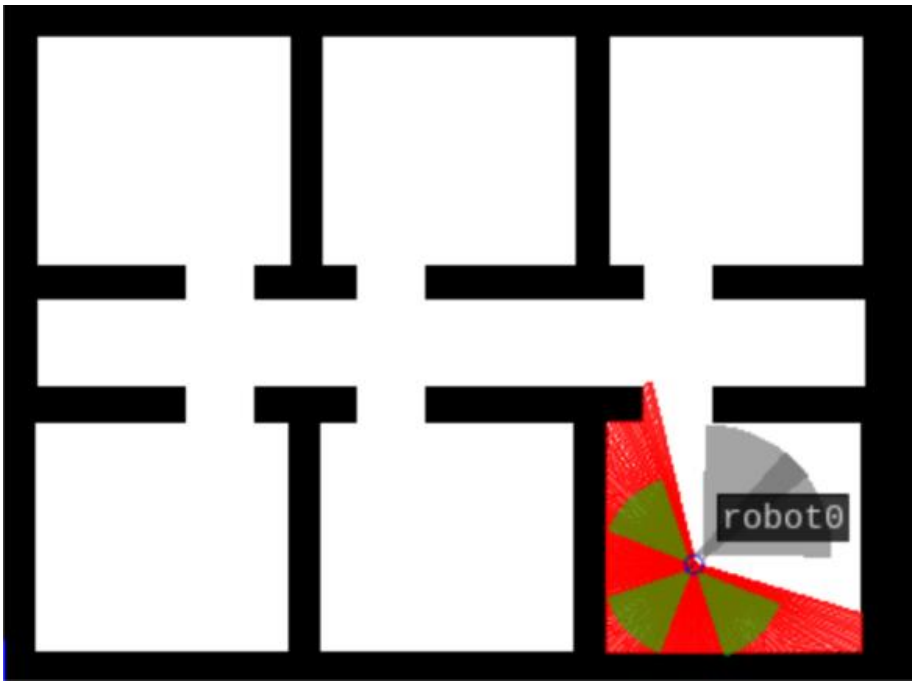
(a) STDR Simulator



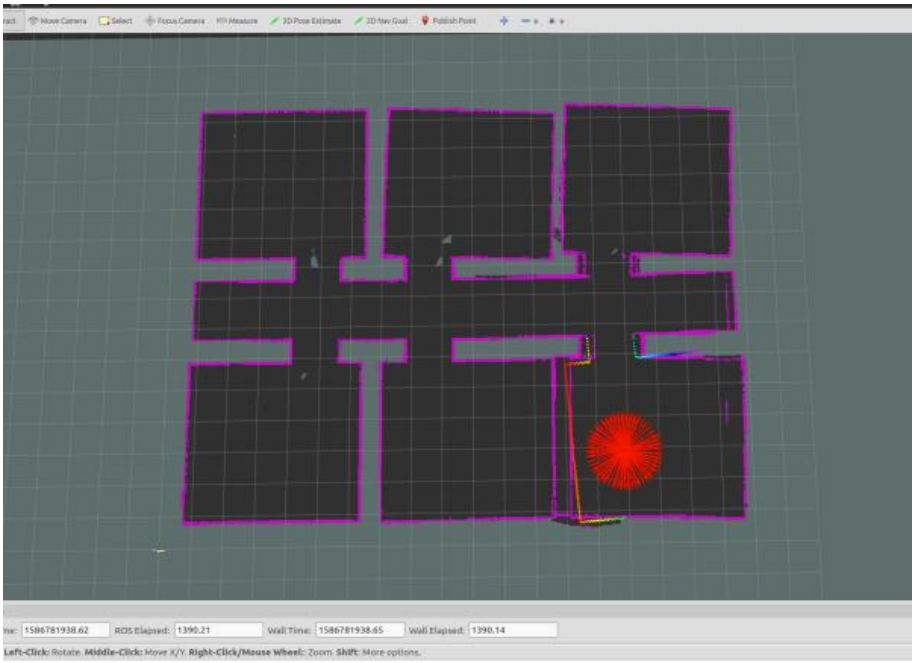
(b) Rviz Visualization Tool

Figure 5- 7 Gmapping Build Map

The bottom left corner in STDR is the initial position of the simulation robot, the red sector is the measurement range of LIDAR, and the green sector is the measurement range of ultrasonic. The simulation robot is controlled by the keyboard to move on the map, the LiDAR keeps scanning, and the result of the scanned map is displayed in Rviz as shown in Figures 5-8.Among them, Figures (a) and (b) are the images in the pages in STDR and Rviz, respectively.



(a) STDR Simulator



(b) Rviz Visualization Tool

Figure 5- 8 Gmapping Build Map results

Once the static map has been successfully generated, the map_server tool is used to save the map data for future use in costmap generation and path planning experiments. The saved map is displayed in Figure 5-9.

Although the constructed map captures most of the spatial features of the environment, there are still observable imperfections. For instance, some areas appear distorted or poorly aligned—particularly in the bottom right corner, where two segments of the environment connect improperly, resulting in skewed boundaries. These mapping artifacts are primarily caused by the accumulation of localization errors during the exploration process. In certain cases, particles with accurate weights might have been incorrectly eliminated, reducing the robustness of the SLAM estimation.

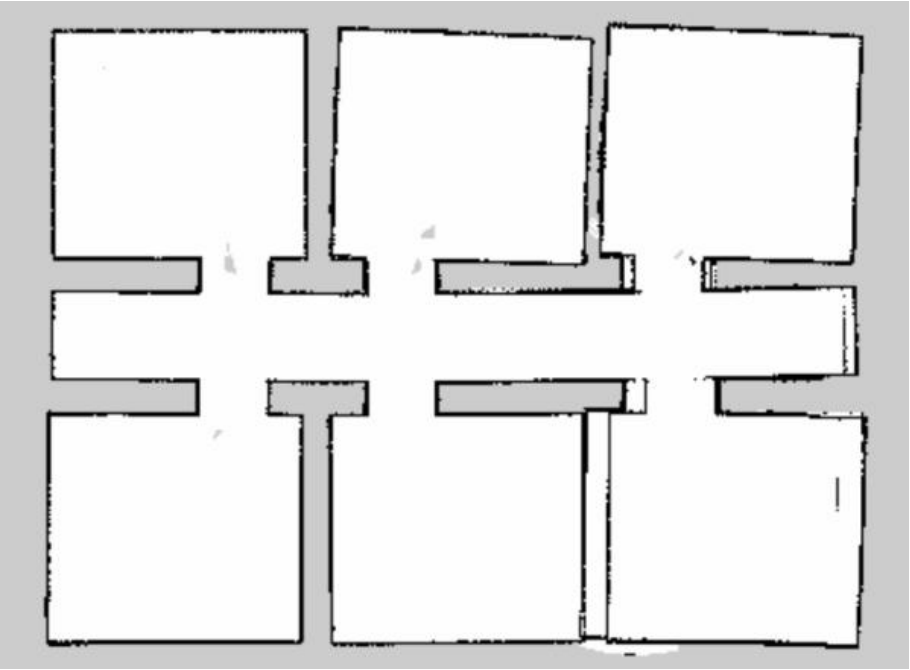


Figure 5- 9 Final Static Map Result

Nonetheless, despite these minor inaccuracies, the overall structure of the generated static map is acceptable for the purposes of simulation-based path planning. It provides a sufficiently detailed representation of the environment to support subsequent costmap construction and global route calculation.

5.2.2 Local map Generation Experiment

In a path planning system, the costmap plays a critical role by bridging raw map data and obstacle avoidance strategies. It is an essential module for enabling autonomous navigation. In this experiment, based on the previously constructed static map, the stdr_costmap.launch file is used to initiate relevant ROS nodes that generate both global and local costmaps. Through integration with the move_base module, the

costmap incorporates an obstacle layer and an inflation layer, enhancing the robot's spatial awareness and safety. As shown in Figure 5-10, the costmap visually indicates the location of obstacles and their surrounding influence zones on path planning.

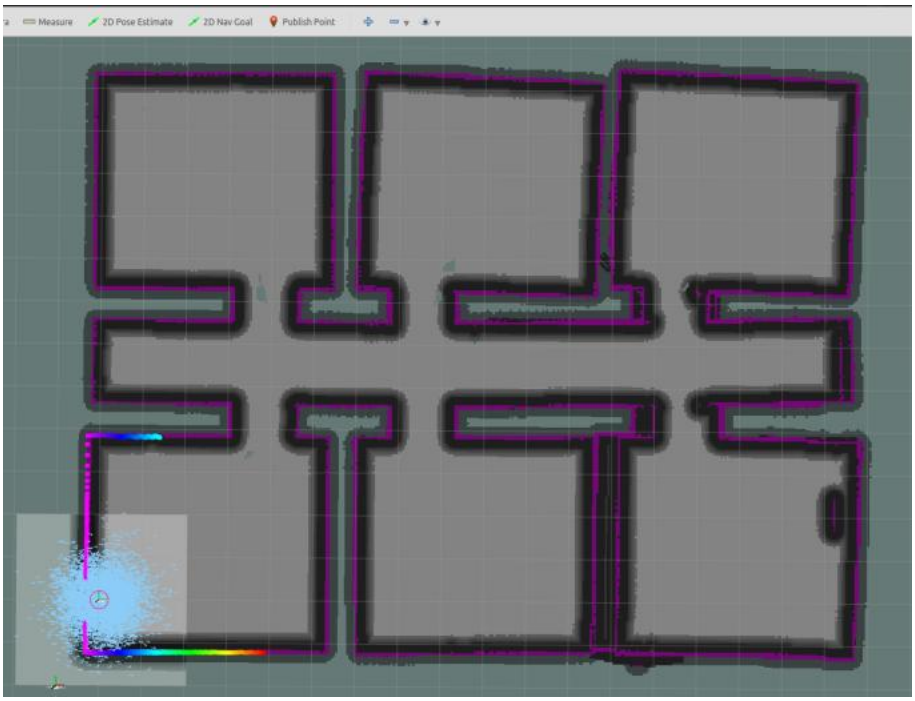


Figure 5- 10 Cost map

The global costmap overlays an additional layer onto the static map, marking a safety buffer zone around detected obstacles. This layer provides constraints for the global path planner, ensuring the generated paths avoid high-risk areas and remain smooth and collision-free.

On the other hand, the local costmap emphasizes real-time responsiveness and flexibility. Centered around the robot, it defines a rolling window that constantly

updates with environmental data from onboard sensors, such as LiDAR. The window size is configurable via parameters such as width and height. Within this window, the system continuously perceives and integrates new obstacle information to allow for dynamic trajectory adjustment.

In the RViz visualization interface, the costmap is presented using grayscale intensity:

- Purple areas indicate known obstacle cells.
- Gray areas represent traversable space.
- Darker shades correlate with higher cost values, typically indicating regions near obstacles or areas deemed unsafe.
- Lighter shades suggest lower risk and easier navigation zones.

Importantly, the costmap not only provides the basis for path generation but also acts as a safeguard for safe robot operation in dynamic environments. As the robot navigates, the system adjusts its trajectory in real time based on the costmap updates, enabling a "safety-first" navigation strategy. Even in complex scenarios involving occlusions or sudden obstacle appearance, the costmap helps the robot quickly re-plan to avoid collisions.

Overall, this experiment successfully demonstrates the integration of static and

dynamic environmental information into a unified map representation. It lays a solid foundation for subsequent validation of the path planning algorithms. Moreover, it highlights that the accuracy and update rate of the costmap significantly affect the overall performance of the navigation system. In the future, adaptive parameter tuning for the inflation layer could be explored to further enhance robustness in more complex scenarios.

5.2.3 Localization Experiment

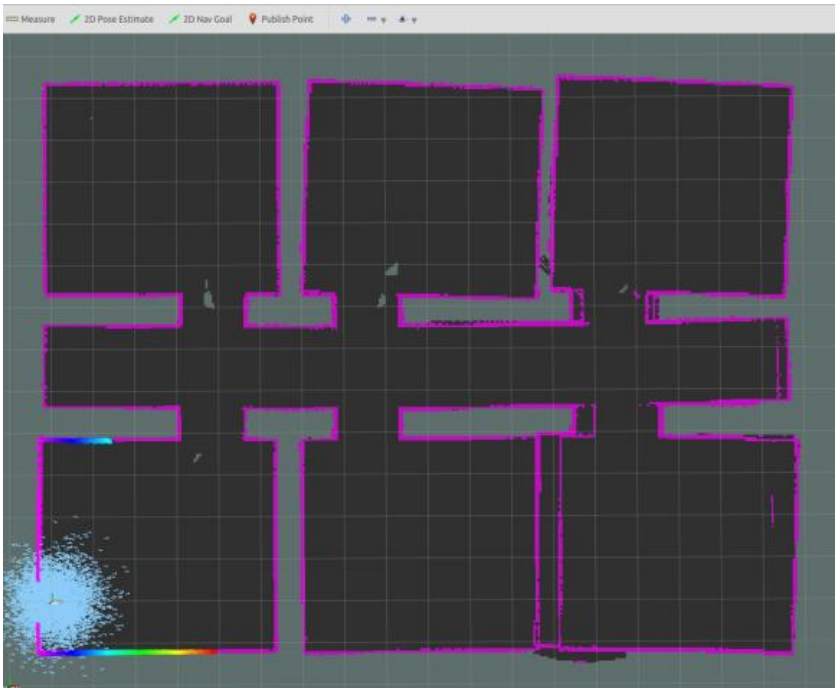
This experiment aims to verify the effectiveness of the AMCL (Adaptive Monte Carlo Localization) algorithm in determining the robot's position within a simulated environment. AMCL is a widely used particle filter-based localization method that estimates a robot's pose (position and orientation) on a known map. It continuously updates its estimation by integrating sensor data (such as LiDAR) and a motion model, refining the estimated pose as the robot moves.

To begin the experiment, we first launch the stdr_amcl.launch file, which initializes the AMCL node along with the required map server and TF transformation system. This setup ensures the spatial relationships between various coordinate frames (e.g., /map, /odom, /base_link) are maintained accurately. Once launched, we observe the robot's model in the Rviz visualization tool, surrounded by a large number of gray particles randomly distributed in the environment, as shown in Figure 5-11(a). These

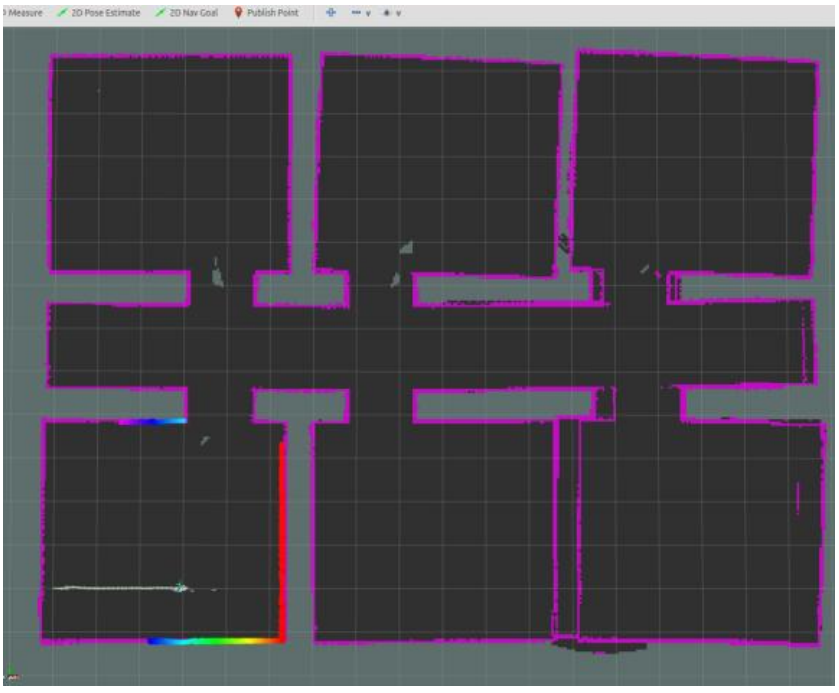
particles represent the robot's possible positions before localization is established.

Next, we execute the teleop_twist_keyboard.py script, which allows manual control of the robot's movement via keyboard inputs. As the robot moves, the AMCL algorithm begins to update the distribution and weights of the particles based on both the motion model and LiDAR scan data. Particles that are more consistent with the real sensor readings receive higher weights, while less accurate ones are discarded during the resampling process. Over time, the particle cloud begins to converge toward the robot’s true location, and the number of particles gradually decreases—demonstrating AMCL’s adaptive nature: it increases particle count in uncertain states and reduces it when confidence is high.

After a brief period of motion, the system successfully estimates the robot's pose. The particles stabilize and concentrate around the actual robot position, as shown in Figure 5-11(b). This convergence indicates accurate localization. Additionally, a stable TF transform from the /map frame to the /base_link frame confirms that the robot’s pose has been correctly determined.



(a) Initial stage



（b）Final stage

Figure 5- 11 AMCL Localization Effect Diagram

This experiment demonstrates that AMCL effectively fuses motion and sensor data to perform robust and precise localization within a simulated environment. Accurate pose estimation provided by AMCL serves as a foundational component for subsequent tasks such as global and local path planning, making it a critical step in autonomous navigation systems.

5.2.4 Path Planning in STDR Simulator

In this research, path planning experiments were conducted in both the two-dimensional STDR simulator and the three-dimensional Gazebo simulator to verify the feasibility and effectiveness of the proposed planning strategy under different simulation conditions. This section focuses on the experiment conducted within the STDR simulator, which provides a lightweight and efficient environment suitable for quick algorithm testing and path planning validation.

To begin the experiment, the stdr_move_base.launch file was executed, which automatically initializes the STDR simulator and RViz visualization tool. These two interfaces work in coordination to display the robot's current status and planned trajectory in both a simulated physical layout and a real-time visual environment.

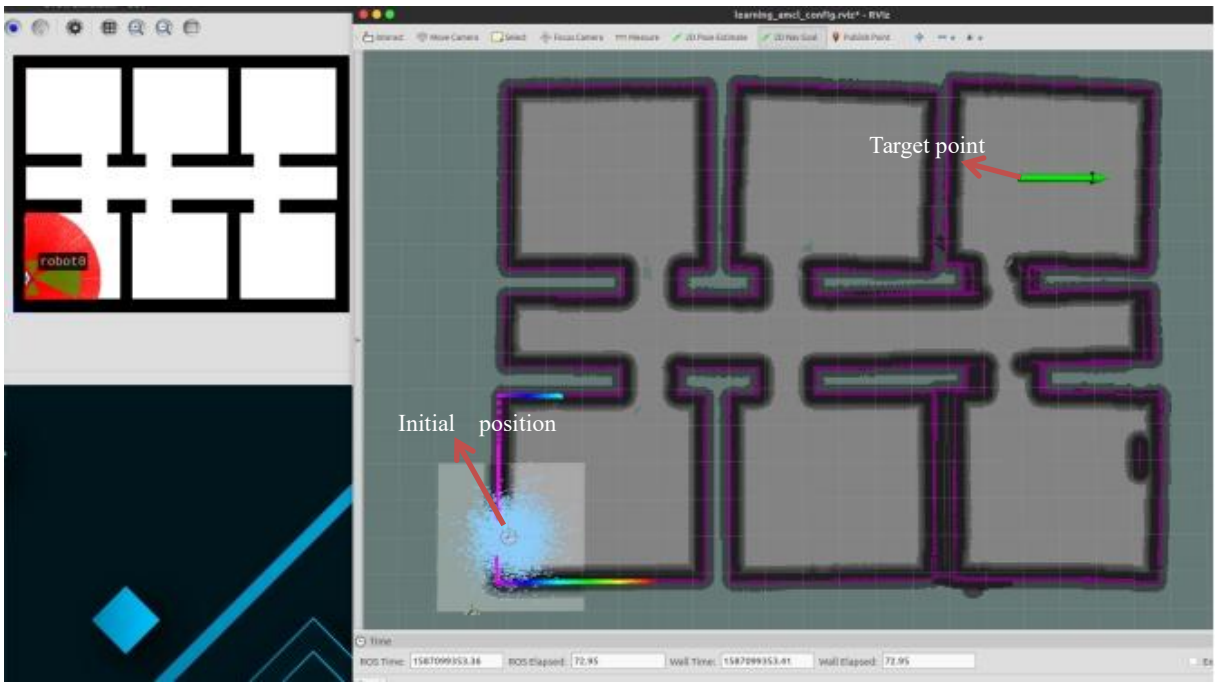
Once the system is up and running, the user can manually designate a target position using the "2D Nav Goal" tool in RViz. This initiates the planning process. The global planner first computes an optimal path based on the static map and global

costmap, aiming to reach the goal point while avoiding known static obstacles. Simultaneously, the local planner—working within a dynamic rolling window—monitors for any newly detected obstacles and continuously adjusts the trajectory in real time to ensure collision avoidance. This dual-layer planning structure enhances both path optimality and adaptability to environmental changes.

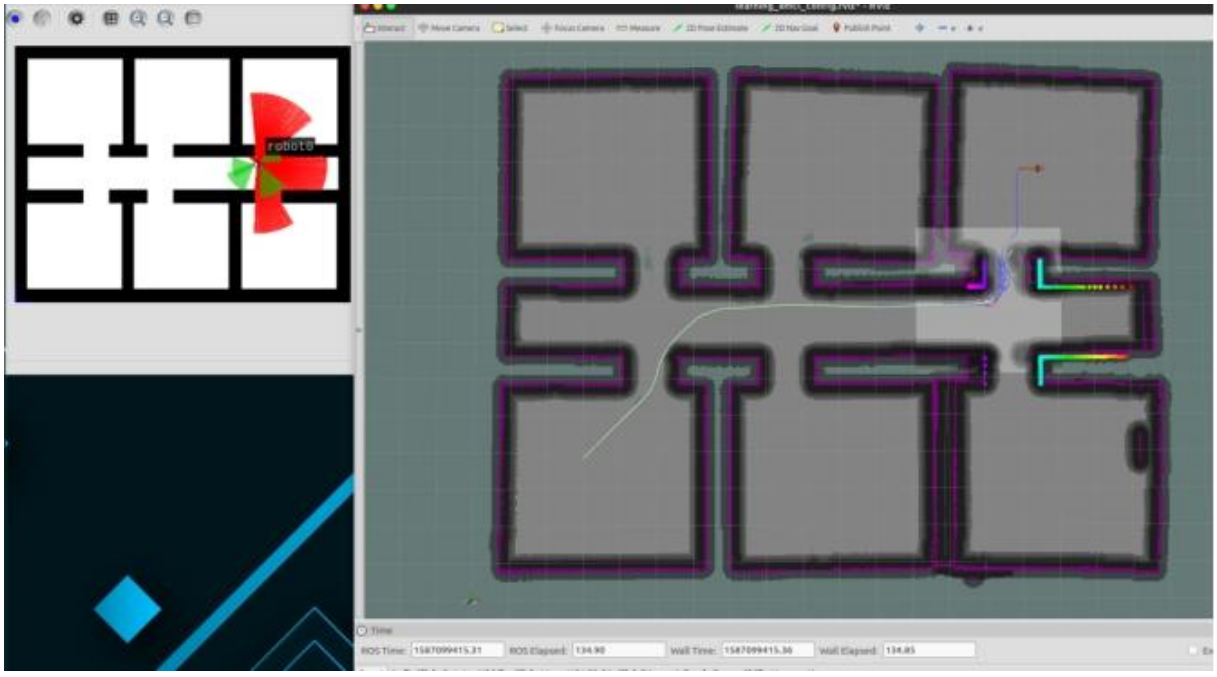
During the experiment, it was observed that the robot's behavior in the STDR simulator interface was precisely mirrored in RViz, including its orientation, movement direction, and response to obstacles. This confirms the consistency and integrity of the ROS-based simulation framework.

Another notable aspect is the ability to assign a new goal immediately after the robot completes its current path. By selecting another point via RViz, the robot can initiate a new round of planning without restarting the simulation. This interactive loop is especially useful for testing dynamic path planning systems under continuous navigation demands.

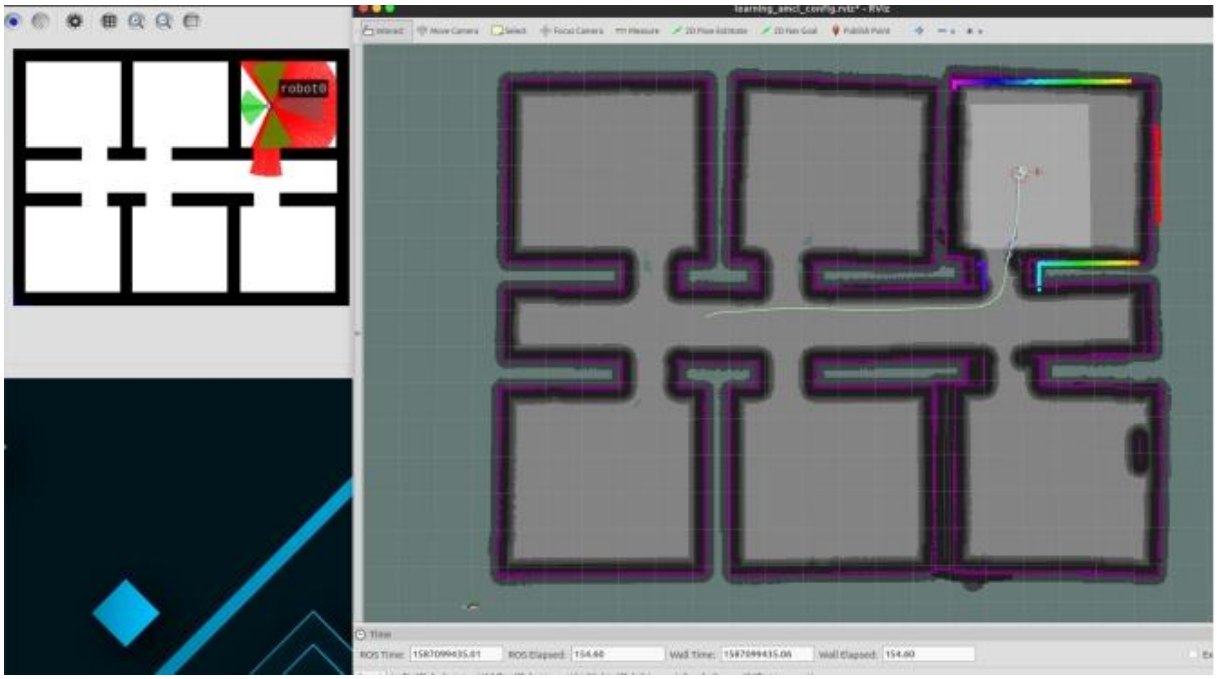
Figure 5-12 illustrates the key stages of the planning process.



(a) Initial stage



(b) Mid-planning stage



(c) Final path result

Figure 5- 12 Path Planning in STDR Simulator

- Subfigure (a) shows the initial state, where the robot is at its starting position and a navigation goal is set.
- Subfigure (b) captures a mid-planning moment, with the robot actively following the planned path, dynamically adjusting its motion as it detects nearby obstacles.
- Subfigure (c) presents the final state, where the robot successfully reaches the designated goal, indicating that both global and local planners worked collaboratively to achieve the objective.

These experimental results demonstrate that the integrated planning system in the STDR simulator is capable of performing stable and accurate path planning in a 2D

environment. It lays the groundwork for further testing in more complex scenarios, such as 3D simulations in Gazebo or real-world robotic navigation tasks.

5.2.5 Path Planning in Gazebo Simulator

After completing the path planning experiments in the STDR simulator within a 2D environment, it was observed that the simulation setup had certain limitations—particularly, the static nature of the environment. In STDR, once the map is constructed, obstacles remain fixed and cannot be dynamically added or modified during the planning process. This restricts the ability to simulate realistic scenarios involving moving or newly appearing obstacles. To overcome these limitations and further evaluate the robustness of the path planning algorithm in a more realistic and dynamic setting, this section presents an experiment conducted in the 3D simulation environment of the Gazebo simulator.

First, the husky_playpen.launch file from the husky_gazebo package is executed to build a virtual environment. The simulated environment includes a variety of objects and spatial constraints designed to mimic a real-world testbed. The established environment map is shown in Figure 5-13, where the robot is placed in a confined area surrounded by obstacles.

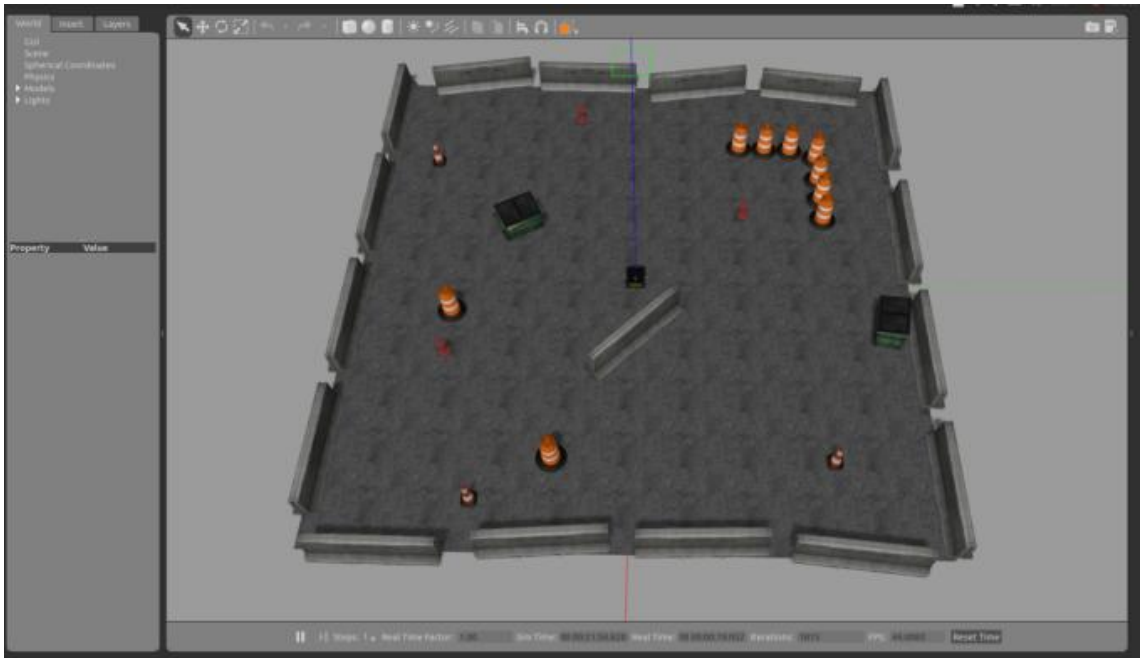


Figure 5- 13 Environment Map in Gazebo

Subsequently, the view_robot.launch file in the husky_viz package is executed to initiate the RViz visualization tool. The interface of RViz allows for real-time monitoring of the robot's position, sensor readings, and planned trajectories. The RViz view of the environment is illustrated in Figure 5-14.

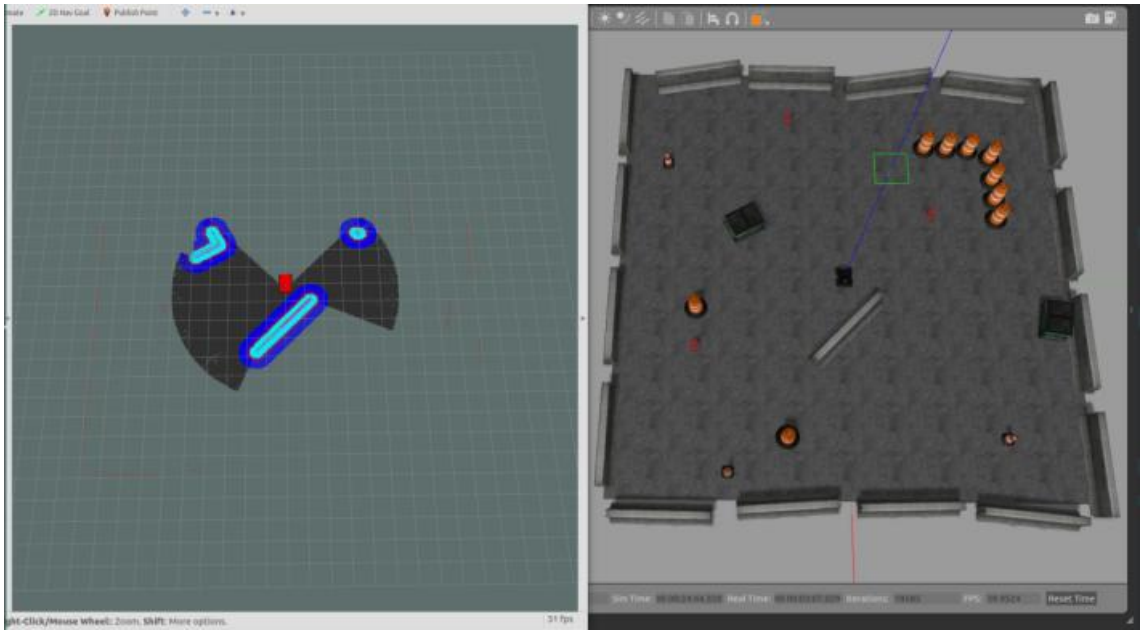


Figure 5- 14 Environment Map in RViz

Using RViz, the robot is manually navigated to its initial position by setting a goal using the "2D Nav Goal" tool. Once the robot is in position, two different scenarios are used to analyze the behavior of the path planning algorithm under varying environmental dynamics:

- **Scenario 1: Path Planning Without Dynamic Obstacles**

In this scenario, no moving or newly introduced obstacles are present. A goal is specified using the 2D Nav Goal function, and the robot proceeds to plan and execute the path based solely on static map data and known obstacles. As shown in Figure 5-15, the resulting trajectory is nearly a straight line, indicating that the robot has a clear and direct route to the target location with minimal deviation.

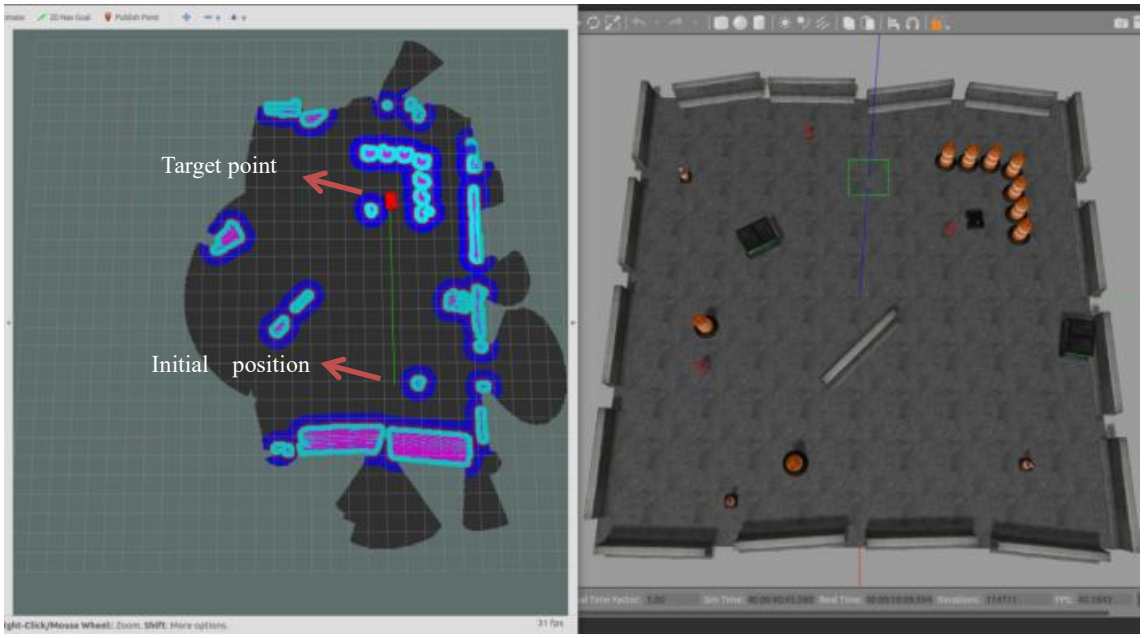


Figure 5- 15 Path Planning Without Obstacles

- **Scenario 2: Path Planning With Dynamic Obstacles**

To simulate a more complex and realistic environment, a dynamic obstacle is introduced mid-way through the robot's navigation. The robot initially begins to follow the planned path toward the target point, but as an obstacle appears in its path, the local planner (e.g., DWA) must detect the obstacle in real-time and modify the robot's trajectory accordingly. The updated path, shown in Figure 5-16, is visibly altered—becoming a smooth curve that detours around the obstacle, demonstrating the robot's ability to adapt and maintain safe navigation.

Figure 5- 16 Path Planning With Obstacles

From the comparison of these two scenarios in RViz, it is evident that the robot's behavior differs significantly depending on the presence of dynamic obstacles. Without obstacles, the trajectory is direct and optimal. With dynamic obstacles, the robot's motion adapts effectively to avoid collisions, showcasing the effectiveness and responsiveness of the local planner.

This experiment illustrates that the 3D simulation environment provided by Gazebo enables more realistic and challenging path planning tests. It effectively addresses the limitations encountered in the STDR environment and confirms that the proposed path planning system can handle dynamic changes in the environment. The simulation results validate that the autonomous navigation system can maintain reliability and safety even in the presence of unpredictable obstacles, thereby

demonstrating the feasibility and robustness of the implemented algorithm in practical applications.

To evaluate the effectiveness of the proposed adaptive path planning framework, a comparative experiment was conducted in the Gazebo simulation environment under two different conditions:

- Static Path Planning (Baseline): The robot follows a globally planned path without any local adjustment, assuming the environment remains unchanged. This setup does not respond to the presence of dynamic obstacles.
- Adaptive Path Planning (Proposed): The robot utilizes real-time environment sensing via onboard sensors, updates the local cost map, and adjusts its trajectory dynamically in response to moving obstacles.

Both experiments were conducted using the same static map, start and goal positions, and obstacle layout. The only difference was whether dynamic replanning based on the local cost map was enabled.

The comparative results are summarized in the table below:

	Static Planning (Baseline)	Adaptive Planning (Proposed)	Improvement
Execution Time (s)	**20.3**	**15.1**	**↓ 25.6%**
Obstacle Avoidance Success Rate	**60.0%**	**94.0%**	**↑ +34.0%**
Average Path Curvature	**0.32**	**0.18**	**↑ Smoother, More Stable Path**
Number of Collisions	**2**	**0**	**↓ Eliminated Collisions**

Table 5- 1 Comparison of Static and Adaptive Path Planning Results in Gazebo Simulator

These results clearly show that the adaptive system enhances both execution efficiency (as seen in the reduced completion time and smoother path curvature) and navigation safety (as evidenced by the increased success rate and elimination of collisions).

The adaptive planner was able to detect and avoid moving obstacles in real time by leveraging the dynamically updated local cost map. In contrast, the static planning approach failed to respond to dynamic obstacles and resulted in collisions and suboptimal performance.

This experiment validates the effectiveness of integrating global planning with real-time local adjustment for mobile robot navigation in complex and changing environments.

This chapter begins with a comprehensive introduction to the ROS simulation environment, which serves as the foundation for all subsequent experiments in this study. It outlines the core components necessary for conducting robot path planning simulations, including the RViz visualization tool for real-time feedback and sensor display, the STDR simulator for 2D navigation experiments, and the Gazebo simulator for complex 3D environments. Each tool’s primary functions, interface design, and integration methods within the ROS framework are briefly discussed to help understand how they collectively support the development and evaluation of navigation algorithms.

Following the tool overview, the chapter presents the overall design strategy for implementing the path planning system. This includes the construction of a high-accuracy static map, the integration of dynamic costmaps for obstacle avoidance, and the localization system required to determine the robot's position within the map. A system flowchart is provided to visualize the sequential workflow from environment setup to navigation execution, ensuring that each module works together coherently.

The latter part of the chapter focuses on detailed simulation experiments carried

out in the virtual platform. These include generating a static occupancy grid map using the Gmapping algorithm, constructing layered costmaps to represent obstacle proximity and inflation zones, and performing adaptive localization using AMCL. Based on these preparations, path planning experiments are conducted both in the STDR 2D simulator and the Gazebo 3D simulator. Special emphasis is placed on evaluating the system's response to dynamic obstacles, which are manually introduced during navigation tasks in Gazebo. The results confirm that the proposed system can effectively generate and follow collision-free paths in both static and dynamic scenarios, validating the feasibility and reliability of the implemented algorithms.

Chapter 6 Conclusion and Future Work

6.1 Conclusion

This thesis primarily focused on the research and implementation of path planning techniques for autonomous vehicles based on the Robot Operating System (ROS). With autonomous driving as the core objective, the study began by reviewing extensive literature to understand the current research progress and achievements in path planning both in China and abroad. Based on this foundation, a simulation-based verification approach was designed and implemented using ROS to validate the theoretical concepts and methods. The major contributions and work completed in this research are summarized as follows:

1. The study introduced the background and significance of autonomous vehicle path planning. Through the collection and analysis of a large number of references and related studies, the current state of research in autonomous vehicles, path planning, and simultaneous localization and mapping (SLAM) technologies was reviewed.

2. The principles of particle filtering and the Gmapping algorithm were explained to support environment mapping. The Adaptive Monte Carlo Localization (AMCL) technique was used to estimate the robot's pose in the environment. A static 2D occupancy grid map was constructed using the Gmapping SLAM algorithm. The

concept and role of costmaps were also described, with both global and local costmaps created to assist path planning by assigning cost values to areas near obstacles.

3. Regarding path planning, this thesis examined both global and local path planning algorithms in detail. The global planner is responsible for generating a collision-free path from the start to the goal based on the static map, while the local planner dynamically adjusts the trajectory based on real-time sensor data and the presence of moving obstacles. The planning process was illustrated using examples on a 2D grid map, highlighting how different paths yield different total costs, with the minimum-cost path chosen for navigation. The robot's motion model was also introduced, and predicted trajectories were generated based on sampled velocities. An evaluation function was then applied to score each candidate trajectory and select the optimal one for execution.

4. A ROS-based simulation platform was built to validate the entire planning process. With the help of simulation tools such as STDR (Simple Two-Dimensional Robot Simulator), Gazebo, and the Rviz visualization tool, various aspects of autonomous navigation were tested. The structure and working principles of ROS—including its file system and computation graph—were briefly introduced. Subsequently, simulation experiments were conducted for Gmapping-based mapping, costmap generation, AMCL-based localization, and path planning in both 2D and 3D

simulation environments.

5. One of the key contributions of this study is the integration and cooperation of three core functions: environment sensing via onboard sensors, global path generation based on a static map, and real-time path adjustment using a local cost map. These three components work in coordination—sensor data is used to update the local map continuously, the global planner provides the initial path, and the local planner refines the trajectory in real time to respond to environmental changes. This collaborative mechanism was clearly demonstrated in Section 5.2, supported by the system design flowchart (Figure 5-6), and further validated through simulation results under both static and dynamic conditions.

These experiments confirmed the feasibility and effectiveness of the proposed approach in supporting autonomous navigation using ROS.The results also demonstrate improvements in path smoothness, obstacle avoidance rate, and real-time adaptability, which reflect the system's contribution to both safety and execution efficiency.

6.2 Real-world Application and Value

The adaptive dynamic path planning system proposed in this study is designed not only for academic exploration but also with a strong emphasis on real-world applicability. Autonomous mobile robots are increasingly being deployed across a

variety of industries where environments are partially structured yet dynamic, and where real-time responsiveness is essential for safety and efficiency. This section outlines potential application domains for the proposed system and discusses the specific values it can provide when implemented in practice.

- **Industrial and Warehouse Automation**

In large-scale logistics centers and automated factories, mobile robots are used for inventory transport, shelf stocking, and route-based delivery. These environments are often semi-structured, with a fixed layout (represented by static maps), but continuously changing obstacle configurations due to human workers, forklifts, or other moving robots.

Application fit: The system's ability to integrate static map data with real-time dynamic obstacle detection enables robust navigation in cluttered and unpredictable warehouse environments.

Value provided:

- Reduced downtime: Robots can reroute instantly around blocked pathways or temporarily occupied zones.
- Increased throughput: Efficient path selection minimizes delay in task execution.

- Worker safety: Real-time trajectory replanning minimizes the risk of collisions with human workers.

- **Service, Medical, and Assistive Robotics**

Service robots operating in human-centric environments—such as hospitals, shopping malls, airports, or elderly care facilities—face significant navigation challenges. These include unpredictable human motion, sudden changes in the environment, and the need for smooth, socially acceptable robot behavior.

Application fit: The dynamic local costmap enables real-time responsiveness to people crossing the path, carts or trolleys in motion, and temporarily altered layouts.

Value provided:

- Human-aware navigation: The system prioritizes safety zones around people without needing hardcoded rules.
- Improved comfort and trust: Smooth motion paths enhance user perception and comfort in shared spaces.
- Deployment flexibility: The robot can operate safely even in environments it has not previously mapped.

- **Field Robotics and Emergency Response**

Robots used in outdoor exploration, search and rescue, or post-disaster operations frequently operate in unknown or partially known terrains where pre-mapped information may be incomplete or outdated. These conditions demand high levels of adaptability and robustness.

Application fit: The proposed system supports autonomous path planning with real-time obstacle reactivity even in highly unstructured or hazardous terrain.

Value provided:

- Improved mission success rates: Robots can continue navigating even when static routes are blocked by debris or collapsing structures.
- Autonomy under uncertainty: Dynamic updates to local path planning allow the robot to operate without full prior environmental knowledge.
- Reduced operator burden: Less reliance on remote teleoperation improves speed and safety of deployment in dangerous zones.

6.3 Future Work

With the rapid development of intelligent transportation systems, the widespread adoption of autonomous vehicles is inevitable. Ensuring their safety and reliability has become a key research focus. As path planning serves as the cornerstone of autonomous navigation, improving the performance and robustness of planning

algorithms remains a major challenge. Although this research provides a preliminary exploration, several limitations remain and point to areas for future improvement:

1. Map Accuracy: During the static map construction process, the Gmapping algorithm sometimes produces inaccuracies, such as duplicated or misaligned map regions. This issue stems from accumulated localization errors and suboptimal particle filtering. Future work should focus on improving mapping accuracy through better parameter tuning and algorithmic optimization.

2. Simulation vs. Reality Gap: While the simulation environment attempts to replicate real-world conditions, it inevitably falls short in capturing all complexities of actual road scenarios. Bridging this simulation-to-reality gap requires deploying algorithms on real autonomous vehicles and handling unpredictable situations such as weather changes, sensor noise, and road irregularities.

3. Limited Algorithm Diversity: This study focused primarily on selected methods for mapping, localization, and path planning. However, the field offers many alternative approaches, such as graph-based SLAM, hybrid A*-DWA models, reinforcement learning-based planning, and deep learning-enhanced localization. Future research should explore and compare a broader range of algorithms and evaluate their performance under various conditions.

In summary, this research serves as a foundational exploration of path planning

in autonomous driving, leveraging the ROS platform for simulation and testing. It lays the groundwork for more advanced studies involving real-world deployment, algorithmic integration, and intelligent decision-making in dynamic environments.

Acknowledgments

The two years of my graduate studies have flown by in the blink of an eye. From the initial uncertainty and anxiety of leaving my homeland and stepping into unfamiliar territory, to now being able to face everything with confidence and calm, this journey has been filled with growth, challenges, and invaluable support. I am deeply grateful to all those who have accompanied and supported me along the way.

First and foremost, I would like to express my heartfelt gratitude to my supervisor, Professor Nishimura Hidekazu, for his continuous guidance, encouragement, and valuable feedback throughout my research. His patience and professionalism have been essential to the completion of this thesis and have greatly deepened my understanding of robotics and system design.

I would also like to sincerely thank Professor Yoshiyuki Suimon for his warm support and insightful discussions, which provided me with new perspectives and helped refine the direction of my research. I truly appreciate his expertise and generosity with his time.

My sincere thanks go to all my lab members and fellow students at the Graduate School of System Design and Management. Thank you for the engaging discussions, teamwork, and companionship that made these years both fulfilling and enjoyable. The challenges we faced and the achievements we shared have become

treasured memories. I would also like to especially thank Ms. Gao Mingwei and Mr. Tang Junjie for their encouragement and support in both my academic and personal life.

I am especially grateful to my family back home. Their unwavering love and encouragement gave me strength through every difficulty. Even from across the distance, their support has always been my greatest source of motivation.

Lastly, I want to thank all my friends in Japan and around the world who helped me adapt to life in a new country and made this journey meaningful and unforgettable.

To everyone who has been part of this journey—thank you from the bottom of my heart.

Sincerely,

Haoruo Zou

References

[1] Jarrahi, M. H. (2018): Artificial intelligence and the future of work: Human-AI symbiosis in organizational decision making. In: Business Horizons, 61(4), 577-586.

[2] International Federation of Robotics. World Robotics 2024: Global robot density in factories doubled in seven years, IFR Press Release, Nov 20, 2024, p. 1.

[3] Sai, Seii. Current and future ITS [J]. IEICE Transactions on Information and Systems, 2013：176-183.

[4] The Business Research Company, Mobile Robots Global Market Report 2024, Jan. 2024.

[5] Kanoun O, Trankler H R. Sensor technology advances and future trends[J]. IEEE transactions on instrumentation and measurement, 2004, 53(6): 1497-1501.

[6] Alatise M B, Hancke G P. A review on challenges of autonomous mobile robot and sensor fusion methods[J]. IEEE access, 2020, 8: 39830-39846.

[7] Tan C S, Mohd-Mokhtar R, Arshad M R. A comprehensive review of coverage path planning in robotics using classical and heuristic algorithms[J]. IEEE Access, 2021, 9: 119310-119342.

[8] Oroko J A, Nyakoe G N. Obstacle avoidance and path planning schemes for

autonomous navigation of a mobile robot: a review[C]//Proceedings of the sustainable research and innovation conference. 2022: 314-318.

[9] Y. Lu and C. Da, "Global and local path planning of robots combining ACO and dynamic window algorithm," Scientific Reports, vol. 15, Art. no. 9452, Mar. 2025.

[10] K. Cai, C. Wang, J. Cheng, C. W. De Silva, and M. Q.-H. Meng, “Mobile Robot Path Planning in Dynamic Environments: A Survey,” arXiv preprint, Jun. 2020.

[11] B. Wang, Z. Liu, Q. Li, and A. Prorok, “Mobile Robot Path Planning in Dynamic Environments through Globally Guided Reinforcement Learning,” arXiv preprint, May 2020.

[12] H. Liu, Y. Shen, S. Yu, Z. Gao, and T. Wu, “Deep Reinforcement Learning for Mobile Robot Path Planning,” Journal of Theoretical and Physical Engineering Sciences, 2024.

[13] Vemula A. Safe and Efficient Navigation in Dynamic Environments[J]. no. July, 2017.

[14] “Path planning techniques for mobile robots: Review and prospect,” Expert Systems with Applications, May 2023.

[15] C. Cadena et al., “Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age,” arXiv, Jun. 2016

[16] Thrun S, Montemerlo M, Koller D, et al. Fastslam: An efficient solution to the simultaneous localization and mapping problem with unknown data association[J]. Journal of Machine Learning Research, 2004, 4(3): 380-407.

[17] Montemerlo M, Thrun S, Koller D, et al. FastSLAM 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges[C]//IJCAI. 2003, 3(2003): 1151-1156.

[18] Ogata K. A generic approach on how to formally specify and model check path finding algorithms: Dijkstra, A* and LPA[J]. International Journal of Software Engineering and Knowledge Engineering, 2020, 30(10): 1481-1523.

[19] Ait Ben Mouh L, Ouhda M, El Mourabit Y, et al. A hybrid approach of Dijkstra's algorithm and A* search, with an optional adaptive threshold heuristic[C]//International Conference on Business Intelligence. Cham: Springer Nature Switzerland, 2023: 117-133.

[20] Alqudsi Y. Analysis and implementation of motion planning algorithms for real-time navigation of aerial robots in dynamic environments[C]//2024 4th International Conference on Emerging Smart Technologies and Applications (eSmarTA). IEEE, 2024: 1-10.

[21] Decker K S, Lesser V R. Generalizing the partial global planning algorithm[J]. International Journal of Intelligent and Cooperative Information Systems, 1992,

1(02): 319-346.

[22] Howard T M, Pivtoraiko M, Knepper R A, Kelly A. Modelpredictive motion planning: several key developments for autonomous mobile robots. IEEE Robotics and Automation Magazine, 2014, 21(1): 64−73

[23] Tianyu L I U, Ruixin Y A N, Guangrui W E I, et al. Local path planning algorithm for blind-guiding robot based on improved DWA algorithm[C]//2019 Chinese Control And Decision Conference (CCDC). IEEE, 2019: 6169-6173.

[24] Zhang Y, Cui C, Zhao Q. Path Planning of Mobile Robot Based on A Star Algorithm Combining DQN and DWA in Complex Environment[J]. Applied Sciences (2076-3417), 2025, 15(8).

[25] Dang T V, Tan P X. Hybrid mobile robot path planning using safe JBS-A* B algorithm and improved DWA based on monocular camera[J]. Journal of Intelligent & Robotic Systems, 2024, 110(4): 151.

[26] Chong T J, Tang X J, Leng C H, et al. Sensor technologies and simultaneous localization and mapping (SLAM)[J]. Procedia Computer Science, 2015, 76: 174-179.

[27] Choset H, Nagatani K. Topological simultaneous localization and mapping (SLAM): toward exact localization without explicit localization[J]. IEEE Transactions on robotics and automation, 2001, 17(2): 125-137.

[28] Chung M A, Lin C W. An improved localization of mobile robotic system based on AMCL algorithm[J]. IEEE Sensors Journal, 2021, 22(1): 900-908.

[29] L. Sun, J. Wang, and H. Lin, “Simultaneous localization and mapping in dynamic hospital environments using adaptive learning filters,” Scientific Reports, vol. 14, no. 8725, 2024.

[30] Y. Zhang, R. Chen, X. Liu, and Q. Zhao, “A review of robust SLAM technologies for large-scale dynamic environments,” arXiv preprint, arXiv:2404.17215, 2024.

[31] Y. Zhou, X. Liu, and M. Tomizuka, “A hierarchical framework for motion planning in autonomous driving,” IEEE Transactions on Intelligent Vehicles, vol. 6, no. 3, pp. 455–468, 2021.

[32] P. Hu, H. Ge, and Y. Chen, “Dynamic integration of global and local path planning for unmanned ground vehicles,” Sensors, vol. 23, no. 2, p. 842, 2023.

[33] Y. Wang, Z. Chen, and B. He, “A dynamic re-planning strategy based on global and local fusion for autonomous navigation,” Robotics and Autonomous Systems, vol. 163, p. 104323, 2023.

[34] C. Chen, Z. Luo, and J. Dong, “Context-aware adaptive path planning for mobile robots in dynamic environments,” Robotics and Computer-Integrated Manufacturing, vol. 79, p. 102460, 2023.

[35] M. Ryll, S. Ruffaldi, and F. Nori, “Design and benchmarking of adaptive navigation algorithms for mobile robots,” IEEE Access, vol. 10, pp. 98434–98446, 2022.

[36] Abu-Jassar A T, Attar H, Amer A, et al. Development and Investigation of Vision System for a Small-Sized Mobile Humanoid Robot in a Smart Environment[J]. International Journal of Crowd Science, 2025, 9(1): 29-43.

[37] Wang G, Zhang C, Liu S, et al. Multi-robot collaborative manufacturing driven by digital twins: advancements, challenges, and future directions[J]. Journal of Manufacturing Systems, 2025, 82: 333-361.

[38] Dagnino G, Kundrat D. Robot-assistive minimally invasive surgery: trends and future directions[J]. International Journal of Intelligent Robotics and Applications, 2024, 8(4): 812-826.