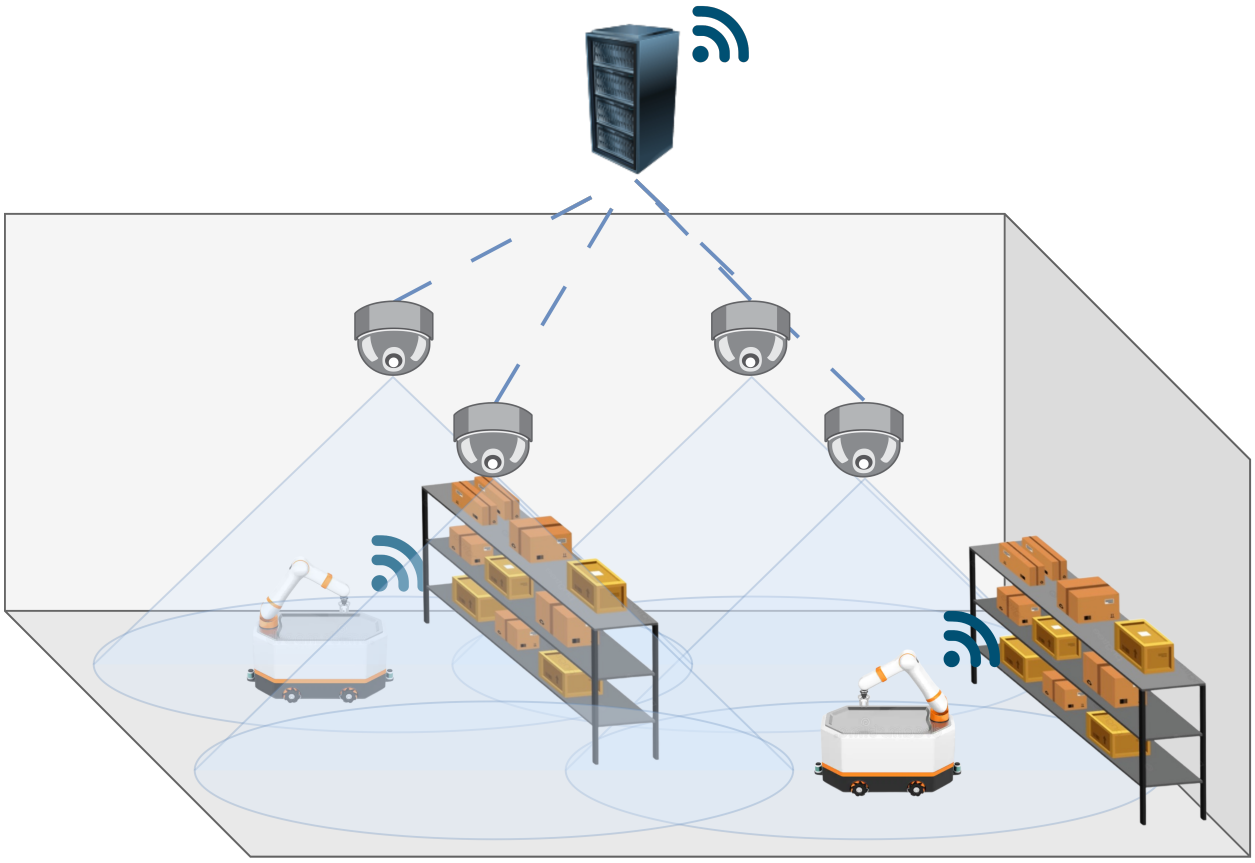


Toward Efficient Collaboration in Autonomous Mobile Robot Fleets: Addressing Latency and Distributed MPC

Master's thesis in Systems, Control and Mechatronics

Yinsong Wang & Zihao Lu

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2025
www.chalmers.se

Master's thesis 2025

Toward Efficient Collaboration in Autonomous Mobile Robot Fleets: Addressing Latency and Distributed MPC

Yinsog Wang & Zihao Lu

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Division of Systems and Control
Automation
Chalmers University of Technology
Gothenburg, Sweden 2025

Toward Efficient Collaboration in Autonomous Mobile Robot Fleets:
Addressing Latency and Distributed MPC
Yinsong Wang & Zihao Lu

Supervisor: Prof. Knut Åkesson, Department of Electrical Engineering
Examiner: Prof. Knut Åkesson, Department of Electrical Engineering

Master's Thesis 2025
Department of Electrical Engineering
Division of Systems and Control
Automation
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2025

Toward Efficient Collaboration in Autonomous Mobile Robot Fleets:
Addressing Latency and Distributed MPC
Yinsong Wang & Zihao Lu
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Cloud-based multi-robot systems offer significant advantages in computation capacity and coordination capabilities, but face critical challenges related to communication latency that can compromise system performance and safety. This paper presents a latency-tolerant hierarchical control architecture for Autonomous Mobile Robot (AMR) fleet coordination that maintains operational integrity despite network delays. The proposed multi-layer framework integrates (1) a cloud component implementing switchable distributed model predictive control (DMPC) and path sampling for trajectory generation, (2) an AMR component featuring switchable controllers (Pure Pursuit and hybrid LQR) for local trajectory tracking, and (3) a comprehensive simulation environment for systematic evaluation. We introduce a custom ROS2-based communication infrastructure with intelligent finite state machine design to manage complex system behavior and enhance resilience to varying network communication latency conditions. Experimental results across multiple test scenarios demonstrate that our approach maintains over 90% success rates even under high-latency conditions (1000ms), with limited path deviation and execution time. The findings provide quantitative insights into the latency impact of MPC solvers based on different scenarios and offer practical solutions for robust cloud-robotic systems design and implementation.

Keywords: Multi-robot system, Multi-layer control, Cloud robotics, Distributed model predictive control, AMR fleet coordination, Trajectory generation, Trajectory tracking, Latency impacts.

Acknowledgements

We would like to express our deepest gratitude to our supervisor, Knut Åkesson, for his continuous support, insightful guidance, and constructive feedback throughout the journey of this thesis. His expertise and encouragement have been invaluable to our growth and learning.

We are also grateful to our classmates, friends, and to Kristian Ceder and Ze Zhang for the stimulating discussions and for fostering a supportive and inspiring environment.

Lastly, we extend our heartfelt thanks to our families for their unwavering support, patience, and belief in us—this journey would not have been possible without them.

Yinsong Wang & Zihao Lu, Gothenburg, June 2025

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

act	Activity Diagram
AMR	Autonomous Mobile Robot
bbd	Block Definition Diagram
CBFs	Control Barrier Functions
CMPC	Centralized Model Predictive Control
DARE	Discrete Algebraic Riccati Equation
DDS	Data Distribution Service
DMPC	Distributed Model Predictive Control
FSM	Finite State Machine
IoT	Internat of Things
LAN	Local Area Network
LQR	Linear Quadratic Regulator
MPC	Model Predictive Control
MRAN	Multi-robot Autonomous Negotiation
MRS	Multi-robot System
OpEn	Optimization Engine
PANOC	Proximal Averaged Newton-type method for Optimal Control
QoS	Quality of Service
RL	Reinforcement Learning
ROS/ROS2	Robot Operating System
RViz	Robot Visualization
SLAM	Simultaneous Localization And Mapping
smd	State Machine Diagram

Nomenclature

Below is the nomenclature of indices, sets, parameters, and variables that have been used throughout this thesis.

Indices

i, j	Indices for robots in the fleet
n	Index for static obstacles
m	Index for half-space constraints
k	Discrete time step
N	Planning horizon length

Sets

$\mathcal{F}$	Set of mobile robots in the fleet
$\mathcal{B}$	Set of boundary constraints
$\mathcal{O}$	Set of static obstacles
$\mathcal{D}$	Set of dynamic obstacles
H_n	Set defining the n-th obstacle (convex polygon)
I	Set of all pairs of robot indices (i, j) where $i \neq j$

State and Control Variables

$\mathbf{x}_k^{(i)}$	State vector $[x_k^{(i)}, y_k^{(i)}, \theta_k^{(i)}]^\top$ of robot i at time step k
(x_k, y_k)	Cartesian coordinates of robot position
θ_k	Heading angle of robot
$\mathbf{u}_k^{(i)}$	Control input vector $[v_k^{(i)}, \omega_k^{(i)}]^\top$ of robot i at time step k
v_k	Linear velocity at time step k
ω_k	Angular velocity at time step k

$\mathbf{c}^{(i)}$	Center position of robot i
$\tilde{\mathbf{x}}_k$	Reference state at time step k
$\tilde{\mathbf{u}}_k$	Reference control input at time step k
$\mathbf{u}_{min}, \mathbf{u}_{max}$	Lower and upper bounds for control inputs

System Model

$f(\mathbf{x}_k, \mathbf{u}_k)$	Discrete-time kinematic motion model
T_s	Sampling time
P	Number of robots in the fleet

Obstacle Modeling

r	Radius of each mobile robot
d_0	Safe distance threshold for collision avoidance
H_n	Convex polygon representation of static obstacle n
$h_{n,m}(\mathbf{p})$	Half-space constraint function for obstacle n, inequality m
$e(\mathbf{p})$	Elliptical obstacle constraint function
$\mathbf{a}_{n,m}, \mathbf{b}_{n,m}$	Coefficients of linear constraints for obstacles
N_o	Number of static obstacles
N_d	Number of dynamic obstacles
M	Number of inequalities defining a polygon
E	Positive definite matrix defining ellipse shape and orientation

Cost Functions and Penalties

J_{R_i}	Overall cost function for robot i
J_τ	Penalty function for deviation from reference trajectory
J_u	Penalty function for deviation from reference control inputs
J_a	Penalty function for control smoothness
$J_\mathcal{F}$	Penalty function for robot-to-robot collisions
$J_\mathcal{O}$	Penalty function for static obstacle avoidance
$J_\mathcal{D}$	Penalty function for dynamic obstacle avoidance
Q_k	Adaptive weighting matrix for state deviation

α	Adaptation gain that scales the sensitivity of the state deviation weight Q_k relative to the deviation magnitude
R	Weighting matrix for control input deviation
S	Weighting matrix for control smoothness
$Q_{\mathcal{F}}, Q_{\mathcal{O}}, Q_{\mathcal{D}}$	Weighting coefficients for penalty functions
$[\cdot]_+$	Function defined as $\max\{0, \cdot\}$
$[\cdot]_-$	Function defined as $\min\{0, \cdot\}$

Pure Pursuit Controller

L	Lookahead distance
(x_L, y_L)	Lookahead point coordinates
(x'_L, y'_L)	Transformed lookahead point in vehicle frame
κ	Curvature for path following
v_{max}	Maximum allowable velocity
α	Tuning parameter for speed reduction in turns
d_i	Distance from current position to waypoint i

LQR Controller

$\mathbf{x}_{ref}$	Reference state for LQR
$\mathbf{u}_{ref}$	Reference control input for LQR
$\delta\mathbf{x}_k$	Error state (deviation from reference)
$\delta\mathbf{u}_k$	Control error (deviation from reference control)
A, B	System matrices for linearized model
Q	Positive semi-definite state weighting matrix
R	Positive definite control weighting matrix
K	Feedback gain matrix
P	Solution to discrete algebraic Riccati equation
$d_{lookahead}$	Lookahead distance for hybrid controller

Evaluation Metrics

A_{dev}	Deviation Area
D_{norm}	Normalized Deviation

T_{exec}	Execution Time
L_p	Path Length
S_l	Linear Smoothness
S_a	Angular Smoothness
R_{conv}	Convergence Success Rate
$R_{success}$	Success Rate
$R_{collision}$	Collision Rate
$R_{timeout}$	Timeout Rate

Contents

List of Figures

List of Tables

1
Introduction

This chapter lays the groundwork for our investigation into cloud-integrated multi-robot systems, with an emphasis on managing communication latency. We begin by reviewing the market trends and architectures in multi-robot systems (MRS), followed by a discussion of our research motivation, objectives, and key questions. A survey of related literature contextualizes our contributions, which aim to improve coordination robustness under variable network conditions.

1.1 Background

The field of MRS has evolved significantly over the decades, reflecting broader advancements in robotics technology and collaborative methodologies. As discussed in [1], multi-robot systems are characterized by the coordinated efforts of multiple robots working together to achieve specific objectives, a paradigm that has gained substantial traction across various domains. This section provides a comprehensive background on multi-robot systems, covering market trends, architectural classifications, application domains, and the integration of cloud computing for enhanced coordination.

As Figure 1.1 shows, the global robotics market is projected to grow by USD 18.79 billion at a compound annual growth rate of 6.1% between 2024 and 2029. According to the analysis in [2], this growth is driven primarily by increasing adoption of collaborative robots and Robotics as a Service business models that eliminate the need for substantial upfront investments.
Multi-robot systems can be classified into three distinct architectural paradigms based on their operational characteristics and scale, as shown in Table 1.1. Teams typically consist of a small number of robots (usually fewer than 10) that optimize individual objectives in either cooperative or competitive scenarios. In a team, the behavior and strategies of each individual agent seek to explicitly maximize a local objective, allowing for specialized roles while contributing to the overall mission. Formations involve robots assigned to specific subtasks, roles, or placements within a structured arrangement, commonly including tens of robots. As described in [3], swarms, by contrast, comprise large groups of dispensable agents whose global capabilities emerge from their collective behavior rather than centralized control.

Multi-robot systems have demonstrated remarkable versatility across diverse applications. From environmental monitoring and search and rescue operations to

Figure 1.1: Market Size Outlook for the robotics market (2019-2029). The market showed accelerating growth momentum with 5.4% Year-over-Year in 2025 and is projected to grow at a CAGR of 6.1% between 2024-2029 [2].

Table 1.1: Classification of Multi-robot Systems

Type	**Scope**	**Size**
Team	Typically small groups; each agent optimizes individual objectives in a cooperative or competitive manner	Typically ≤ 10
Formation	Each agent is typically assigned a specific subtask, role, or placement	Typically ≤ 10
Swarm	Typically large groups of dispensable agents; global capability arises from emergent behavior	Large

manufacturing and logistics, these systems leverage collective behavior to achieve objectives that would be difficult or impossible for single robots. Particularly noteworthy is the field of swarm robotics, which, as explored in [4], draws inspiration from biological systems to create large-scale robot collectives that exhibit emergent intelligence through relatively simple individual behaviors.

As multi-robot systems grow in complexity and scale, one of the key challenges they face is the management of computational resources and coordination mechanisms. Cloud robotics has emerged as a transformative approach to address these limitations, enabling robots to leverage distributed computing resources for enhanced performance and capabilities beyond what their onboard systems can provide independently.

The integration of cloud computing with multi-robot systems offers several dis-

tinct advantages for fleet coordination, particularly for AMR applications. In the work [5], the concept of Cloud Networked Robotics was introduced, demonstrating how distributed infrastructure can support multi-location robotic services, which is particularly valuable in scenarios requiring coordination across physically separated robots. This approach has significant implications for large-scale logistics operations where AMRs must operate cohesively across extensive facilities.

Building on this foundation, in [6], Wang et al. proposed a multi-robot autonomous negotiation (MRAN) module within cloud infrastructure that facilitates co-localization among robots. This collaborative functionality is essential for AMR fleet coordination, where precise relative positioning between robots enables efficient path planning and collision avoidance in shared workspaces.

For computation-intensive tasks common in multi-robot coordination, such as simultaneous localization and mapping (SLAM), offloading to cloud resources has proven effective. The research [7] explored SLAM task offloading to edge and fog computing services, alleviating the computational burden on individual robots. This approach is particularly valuable in complex warehouse environments where multiple AMRs must maintain and share consistent environmental maps while navigating dynamically changing surroundings.

Energy efficiency represents another critical consideration in AMR fleet operations. In [8], Rahman et al. developed a multi-layer decision-making scheme specifically designed for energy-efficient task offloading in cloud networked multi-robot systems. Their framework optimizes the offloading process by considering factors such as task selection and network access point determination, thereby extending operational duration and reducing charging requirements for AMR fleets.

These studies highlight the need for robust coordination strategies in cloud-integrated MRS environments. However, existing frameworks rarely quantify the impact of latency on control performance or implement latency-tolerant development.

1.2 Motivation

Cloud robotics has emerged as a paradigm that extends traditional robotic systems by leveraging cloud computing infrastructure. As defined in the work of [9], cloud robotics enables robots to benefit from the cloud's vast computational resources, storage capacities, and collaborative capabilities. This integration offers several significant advantages for multi-robot systems. First, it facilitates complex computations that would be stressful on resource-constrained robotic platforms, enabling advanced algorithms for perception, planning, and learning, as demonstrated in [10]. Second, cloud-based architectures promote efficient sharing of data and experiences among multiple robots, accelerating collective learning and adaptation according to the research presented in [11].

Despite these compelling advantages, the integration of cloud computing with robotics

introduces significant challenges, particularly in terms of communication latency. This issue becomes increasingly critical in cloud robot cluster systems, where timely coordination among multiple robots is essential for effective operation. The impact of latency varies substantially under different optimization strategies, system architectures, and network conditions, warranting dedicated investigation.

Recent studies in [12] and [13] highlight how packet loss and high latency degrade the quality of service (QoS), especially in cloud-IoT ecosystems. The findings in these works reveal that transmission delays can render time-sensitive robotic operations unsafe or ineffective, creating a fundamental challenge for real-time coordination.

While the layered architecture of cloud robotics effectively offloads resource-intensive functions to the cloud—thereby minimizing resource consumption on edge nodes—as highlighted in the research of [14], this approach inevitably introduces latency-related challenges that can compromise system performance and reliability. This trade-off becomes particularly apparent in multi-robot scenarios, where coordination timing directly impacts collective behavior, leading to potential collision or disconnection problems.

The tension between the significant advantages of cloud robotics and the inherent challenges of communication latency motivates our research.

1.3 Research Questions

This research addresses four key questions regarding latency management in cloud-based multi-robot systems, focusing on architectural design, coordination mechanisms, error handling, and performance evaluation.

- **RQ1: How to design cloud-based architecture for efficient AMR fleet coordination?**
- **RQ2: How can DMPC be applied to coordinate robot fleets effectively under high-latency conditions?**
- **RQ3: What strategies can be employed on individual AMRs to manage outdated trajectory information when latency and computation bottlenecks exist?**
- **RQ4: What is the quantitative impact of varying communication latency conditions on multi-robot system performance?**

1.4 Related work

The growing complexity of multi-robot systems—particularly in cloud-integrated and latency-sensitive environments—demands robust architectural designs and communication strategies capable of supporting real-time coordination and control. To contextualize these challenges, the following reviews of related works addressing system architecture, communication latency, trajectory coordination, and tracking are provided. While cloud robotics enables scalable fleet management and intelligent task allocation, it also introduces significant challenges related to communication latency and system responsiveness. The trade-offs between centralized and distributed architectures critically influence how efficiently robots can perceive, plan, and act in dynamic environments. Moreover, reliable trajectory coordination and tracking require both effective path planning algorithms and well-designed communication frameworks to ensure safe execution. As multi-robot applications expand into mission-critical domains, addressing these interrelated challenges becomes essential for achieving scalable, responsive, and dependable robotic collaboration.

1.4.1 Multi-robot System integrated Cloud Robotics

Multi-robot systems require well-designed structural frameworks to efficiently manage resources and communication. In their research, [15] defines robot cloud as an emerging cloud computing paradigm that leverages robot networking, big data, AI, and edge computing technologies to manage robot resources, provide backend AI capabilities, support edge computing orchestration, and analyze operational data for business intelligence.

Recent research on multi-robot system architectures emphasizes the importance of efficient and robust architecture design. As presented in [16], contemporary studies highlight the integration of autonomous robots with advanced communication resources as key enablers for enhanced multi-robot collaboration, with frameworks organized around perception, planning, and coordination as fundamental pillars. According to the classification in [17], architectural approaches to multi-robot coordination can be categorized into several algorithmic categories, including task allocation, path planning, area coverage, and centralized versus distributed planning methods. In the review by [18], it is noted that the choice between centralized architectures—where a single control unit manages all environmental information and coordinates subordinate robots through command issuance—and distributed architectures—where individual robots maintain greater autonomy—directly impacts a system's ability to operate effectively under varying network conditions.

Meanwhile, some studies focus on task allocation strategies within cloud robotics. For example, in works of [19] [20] proposed a sector-based hierarchical architecture to coordinate traffic heatmaps across different sectors. Meanwhile, several cloud platforms have emerged, such as the KubeEdge-based Multi-Robot Collaboration Framework described in [21] and the Rapyuta Cloud Robotics Platform introduced in [22]. However, these platforms primarily address high-level aspects of

AMR fleet coordination and management, including task assignment as discussed in [21], scheduling, and traffic balancing methods presented in [20].

1.4.2 Latency in Multi-robot Systems

Research has highlighted the critical importance of achieving minimal communication latency in cloud-based multi-robot systems, particularly for mission-critical applications. Traditional TCP/IP communication protocol-based synchronizing middleware that relied on ROS's master-based architecture consumed significant bandwidth and introduced communication delays. In the work [23], it has been demonstrated that implementing ROS2-based architectures with masterless packet discovery mechanisms can substantially reduce latency (by approximately 12% in simulation and 11% in real-world deployments) while simultaneously improving reliability through reduced packet loss (10% reduction in simulation and 15% in real environments). The work described in [24] emphasizes that network latency is the key point of concern in cloud robotics implementations, highlighting the critical nature of communication efficiency in these systems.

Further research has revealed significant challenges in managing communication latency for multi-robot systems in industrial environments. In [25], Kronauer et al. demonstrated that ROS2's end-to-end latency is heavily dependent on the chosen Data Distribution Service (DDS) middleware, with findings indicating that ROS2 can introduce up to 50% latency overhead compared to direct low-level DDS communications, which has profound implications for time-critical robotic operations. Similarly, studies on real-time video streaming between robots and their control stations, as described in [26], have emphasized that control signal transmission and video feedback impose strict requirements on network bandwidth and latency, with measurements showing that typical video latencies in LAN environments range between 50-150 ms.

These findings underscore the importance of evaluating ROS2-based systems under latency constraints—an area our research addresses the latency's impacts on multi-robot system.

1.4.3 Trajectory Coordination in Multi-robot system

Trajectory planning for multi-agent systems has been approached from various angles, including classical search-based algorithms, learning-based strategies, control-theoretic frameworks, and hybrid methods. Each of these paradigms aims to balance safety, scalability, and real-time feasibility while addressing the challenges posed by dynamic and uncertain environments.

Traditional graph-based methods such as A^* remain relevant due to their simplicity and effectiveness in structured environments. [20] utilized A^* for trajectory generation and incorporated a traffic heat map to balance congestion and improve flow in

dense multi-agent scenarios.

To handle the uncertainties of the real world and learn adaptive behaviors, Reinforcement Learning (RL) has gained attention. The research [27] and [28] demonstrated the application of RL for socially aware and collision-free navigation in dynamic, human-populated environments.

Hybrid approaches have also emerged to combine the strengths of model-free and model-based methods. [29] proposed a framework integrating RL with Model Predictive Control (MPC), allowing for adaptive planning with constraint handling. Similarly, the work [30] combined RL with Control Barrier Functions (CBFs) to enforce safety constraints during policy learning.

CBF-based methods, in particular, have been used to provide formal safety guarantees in decentralized settings. As demonstrated in [31], the researchers presented a scalable approach that ensures provably collision-free behavior by modifying nominal controllers to satisfy safety constraints in real-time.

Among the most widely adopted frameworks, optimization-based MPC has proven especially effective for dynamic and multi-agent environments. The research [32] proposed a decentralized MPC for navigating in dynamic scenes. In [33], the researchers further applied DMPC to enable multi-robot coordination, while the studies conducted in [34] and [35] extended similar DMPC strategies to handle dynamic obstacle avoidance and trajectory re-planning under latency and resource constraints.

1.4.4 Trajectory Tracking

Trajectory tracking methods for autonomous vehicles are generally divided into geometric and model-based approaches, as categorized in the research presented in [36].

Geometric methods rely on the spatial relationship between the vehicle and the reference path. They are simple, efficient, and do not require detailed vehicle models. Among them, Pure Pursuit is widely used for its ease of implementation and reasonable performance. However, according to the analysis in [37], its accuracy is sensitive to the choice of look-ahead distance, especially in dynamic environments.

Model-based methods use kinematic or dynamic models to enhance tracking accuracy and robustness. Though computationally more demanding, they are better suited for real-world scenarios. For example, as demonstrated in the work described in [38], Linear Quadratic Regulator (LQR) control has shown effective performance in minimizing tracking errors by optimizing control inputs.

1.5 Contributions

In this paper, we present several contributions to the construction of cloud robotics and multi-robot systems:

- **Multilayer Control Architecture:** We design and implement a hierarchical control system that operates at both local robot and cluster levels. This architecture allocates control responsibilities between local robots and cloud infrastructure based on network conditions and computing resources.

- **Switchable Trajectory Generation:** We provide an integrated DMPC and path sampling method for trajectory generation.

- **Flexible Trajectory Tracking Strategies:** We design a switchable trajectory tracking approach by using hybrid LQR and pure pursuit depending on the operational context.

- **Well-designed Local Decision-making:** We develop multiple error-handling process that ensures safe robot operation even during communication disruptions. This approach provides fail-safe collision avoidance capabilities that remain effective despite intermittent or delayed cloud connectivity.

- **Comprehensive Simulation Platform:** We develop an integrated simulation environment that combines RViz as the observer, Gazebo providing physics simulation, and an automated testing module.

1.6 Thesis Outlines

The remainder of this thesis is structured as follows.

Chapter 2: System Architecture and Communication. Introduces the hierarchical framework for distributed multi-robot coordination, detailing the Cloud, AMR, and Simulator components. It focuses on communication design and latency management to enable robust, multi-level decision-making.

Chapter 3: Modeling and Control. Presents the mathematical models, trajectory optimization, and collision avoidance strategies. It compares centralized and distributed control approaches, and introduces a hybrid Pure Pursuit–LQR method for adaptable, real-time trajectory execution.

Chapter 4: Evaluation and Results. Describes the AutoTest framework and test cases used to assess coordination performance under varying latency conditions. Results highlight the system's robustness in terms of success rate, path accuracy, and control stability.

Chapter 5: Conclusion. Summarizes contributions, answers the research questions, discusses current limitations, and outlines future work directions in latency-tolerant multi-robot coordination.

2

System Architecture and Communication

This chapter presents a comprehensive framework and architectural design for achieving robust coordination among multiple robots under realistic constraints. We begin by discussing the limitations of traditional sequential processing paradigms in multi-robot research and introduce our parallel decision-making approach that more accurately reflects real-world operating conditions. After that, we examine various distributed system architectural paradigms before detailing our hierarchical implementation, which optimizes information flow and computational efficiency. We then explore our communication design, including inter-component communication topics, communication patterns, and solutions to distributed communication challenges. The chapter concludes with a detailed examination of our three primary system components: the Cloud Component for centralized coordination and trajectory planning, the AMR Component for local robot control, and the Simulator Component for realistic testing and validation. By adopting a hierarchical and communication-aware design, we ensure that the system remains scalable, resilient, and capable of real-time coordination in dynamic environments.

2.1 Distributed Multi-Robot System

For the sake of simplicity in system design, researchers often develop and test multi-robot cluster control algorithms using sequential processing paradigms, wherein robots are processed one after another even when there are no logical dependencies between their operations, as observed in the work described in [32]. However, such approaches fail to reflect real-world operating conditions. Current implementations process robots serially within each time step, allowing each robot to access the complete state information of previously processed robots when making planning decisions. According to the principles outlined in [39], this artificial sequential information flow is unrealistic due to the inherently parallel nature of distributed systems, communication delays, and asynchronous state updates that occur in physical deployments. In real-world scenarios, each robot operates simultaneously with others, possessing only limited sensing capabilities and requiring explicit communication mechanisms to coordinate actions.

To address these limitations of sequential processing approaches, we propose a distributed multi-agent communication framework that realizes truly parallel decision-

making processes, wherein all robots simultaneously complete planning and control phases based solely on locally available information. This paradigm shift from sequential to parallel processing more accurately reflects the concurrent nature of real multi-robot systems, as emphasized in [18]. At the same time, we have made customized adjustments for different modules and communication topics for explicit information flow. This approach not only better approximates real-world operating conditions but also provides a more robust testbed for evaluating the resilience and adaptability of multi-robot coordination strategies under realistic constraints.

To better understand the context of our design choices, it is essential to examine the spectrum of available architectural approaches for distributed systems. As explored in the research presented in [40], distributed multi-agent systems can be implemented through various architectural paradigms, each with distinct characteristics suited for different operational contexts.

Figure 2.1 illustrates the fundamental topological differences between centralized and decentralized architectures, highlighting how information flows and control decisions propagate differently across these paradigms. The centralized structures like Master-Slave and Star configurations (left) concentrate all critical information and decision-making authority in one or few central nodes, while decentralized approaches (right) distribute both information and control authority across multiple interconnected nodes.

As further analyzed in [18], these architectural paradigms present different advantages and limitations for real-world multi-robot deployments, summarized in Table 2.1.

Table 2.1: Comparison of Centralized and Decentralized Architectural Paradigms

Aspect	**Centralized Architectures**	**Decentralized Architectures**
Pros	Simplified control Strong consistency guarantees Reduced coordination overhead	Better scalability No single point of failure Localized decision-making
Cons	Single point of failure Communication bottlenecks High dependency on central node	Increased coordination complexity Potential consistency challenges Higher communication overhead

The choice between these architectural paradigms involves critical trade-offs among reliability, scalability, communication overhead, and decision-making efficiency, with the selection depending on specific application requirements and deployment constraints.

Our implementation specifically leverages a hierarchical architecture, which, as presented in the research described in [41], offers several critical advantages for distributed multi-agent control systems. This architectural choice enables effective decomposition of control tasks into state management, trajectory planning, and trajectory tracking through multi-level organization, where higher-level agents coor-

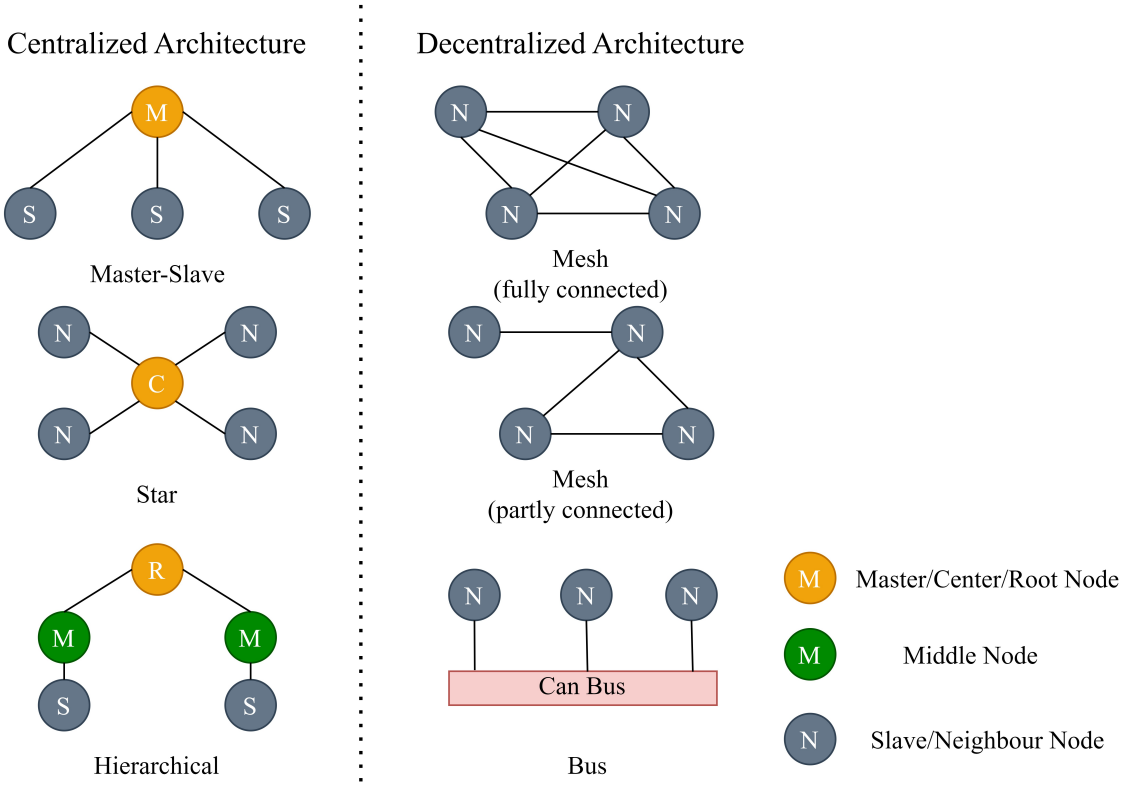


Figure 2.1: Topology of Centralized Architecture and Decentralized Architecture in Distributed Multi-agent Systems

dinate strategic decisions while lower-level agents focus on tactical execution. This separation of concerns significantly reduces computational load at some nodes and reduce communication demands, as nodes only interact with the upper and lower levels of their hierarchy, rather than interacting with the entire network. Furthermore, this architecture also provides natural scalability by allowing additional nodes to be incorporated at appropriate hierarchical levels without requiring system-wide reconfiguration. This brings convenience to a different number of robot control tasks. In our implementation, we design the explicit information flow between hierarchical levels, ensuring that agents maintain sufficient situational awareness while avoiding the excessive communication burden characteristic of fully connected mesh networks.

2.2 System Communication Design

The communication architecture is a critical component of our distributed multi-robot system, providing the foundation for coordination and information exchange between various modules. This section details the communication design principles, implementation choices, and performance considerations that underpin our system, including inter-component communication topics, communication patterns, distributed challenges and solutions, and advantages of the ROS2 framework.

Figure 2.2 illustrates the structure of the Cloud Component and its internal and external message flows, showcasing the hierarchical communication pattern that enables efficient coordination among multiple robots.

2.2.1 Inter-Component Communication Topics

Our distributed system architecture implements a comprehensive set of communication pathways that connect different components both externally and internally. These pathways are realized through specialized ROS2 topics and services that facilitate different types of information exchange:

- **Cloud Component Internal Communication:**
  - `/cluster_{robot_id}/state`: Transmits processed robot state from Cluster nodes to the Manager
  - `/manager/robot_states`: Broadcasts consolidated state information from Manager to all Cluster nodes
  - `/manager/global_start`: Synchronizes system initialization across all Cluster nodes
  - `/register_robot`: Service interface for robot registration with the Manager
- **Cloud-to-AMR Communication:**
  - `/cluster_{robot_id}/trajectory_delayed`: Delivers optimized trajectories from Cluster nodes to individual robots
  - `/cluster_{robot_id}/heartbeat_delayed`: Maintains connection status between Cluster and Robot nodes
- **AMR-to-Cloud Communication:**
  - `/robot_{robot_id}/state`: Reports robot state information including position, velocity, and operational status
  - `/robot_{robot_id}/heartbeat`: Confirms robot operational status and connection health
- **AMR-to-Simulator Communication:**
  - `/robot_{robot_id}/command`: Service interface for sending velocity commands to the simulated robot
- **Simulator-to-Cloud Communication:**
  - `/robot_{robot_id}/sim_state_delayed`: Provides simulated robot state information to the Manager
- **Simulator-to-AMR Communication:**
  - `/robot_{robot_id}/odom`: Provides simulated odometry data
  - `/robot_{robot_id}/robot_collision`: Reports collision detection events
  - `/robot_{robot_id}/f_scan`: Front-facing laser scan data for obstacle detection
  - `/robot_{robot_id}/b_scan`: Rear-facing laser scan data for obstacle detection

The communication architecture reflects the hierarchical nature of our distributed system. The Manager node serves as the central coordination hub within the Cloud

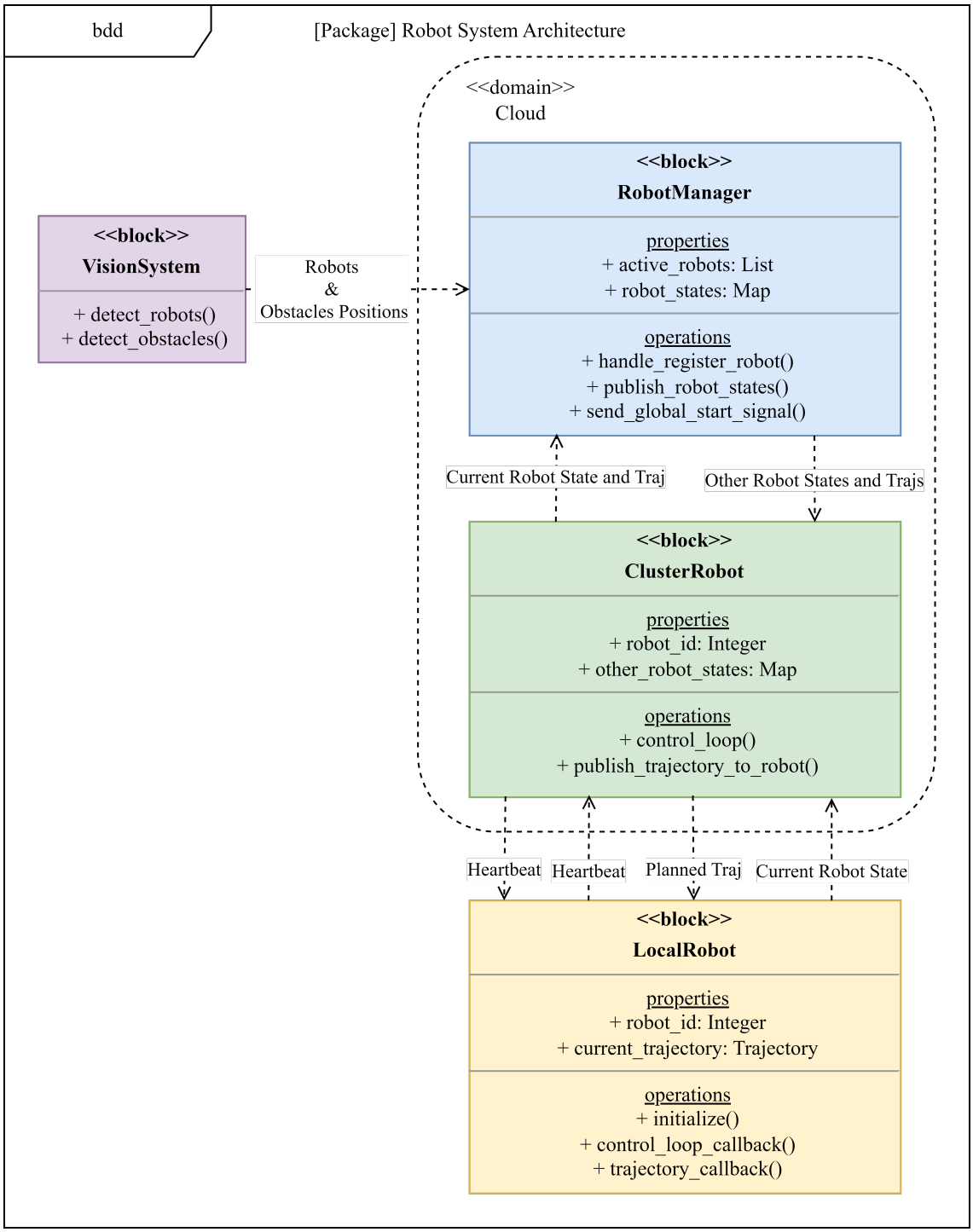


Figure 2.2: Topology of Cloud Component and internal/external message flow

Component, handling robot registration, consolidating state information from multiple sources, and broadcasting synchronized state information to all Cluster nodes. Each Cluster node is paired with a specific Robot node, forming a dedicated control channel responsible for trajectory planning and execution for that robot.

The bidirectional heartbeat mechanism between the Cluster and Robot nodes provides connection monitoring. When heartbeats are missed, robots can autonomously enter disconnecting handling modes while attempting to restore communication, ensuring operational safety even during network disruptions.

2.2.2 Communication Patterns

Beyond simply listing the communication topics, it is important to understand the different communication patterns employed in our system and how they address various distributed robotics challenges:

- **Publisher-Subscriber Pattern:** The majority of our system communication utilizes this asynchronous pattern, which decouples message publishers from subscribers and facilitates many-to-many relationships. This pattern is particularly valuable for real-time state updates, sensor data broadcasts, and trajectory distributions, where multiple components may need the same information simultaneously. For example, robot state information is published by individual robots and consumed by both the Manager and the visualization system without requiring direct point-to-point connections.

- **Service-Based Communication:** For operations requiring immediate confirmation or atomic transactions, we employ ROS2's service-based communication. The robot registration process exemplifies this pattern—each robot initiates registration through a service call to the Manager, which responds with confirmation only after successfully establishing the necessary communication channels. This synchronous pattern ensures system consistency during critical setup phases.

- **Parameter-Based Configuration:** System-wide settings and calibration values are managed through ROS2's parameter server, providing a centralized configuration mechanism. This pattern facilitates dynamic reconfiguration without requiring component restarts and ensures configuration consistency across the distributed system.

Each communication pattern is selected based on the specific requirements of the information exchange, considering factors such as timing constraints, delivery guarantees, and relationship complexity between senders and receivers.

2.2.3 Distributed Communication Challenges and Solutions

Implementing communication in distributed multi-robot systems presents several unique challenges that we have addressed with specific design solutions:

- **State Consistency:** Maintaining a consistent view of the system state across distributed components is challenging due to communication delays and message ordering issues. Our architecture addresses this through a state version tracking mechanism where each state update includes a sequence identifier. The Manager implements a state reconciliation algorithm that merges updates based on both timestamps and sequence numbers, resolving conflicts when needed.

- **Communication Failures:** Unreliable communication presents a significant risk in distributed robotics applications. Our system implements a multi-tiered reliability strategy. Using bidirectional heartbeat mechanisms, detect connection failures between paired components (Cluster-Robot, Manager-Cluster). Besides, Safety-first policies trigger emergency stops when critical communication failures persist beyond threshold durations.

2.2.4 ROS2 Framework and Quality of Service Advantages

Our system leverages ROS2 as the communication framework, which offers several significant advantages over ROS for distributed multi-robot applications. A key strength of ROS2 is its support for DDS and QoS policies, enabling fine-grained control over communication behavior that is essential for robust multi-robot coordination.

- **Built-in Data Distribution Service (DDS):** Unlike ROS's centralized master architecture, ROS2 uses DDS middleware that enables true peer-to-peer communication without a single point of failure, as demonstrated in the work described in [42]. This distributed architecture is essential for robust operation in multi-robot systems where components may join or leave dynamically.

- **QoS Controls:** ROS2 introduces comprehensive QoS settings that allow fine-grained control over communication characteristics. This capability is crucial for our system, as different types of messages have vastly different requirements regarding reliability, timeliness, and persistence.

- **Native Multi-threading Support:** ROS2's executor model enables true parallel processing within nodes, allowing our system to efficiently handle multiple concurrent operations such as state processing, trajectory planning, and communication management.

Our system strategically employs different QoS profiles for various communication

channels based on their specific requirements. Table 2.2 summarizes the QoS profiles used in our system and their application contexts.

Table 2.2: Comparison of QoS Profiles Used in the Multi-Robot System

QoS Profile	Key Characteristics	Primary Use Cases
Volatile Reliable	Reliable delivery & No history retention	• Robot state updates • Position broadcasts • Status information
Transient Local Reliable	Reliable delivery & History caching	• Map data • Obstacle information • Reference trajectories
Best Effort	No delivery guarantees & Minimal overhead	• Sensor data streams • Heartbeat messages • Non-critical telemetry

The combination of ROS2's distributed architecture and configurable QoS profiles has proven essential for our multi-robot system's resilience. For example, using Transient Local Reliable QoS for reference trajectories ensures that robots can recover planned paths even after brief communication interruptions, while Best Effort QoS for heartbeat messages minimizes network overhead during normal operation. These architectural advantages make ROS2 significantly more suitable than ROS for distributed multi-robot systems, particularly in scenarios requiring robust communication under variable network conditions.

2.3 Modules Design

The proposed architecture comprises three primary components that work together to create a comprehensive distributed multi-robot system with realistic communication characteristics:

- **Cloud Component:** This component is centralized around a singleton Manager class that coordinates the operation of multiple robot instances. The Robot Manager node serves as the central coordination hub for the distributed robotic system, fulfilling several critical functions: it is responsible for registration and management of all robot states, broadcasting global initialization messages, and performing trajectory planning tasks. The Cluster Robot module within this component is mainly responsible for receiving robot status and dynamic obstacle information, executing DMPC for trajectory planning, and sending the resulting trajectories to the corresponding AMR nodes.

- **AMR Component:** This component consists of several local robot nodes that perform trajectory tracking and local control based on planning informa-

tion received from the Cluster Robot module.

- **Simulator Component:** This component simulates responses to control requests in physical environments and emulates sensor information available to robots, such as odometry and Laser data.

2.3.1 Cloud Component

The Cloud Component forms the central coordination hub of our distributed system, providing high-level planning, state management, and inter-robot coordination. This section details the two main elements of this component: the Robot Manager that oversees the entire system, and the Cluster Robot nodes that handle individual robot planning.

2.3.1.1 Robot Manager

The core of the Cloud Component is the Robot Manager, which undertakes the following main tasks:

- **Robot Registration:** Handles robot registration through a service interface, validating requested robot IDs against expected configurations and maintaining a registry of active robots. When a robot registers successfully, the manager dynamically creates a corresponding cluster node, establishing necessary communication channels.

- **State Management:** Implements a comprehensive state management system that collects robot state information from multiple sources (cluster nodes and simulation converter nodes), reconciles potentially conflicting state data based on timestamps, and publishes consolidated state information to all connected components.

- **Global Coordination:** Provides global coordination through a start signal broadcasting mechanism. Once all expected robots have successfully registered, it transmits a synchronized start command to initiate coordinated operations.

The implementation leverages ROS2's communication patterns with quality of service profiles that ensure reliable message delivery. Thread safety is maintained through mutex locks that protect shared resources during concurrent operations. The multithreaded executor design allows the manager to handle multiple operations simultaneously, including robot registration, state updates, and publishing.

2.3.1.2 Cluster Robot

The Cloud Component also includes a dedicated ClusterNode implementation that functions as a bridge between the centralized Manager and individual robot controllers. When a local robot registers with the system, the Manager spawns a corresponding ClusterNode instance that handles trajectory planning and coordination for that specific robot.

The ClusterNode architecture encompasses several key functionalities:

- **Communication Management:** Maintains bidirectional communication channels with both the Manager and its associated robot, enabling efficient information flow.

- **Trajectory Planning:** Implements local trajectory planning capabilities by integrating path coordination with DMPC. Upon receiving the global start signal, the node initializes planning components and generates optimal trajectories by considering both static obstacles, dynamic obstacles, and predicted movements of other robots.

The control loop operates at a configurable frequency, where each iteration follows a systematic process:

1. Retrieval of current state information
2. Calculation of local reference trajectories
3. Acquisition of other robots' states
4. Execution of the DMPC controller to generate collision-free trajectories
5. Publication of the resulting trajectory to the robot controller

To systematically govern the behavioral logic of the cluster robot, a Finite State Machine (FSM) is employed as illustrated below. This FSM formalizes the robot's operational workflow by encapsulating its behavior into discrete, well-defined states. Such an approach enhances modularity, facilitates state-dependent control strategies, and ensures deterministic transitions across various execution phases—from initialization and task execution to completion.

While the FSM governs the high-level behavioral transitions of the cluster robot, it is equally important to detail the underlying control procedure executed within the operational states—particularly the Running state. This control procedure outlines the internal logic and data flow that enable the robot to perceive its environment, coordinate with fleet-level directives, and generate safe, executable trajectories in real time. The following activity diagram presents a structured view of this multi-channel control architecture, which ensures robust decision-making and adaptive motion planning in dynamic and potentially uncertain environments.

The design of the control procedure for the Node Cluster Robot effectively balances computational power and coordination across the various subsystems. By structuring the system into four channels—Fleet Coordinator, Path Generation, Decision-Making, and MPC Solver—the design allows for dynamic allocation of resources based on the operational context. The Fleet Coordinator and Path Genera-

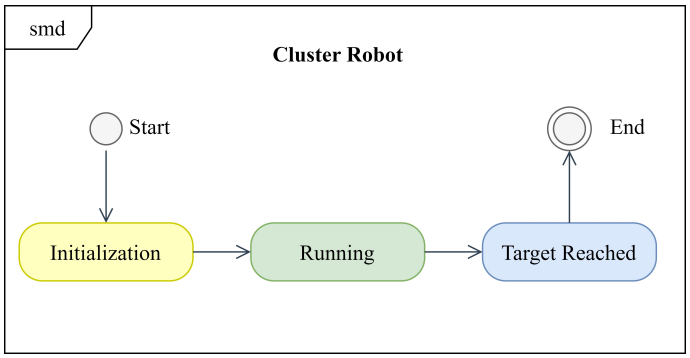


Figure 2.3: FSM of the Node Cluster Robot. In the **Initialization** state, the node loads all necessary information from the *RobotManager* and establishes communication with the local robot. During the **Running** state, it primarily handles the trajectory generation procedure. Upon detecting that the robot has reached its target, the node transitions to the **Target Reached** state, indicating readiness to terminate the process.

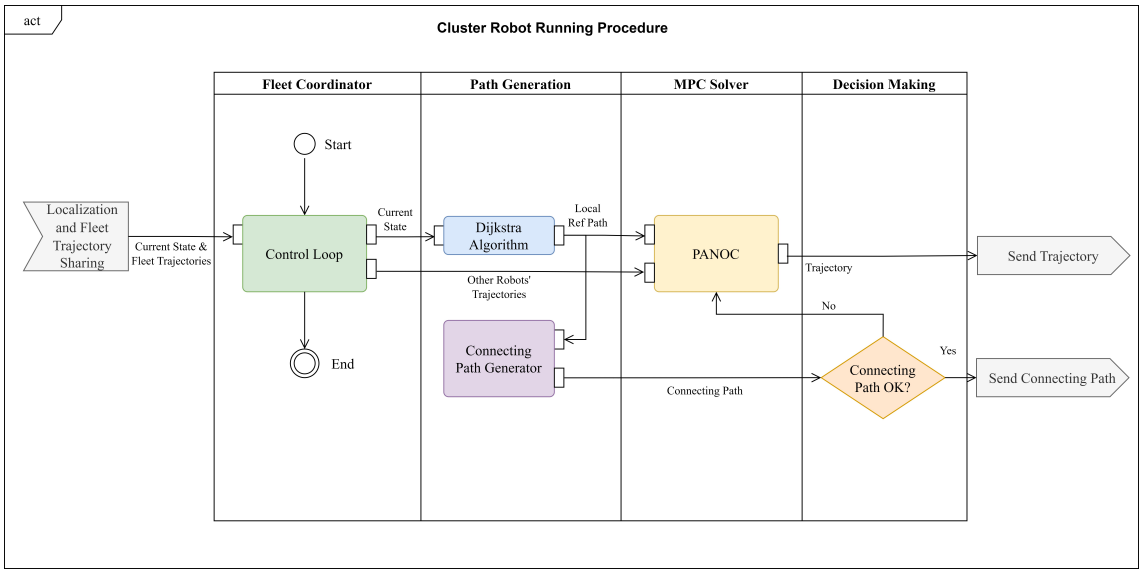


Figure 2.4: Control Procedure of the Node Cluster Robot. Once the control loop is active, four sequential channels are executed to enable coordinated multi-robot trajectory planning and generation. The **Fleet Coordinator** channel processes information from both the local robot and the *RobotManager*. The **Path Generation** channel then constructs a reference path based on a pre-defined navigation network and current robot states, along with a connecting path to the reference. Next, the **Decision-Making** channel evaluates potential static or dynamic obstacles along the planned path and within a forward semi-circular region around the robot. If the path is deemed safe, the connecting path is sent directly to the local robot for tracking. Otherwise, the system transitions to the **MPC Solver** channel, where PANOC(Proximal Averaged Newton-type method for Optimal Control) is invoked to compute a collision-free trajectory, which is subsequently sent to the local robot for execution.

tion channels leverage pre-defined information and current states to generate paths efficiently, minimizing computational load during steady operations. The decision-making process, which assesses the presence of static and dynamic obstacles, serves as a critical switch between computational modes. If no obstacles are detected, the system follows a low-complexity path tracking mode, maintaining system efficiency, especially in multiple robot scenarios. However, in the presence of obstacles, the MPC Solver channel is activated, leveraging the PANOC solver to compute a collision-free trajectory. This switchable mode design ensures that the robot is capable of maintaining high computational efficiency during routine tasks while also dynamically scaling computational power when the environment presents challenges, thus optimizing overall system performance and coordination.

2.3.2 AMR Component

The AMR Component serves as the final layer in the distributed system architecture, implementing the interface between the Cloud Component's planning capabilities and physical robot control. This implementation follows a client-server model where each robot node operates independently while maintaining synchronization with its corresponding cluster node.

The RobotNode is designed with a comprehensive initialization process that includes registration with Robot Manager in Cloud Component, establishment of communication channels with its dedicated Cluster node, and configuration of control parameters based on robot specifications. Once initialized, it implements a bidirectional communication pattern, receiving trajectory commands from the Cluster while providing state updates derived from physical sensors or simulation.

Key features of the AMR Component implementation include:

- **Switchable Controller:** Implements various control algorithms (Pure Pursuit, LQR) that are switchable based on different trajectory types.

- **Trajectory Tracking:** Implements trajectory tracking controllers that transform high-level trajectory plans into low-level control commands suitable for the robot's motion model.

- **Error Handling:** Implements logic that ensures received paths are physically feasible given the robot's current state, executing emergency stops when necessary. Implements local obstacle detection to ensure safety. Handles timing differences between planning cycles and control execution to maintain system stability.

The control loop on each local robot node operates at a configurable frequency. In each iteration, it performs trajectory interpretation, selects the appropriate controller, computes control commands subject to relevant constraints, and executes them through a command service interfacing with either physical or simulated robot hardware.

The associated FSM provides a structured and fault-tolerant framework for managing the robot's operational states, ensuring robust coordination with the cloud system and safe, reliable execution of assigned trajectories.

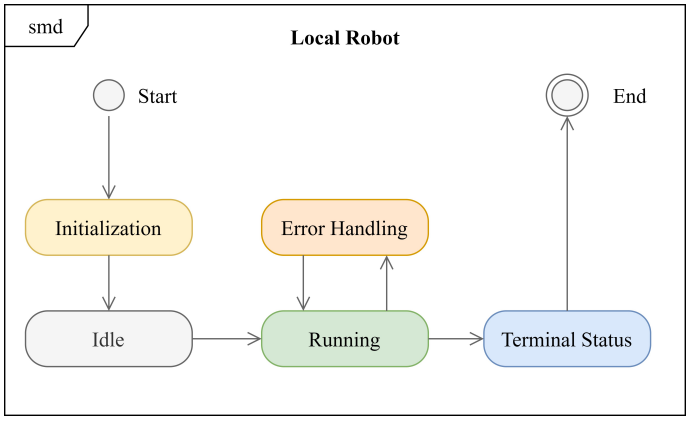


Figure 2.5: FSM of the Node-Local Robot. The node-local robot is designed with multiple operational states. During the **Initialization** state, the robot registers with the cluster, establishes communication with the cloud-level robot, and connects to the simulator environment. Upon completion, it transitions to the **Idle** state, where it waits for a trajectory to be published by the cluster robot. Once a trajectory is received, the FSM moves to the **Running** state. From this state, the system may transition to **Error Handling** in the case of faults, or to a **Terminal** state upon reaching the target or experiencing a collision—both of which conclude the execution in the **End** state.

The Local Robot's FSM Figure 2.5 presents the high-level operational states and transitions of the node-local robot, while the control procedure in Figure 2.6 details the internal logic executed within each state, especially during Running. The FSM defines when transitions occur, and the control procedure defines how actions are performed, including sensor processing, error detection, and control execution. Together, they provide a clear and comprehensive view of both state management and real-time onboard decision-making.
The node-local robot receives input from both the cluster robot (e.g., trajectory information) and the simulation environment (e.g., odometry and LaserScan data). The control process begins in the Local Controller channel, where sensor data is used to identify nearby obstacles within a predefined elliptical region, analyze the received trajectory type, and compute the minimum distance to avoid outdated trajectories. Additionally, a heartbeat mechanism is used to monitor the connection status with the cluster robot.

In the Decision-Making channel, these inputs are evaluated to determine if any error conditions are present. Depending on the situation, the system may transition to one of several error-handling procedures in the Error Handling channel: Safety Stop for nearby obstacles, Emergency Stop for distant trajectories, and Disconnect Handling when no heartbeat is received for a predefined duration (currently 2 seconds).

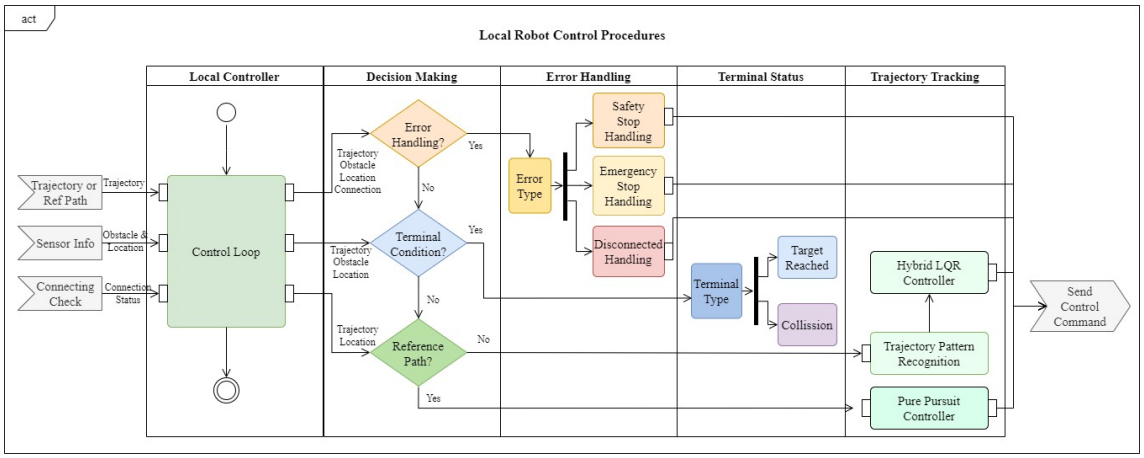


Figure 2.6: Control Procedures of the Node-Local Robot. This figure outlines the onboard decision-making and control flow based on inputs from both the cluster robot and the simulation environment.

If none of these conditions are met, the system checks for terminal states, such as reaching the goal or detecting a collision. Otherwise, it proceeds to the Trajectory Tracking channel, where a pure-pursuit algorithm is used for path tracking and a hybrid LQR controller for full trajectory tracking. The resulting control commands are then sent to the simulation environment. Further details of the tracking algorithms are discussed in the following chapter.

The design of the local robot highlights the clarity and modularity of the onboard control architecture. The FSM ensures well-defined state transitions, while the control procedure delineates the real-time logic executed within each state. Notably, the architecture supports a switchable controller design, allowing the system to dynamically choose between path tracking and trajectory tracking strategies—such as pure pursuit and hybrid LQR—based on the operational context. This flexibility enables robust performance across varying task requirements and environmental conditions.

2.3.3 Simulator Component

The Simulator Component provides a realistic environment for testing the distributed multi-robot system without requiring physical hardware. This component emulates the physical characteristics and sensing capabilities of real robots, allowing for comprehensive validation of the system architecture before deployment on actual hardware. Through this simulation environment, we can evaluate the performance of our distributed communication framework under various scenarios and identify potential issues before real-world implementation.

The Simulator Component consists of three integrated parts: Gazebo, Robot Visualization (RViz), and Message Buffer:

- **Gazebo Simulation:** Serves as the primary physics-based simulation engine that implements realistic robot dynamics, sensor simulation (odometry, Laser), and environmental interactions. Gazebo handles all physics calculations including collision detection, motion dynamics, and sensor data generation.

- **RViz:** Functions as a real-time visualization system that renders and analyzes critical system information including map data, robot states, trajectories, and sensor ranges. This visualization module provides both qualitative visual feedback and quantitative performance metrics for comprehensive system evaluation.

- **Message Buffer:** Acts as a communication middleware that caches and forwards messages on designated topics to simulate real-world communication challenges such as network delays, packet loss, and out-of-order message arrival. This component is crucial for evaluating the robustness of the distributed communication framework under realistic network conditions.

2.3.3.1 Gazebo Simulation

The Gazebo implementation handles the dynamic creation and initialization of robot entities within the simulation environment. This module leverages ROS2's node architecture and service-based communication to interact with the Gazebo simulation engine. Key aspects of the Gazebo implementation include:

- **Custom Interface Plugin Integration:** The implementation incorporates differential drive controllers and contact sensor plugins. These plugins enable the translation of high-level trajectory commands into low-level joint controls and the generation of sensor data (odometry, collision detection) that mirrors real-world robot capabilities.

- **Converter Node Management:** For each successfully spawned robot, the system automatically initializes a dedicated converter node that transforms Gazebo's native message formats into standardized formats expected by the AMR and Cloud Components. These converter nodes run as separate processes to maintain system modularity and robustness against individual component failures.

2.3.3.2 Robot Visualization

The RViz visualization module is implemented through the `RobotStateVisualizer` class that provides comprehensive real-time visualization and analytical capabilities. This component leverages ROS2's publisher-subscriber architecture to collect, process, and display various aspects of the multi-robot system. Key features of the RViz implementation include:

- **Multi-layered Visualization System:** The visualization is organized into distinct marker arrays with different update frequencies—static environmental elements (low frequency), robot states (high frequency), path histories (medium frequency), planned trajectories (medium frequency), and reference paths (low frequency).

- **Status-Based Visual Encoding:** Robot states are visually encoded using a color-coding scheme that reflects their operational status (initializing, running, emergency stop, target reached, etc.), providing immediate visual feedback on system conditions.

- **Temporal Path Analysis:** The system maintains and visualizes historical path data with configurable resolution, allowing for detailed analysis of robot movements over time.

- **Automated Performance Metrics:** The RViz module automatically calculates and publishes quantitative performance metrics such as path deviation, execution time, path length, and motion smoothness. These metrics enable objective comparison between different control algorithms and parameter configurations.

The simulation framework implements proper error handling and logging mechanisms to facilitate debugging and provide transparency into the simulation state. Thread management ensures clean termination of all spawned processes when the simulation is stopped, preventing resource leaks and system instability during repeated testing cycles.

2.3.3.3 Message Buffer

The Message Buffer component implements a communication middleware layer that simulates realistic network conditions in multi-robot systems. Key aspects of the Message Buffer implementation include:

- **Topic-specific Delay Simulation:** The system applies configurable delay models to each communication topic independently. The buffer accurately simulates the stochastic nature of real-world network latencies. Messages are captured, timestamped with a future release time, and stored in priority queues until their scheduled delivery time.

- **Interactive Configuration Interface:** The implementation includes an optional graphical user interface that allows researchers to dynamically adjust delay parameters for individual topics or globally across all communication channels.

- **Asynchronous Message Processing:** The implementation utilizes ROS2's reentrant callback groups and high-frequency timers to ensure non-blocking,

asynchronous message handling. This design allows the buffer to process thousands of messages per second while maintaining accurate delay simulation, even under high communication loads that might occur during complex multi-robot coordination scenarios.

This comprehensive simulation environment, with its three integrated components (Gazebo for physics simulation, RViz for visualization, and Message Buffer for realistic communication delay simulation), provides an ideal testbed for evaluating our distributed multi-robot system. By simulating realistic sensor data, physical interactions, and communication dynamics, the Simulator Component enables thorough validation of both individual robot control algorithms and collective coordination strategies before deployment on physical hardware. Furthermore, the integrated performance evaluation capabilities allow for quantitative comparison between different system configurations, providing valuable insights for optimization and refinement of the distributed architecture.

3
Modeling and Control

Modern AMR fleets require a robust control framework that integrates cloud-based fleet-wide coordination, trajectory generation, and local trajectory tracking. An effective solution must balance three critical objectives:

- **Collision-free coordination** – Ensuring safe and cohesive navigation in dynamic environments.
- **Computational efficiency** – Enabling real-time operation for large-scale fleets.
- **Robust trajectory tracking** – Mitigating the impact of latency in real-world deployments.

This chapter establishes a mathematical foundation to address these challenges using a multi-layer control framework. The proposed approach integrates distributed nonlinear DMPC for high-level coordination with trajectory tracking and motion control strategies for low-level execution.

The formulation is structured around two key components:

- **Fleet-wide collision avoidance coordination** – Ensuring safe and efficient navigation among AMRs.
- **Latency-tolerant local trajectory tracking and motion control** –Enhancing responsiveness in simulation environments with real-world dynamics effects like inertia and friction, etc.

3.1 Modeling of Robot

A discrete-time robot model is essential for aligning trajectory generation and tracking within the same computational framework. It reflects the inherently digital nature of real-time control systems, enabling accurate step-wise predictions, effective latency handling, and seamless integration with optimization-based methods. This consistency enhances robustness, responsiveness, and overall control performance.

The discrete-time kinematic motion model of an autonomous mobile robot at time step k is expressed as:

$$\mathbf{x}_{k+1} = f(\mathbf{x}_k, \mathbf{u}_k) \tag{3.1}$$

where $\mathbf{x}_k$ represents the state of the robot at time step k, including its current

position and orientation, see the definition in Figure 3.1, while $\mathbf{u}_k$ denotes the control input at time step k, adjusting the velocity and angular velocity.
In this thesis, the state vector is defined as

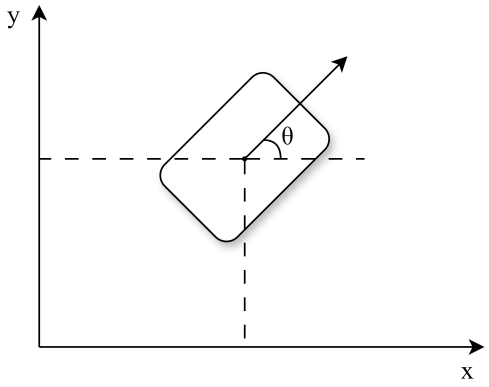


Figure 3.1: Robot States coordinate frame

$$\mathbf{x}_k = [x_k, y_k, \theta_k]^\top$$

where (x_k, y_k)represents the Cartesian coordinates of the robot, and θ_k is its heading angle.

The control input vector is given by

$$\mathbf{u}_k = [v_k, \omega_k]^\top$$

where v_k is the linear velocity and ω_k is the angular velocity.

To model the robot's motion over discrete time steps, we introduce the full state-space representation with a sampling time T_s. The system's state evolution follows:

$$\mathbf{x}_{k+1} = f(\mathbf{x}_k, \mathbf{u}_k) = \begin{bmatrix} x_k + T_s v_k \cos(\theta_k) \\ y_k + T_s v_k \sin(\theta_k) \\ \theta_k + T_s \omega_k \end{bmatrix} \tag{3.2}$$

This formulation describes the robot's kinematics, assuming constant velocity within each sampling interval. It forms the basis for motion prediction and control in the following sections. The differential drive model incorporates essential nonholonomic constraints while remaining computationally efficient for real-time optimization.

3.2 Trajectory Planning

Modern autonomous systems must navigate complex environments while maintaining strict safety margins and ensuring smooth motion. Our formulation addresses four key requirements: 1) Accurate motion prediction using kinematic modeling, 2) Optimal trajectory generation through cost minimization, 3) Comprehensive collision avoidance for various obstacle types, and 4) Real-time trajectory execution via

distributed MPC. This integrated framework enables robots to dynamically replan paths while adhering to physical constraints and interaction dynamics. As detailed in the system architecture chapter, this module operates within a cluster.

3.2.1 Reference Path

To determine the optimal reference path for each robot, we employ Dijkstra's algorithm, a fundamental graph search algorithm that efficiently solves the single-source shortest path problem for graphs with non-negative edge weights. This algorithm enables us to identify the most efficient route through the workspace while avoiding obstacles.

The mathematical formulation of Dijkstra's algorithm can be expressed by the relation:

$$d(s, v) = \min d(s, v), d(s, u) + w(u, v) \tag{3.3}$$

where $d(s, v)$ represents the shortest distance from source node s to node v, $d(s, u)$ denotes the shortest distance from source node s to node u, and $w(u, v)$ signifies the weight or distance of the edge connecting node u to node v. The algorithm initializes $d(s, s) = 0$ and $d(s, v) = \infty$ for all other nodes v, then iteratively refines these estimates until the optimal solution is reached.

During the trajectory tracking process, we implement a sampling strategy to obtain a local reference for each control loop. This local reference path serves as the foundation for the Model Predictive Control (MPC) algorithm and facilitates the connecting path generation: connection from the robot's current position to the first point in the local reference path.

These alternative connecting paths are evaluated based on predefined criteria, and the optimal path is selected for execution. The subsequent chapter will present a more detailed analysis of path selection and execution strategies.

3.2.2 Trajectory Optimization Objectives

To ensure smooth behavior while following given trajectories, the objective function consists of three penalty terms:

1. **Deviation from the reference trajectory:** Penalizes differences between the planned state and the reference state, which is formulated in function 3.4.
2. **Deviation from the reference control inputs:** Encourages the control inputs to remain close to the reference, which is formulated in function 3.5.
3. **Control smoothness:** Penalizes sudden changes in control inputs to promote smoother motion, which is formulated in function 3.6.

Given a planning horizon N, reference state $\tilde{\mathbf{x}}$, and reference input $\tilde{\mathbf{u}}$, the cost components are formulated as follows:

$$J_\tau(\tilde{\mathbf{x}}_k, \mathbf{x}_k) = \|\tilde{\mathbf{x}}_k - \mathbf{x}_k\|^2_{Q_k}, \quad Q_k = Q_0 \left(1 + \alpha \|\tilde{\mathbf{x}}_k - \mathbf{x}_k\|_2\right), \quad k \in \mathbb{N}_{[0,N]} \tag{3.4}$$

$$J_u(\tilde{\mathbf{u}}_k, \mathbf{u}_k) = \|\tilde{\mathbf{u}}_k, \mathbf{u}_k\|^2_R, \quad k \in \mathbb{N}_{[0,N-1]} \tag{3.5}$$

$$J_a(\mathbf{u}_{k+1}, \mathbf{u}_k) = \|\mathbf{u}_{k+1} - \mathbf{u}_k\|^2_S, \quad k \in \mathbb{N}_{[0,N-1]} \tag{3.6}$$

The overall cost for a single robot i at time step k is given by:

$$J_{R_i}(\tilde{\mathbf{x}}^{(i)}_k, \mathbf{x}^{(i)}_k, \tilde{\mathbf{u}}^{(i)}_k, \mathbf{u}^{(i)}_k, \mathbf{u}^{(i)}_{k+1}) = J_\tau(\tilde{\mathbf{x}}_k, \mathbf{x}_k) + J_u(\tilde{\mathbf{u}}_k, \mathbf{u}_k) + J_a(\mathbf{u}_{k+1}, \mathbf{u}_k) \tag{3.7}$$

For simplicity, the time index k may be omitted in later expressions if there is no ambiguity.

3.2.3 Collision Avoidance Formulation

In the AMR fleet, for a given mobile robot i at time step k, the state and control input vectors are denoted as $\mathbf{x}^{(i)}_k$ and $\mathbf{u}^{(i)}_k$, respectively. The objective is to plan the motion for a fleet of P robots, each governed by a given dynamic model $f(\cdot)$, ensuring collision-free navigation.

The motion planning must account for the following three categories of obstacles:

- **Other Robots ($\mathcal{F}$)** – The remaining robots in the fleet, which act as dynamic obstacles to each other.
- **Static Obstacles ($\mathcal{O}$)** – Fixed polygonal obstacles that must be avoided.
- **Dynamic Obstacles ($\mathcal{D}$)** – Moving elliptical obstacles representing objects or agents with time-varying positions.

The goal is to generate collision-free trajectories for all P robots while considering these factors.

1. **Collision Avoidance within the AMR Fleet ($\mathcal{F}$):** The mobile robots in the fleet are geometrically modeled as circles with radius r and center position $\mathbf{c}$. According to the approach presented in [43], the collision avoidance constraint within the fleet can be formulated in two ways:
   - *Hard Constraint:* The distance between the centers of any two robots i and j must be at least $2r$, ensuring no overlap:
   $$\|\mathbf{c}^{(i)} - \mathbf{c}^{(j)}\|_2 \geq (2r)^2. \tag{3.8}$$
   - *Soft Constraint:* A penalty function $J_\mathcal{F}$ is introduced to encourage a safe distance d_0 (set to $d_0 = 2r$ in simulations). The penalty cost is formulated as:
   $$J_\mathcal{F}(\mathbf{c}^{(i)}, \mathbf{c}^{(j)}) = Q_\mathcal{F} \left[d_0^2 - \|\mathbf{c}^{(i)} - \mathbf{c}^{(j)}\|_2\right]^2_+, \tag{3.9}$$

where $Q_{\mathcal{F}}$ is a weighting factor, and $[\cdot]_+$ denotes $[x]_+ = \max\{0, x\}$, ensuring that the cost is applied only when the distance is below d_0.

2. **Static Obstacles ($\mathcal{O}$):** Static obstacles are modeled as convex polygons, where the n-th obstacle is defined by the set:

$$H_n = \{\mathbf{p} \in \mathbb{R}^2 : \mathbf{b}_{n,m} - \mathbf{a}_{n,m}^\top \mathbf{p} > 0, \quad m \in \mathbb{N}[1, M]\} \tag{3.10}$$

where M is the number of inequalities defining the polygon, and $a_{n,m}$, $b_{n,m}$ are the coefficients of the linear constraints.

A point $\mathbf{p}$ is considered outside an obstacle H_n if the following constraint holds for all obstacles n:

$$\forall n \in \mathcal{N}[1, N_o], \quad \prod_{m=1}^{M} [h_{n,m}(\mathbf{p})]_+^2 = 0, \tag{3.11}$$

where $h_{n,m}(\mathbf{p}) = \mathbf{b}_{n,m} - \mathbf{a}_{n,m}^\top \mathbf{p}$.

For N_o obstacles, a soft obstacle avoidance cost is introduced with a penalty term $Q_{\mathcal{O}}$, formulated as:

$$J_{\mathcal{O}}(\mathbf{p}) = Q_{\mathcal{O}} \sum_{n=1}^{N_o} \prod_{m=1}^{M} [h_{n,m}(\mathbf{p})]_+^2. \tag{3.12}$$

Here, the penalty cost ensures that the robot maintains a safe distance from obstacles, with higher costs applied when the robot gets closer to the polygon boundaries.

3. **Dynamic Obstacles ($\mathcal{D}$):** Dynamic obstacles are modeled as ellipses, as their future positions need to be estimated using Gaussian distributions to account for uncertainty. As described in the research presented in [44], the shape of an elliptical obstacle is defined by the function:

$$e(\mathbf{p}) = \left[1 - (\mathbf{p} - \mathbf{c})^\top E (\mathbf{p} - \mathbf{c})\right]_+^2, \tag{3.13}$$

where $\mathbf{c}$ represents the center of the ellipse, E is a positive definite matrix defining the shape and orientation of the ellipse, ensuring that violations of the constraint contribute to the penalty.

A soft constraint is introduced to penalize proximity to dynamic obstacles. Given N_d dynamic objects, the penalty function is formulated as:

$$J_{\mathcal{D}}(\mathbf{p}) = \sum_{n=1}^{N_d} Q_{\mathcal{D}} \left[1 - (\mathbf{p} - \mathbf{c}_n)^\top E (\mathbf{p} - \mathbf{c}_n)\right]_+^2, \tag{3.14}$$

where $Q_{\mathcal{D}}$ is a weighting coefficient that determines the strength of the penalty.

This formulation ensures that robots maintain a safe distance from moving obstacles while incorporating uncertainty in their predicted locations.

3.2.4 Distributed MPC for Multi-Robot Coordination

The coordination of autonomous robot fleets in dynamic environments demands control architectures that balance computational efficiency with collaborative trajectory planning. Building on the formulation that we established robot motion model, the objective trajectory function, and the collision avoidance formulation, we now analyze distributed control strategies that address the limitations of centralized approaches.

Modern multi-robot systems require control paradigms that adapt to two critical challenges: (1) the combinatorial growth of collision constraints as fleet size increases, and (2) the need for real-time replanning in uncertain environments. As described in [45], MPC has emerged as a dominant methodology in process automation and robotics due to its inherent capability to handle constrained optimization problems through receding horizon control. MPC framework solves finite-horizon optimal control problems iteratively, executing only the immediate control action before recomputing with updated state information.

3.2.4.1 Centralized MPC Formulation

Given a fleet of P robots, denoted as $\mathcal{F} = \{R_1, \ldots, R_P\}$, and the set of all pairs of robot indices as

$$I = \{(i,j) \mid R_i, R_j \in \mathcal{F}, i \neq j\},$$

the centralized MPC is formulated as [43]:

$$\begin{aligned} \min_{\mathbf{u}} \quad & \sum_{k=0}^{N-1} \left(\sum_{i=1}^{P} \left[J_{R_i} + J_{\mathcal{O}}(\mathbf{p}_k^{(i)}) + J_{\mathcal{D}}(\mathbf{p}_k^{(i)}) \right] + \sum_{(i,j)\in I} J_{\mathcal{F}}(\mathbf{p}_k^{(i)}, \mathbf{p}_k^{(j)}) \right) \\ & + \sum_{i=1}^{P} J_\tau(\tilde{\mathbf{x}}_N^{(i)}, \mathbf{x}_N^{(i)}) \qquad (3.15) \\ \text{s.t.} \quad & \mathbf{u}_{\min} \leq \mathbf{u}_k^{(i)} \leq \mathbf{u}_{\max}, \\ & \mathbf{x}_{k+1}^{(i)} = f(\mathbf{x}_k^{(i)}, \mathbf{u}_k^{(i)}), \qquad (3.16) \\ & \forall k \in \mathbb{N}_{[0,N-1]}, \quad \forall i \in \mathbb{N}_{[1,P]} \end{aligned}$$

3.2.4.2 Distributed MPC Formulation

Distributed methods allocate computational tasks across multiple agents, enabling decentralized processing. In our scenario, each robot operates an independent solver, and predicted future states are exchanged among them. By sharing planned trajectories, robots can proactively identify and mitigate potential collisions. This coordination relies on position data obtained from a global camera system, ensuring accurate situational awareness.

The optimization problem presented below is formulated for a single mobile robot at a time. The cost function component $J_{\mathcal{F}}$ specifically penalizes deviations between the executing robot's current position and the anticipated locations $\hat{\mathbf{p}}_k^{(j)}$ of neighboring robots, promoting collision-free navigation.

$$\begin{aligned}
\min_{\mathbf{u}} \quad & \sum_{k=0}^{N-1} \left(J_R + J_{\mathcal{O}}(\mathbf{p}_k) + J_{\mathcal{D}}(\mathbf{p}_k) + \sum_{j=1}^{P} J_{\mathcal{F}}(\mathbf{p}_k, \hat{\mathbf{p}}_k^{(j)}) \right) \\
& + J_\tau(\mathbf{x}_N^{\text{ref}}, \mathbf{x}_N) && (3.17) \\
\text{s.t.} \quad & \mathbf{u}_{\min} \leq \mathbf{u}_k \leq \mathbf{u}_{\max}, \\
& \mathbf{x}_{k+1} = f(\mathbf{x}_k, \mathbf{u}_k) && (3.18) \\
& \forall k \in \mathbb{N}_{[0,N-1]},
\end{aligned}$$

3.2.5 Comparison of Centralized and Distributed MPC Formulations

Centralized Model Predictive Control (CMPC) offers a global perspective and potentially optimal coordination but suffers from scalability and robustness limitations. In contrast, DMPC promotes scalability and robustness at the expense of global optimality and coordination consistency. The Table 3.1 summarizes the key differences between these two formulations across several critical dimensions relevant to multi-agent control systems.

In summary, while centralized MPC guarantees optimal solutions for single-robot systems, its direct extension to multi-agent scenarios is hindered by fundamental scalability limitations. The joint state space grows as $O(P^3)$ for P robots, leading to computational bottlenecks that make real-time implementation impractical for large fleets. To address this challenge, DMPC has been developed, enabling scalable optimization by decomposing the global problem into localized subproblems while preserving inter-agent coordination through trajectory sharing.

3.3 Trajectory Tracking

This section presents multiple control methods for AMR local control, including Pure Pursuit and LQR-based strategies, to track trajectories provided by the cluster fleet coordinator. To ensure consistency between planning and execution, the same discrete-time robot model has been employed. Pure Pursuit offers smooth geometric tracking with adaptive speed control, while LQR provides optimal feedback based on system dynamics. By combining both in a hybrid strategy and also providing switchable function, we enhance tracking accuracy, responsiveness, and robustness for real-time autonomous navigation.

Table 3.1: Comparison of Centralized and Distributed MPC Formulations

Aspect	**CMPC**	**DMPC**
Computational Complexity	Optimization considers all robots simultaneously, leading to a high-dimensional problem; challenging for real-time execution as fleet size increases.	Each robot solves its own problem, reducing per-agent load; communication overhead from trajectory sharing exists.
Coordination and Collision Avoidance	Collision avoidance is enforced globally via joint optimization; it enables globally optimal solutions but requires solving a large-scale problem.	Collision avoidance through penalizing predicted vs. actual positions of others using shared data; prone to inconsistencies with delays or mismatches.
Scalability	Poor scalability as problem size grows exponentially with fleet size; limited by computational requirements.	Better scalability due to local optimization; network congestion and synchronization issues may arise with dense fleets.
Communication Requirements	Central unit gathers states, computes controls, and distributes results; demands high bandwidth and reliability.	Robots exchange predicted states and plans; lower bandwidth needs but sensitive to latency and packet loss.
Robustness to Failures	Single point of failure in central solver; failure disrupts the whole system.	More robust; individual robot failures don't necessarily affect the whole, though coordination may degrade.
Optimality and Performance	Globally optimal solutions possible; real-time performance can suffer due to computation limits.	Suboptimal but real-time feasible; performance depends on prediction accuracy and communication efficiency.

3.3.1 Pure Pursuit

Lookahead Point Selection

The lookahead distance L is a predefined parameter. The lookahead point (x_L, y_L) is the first waypoint in $\mathcal{T}$ that satisfies:

$$d_i = \sqrt{(x_i - x_k)^2 + (y_i - y_k)^2} \geq L, \tag{3.19}$$

where (x_k, y_k) is the current position of the vehicle.

Transformation to Vehicle Frame

The lookahead point is transformed into the vehicle's coordinate frame:

$$x'_L = (x_L - x_k)\cos(-\theta_k) - (y_L - y_k)\sin(-\theta_k), \tag{3.20}$$
$$y'_L = (x_L - x_k)\sin(-\theta_k) + (y_L - y_k)\cos(-\theta_k). \tag{3.21}$$

Curvature Calculation

The required curvature κ to follow the lookahead point is given by:

$$\kappa = \frac{2y'_L}{L^2}. \tag{3.22}$$

Control Inputs Computation

The control inputs are computed as:

$$v_k = v_{\max} \cdot e^{-\alpha|\kappa|}, \tag{3.23}$$
$$\omega_k = v_k \cdot \kappa, \tag{3.24}$$

where:

- $v_{\max}$ is the maximum allowable velocity,
- α is a tuning parameter that reduces speed during sharp turns.

This formulation implements a state-space-based Pure Pursuit controller that dynamically adjusts both linear and angular velocity to ensure smooth and stable path-following behavior.

3.3.2 Linear Quadratic Regulator (LQR)

The LQR controller is designed for trajectory tracking using a discrete-time state-space model of a unicycle.

The desired or reference state is:

$$\mathbf{x}_{\text{ref}} = \begin{bmatrix} x_{\text{ref}} \\ y_{\text{ref}} \\ \theta_{\text{ref}} \end{bmatrix}, \tag{3.25}$$

with the corresponding reference control input:

$$\mathbf{u}_{\text{ref}} = \begin{bmatrix} v_{\text{ref}} \\ \omega_{\text{ref}} \end{bmatrix}. \tag{3.26}$$

Defining the error state as:

$$\delta\mathbf{x}_k = \mathbf{x}_k - \mathbf{x}_{\text{ref}}, \tag{3.27}$$

and the control error is:

$$\delta\mathbf{u}_k = \mathbf{u}_k - \mathbf{u}_{\text{ref}}. \tag{3.28}$$

Linearization

Linearizing the state-space model about $(\mathbf{x}_{\text{ref}}, \mathbf{u}_{\text{ref}})$ yields the error dynamics:

$$\delta\mathbf{x}_{k+1} = A\,\delta\mathbf{x}_k + B\,\delta\mathbf{u}_k, \tag{3.29}$$

where the system matrices are:

$$A = \begin{bmatrix} 1 & 0 & -T_s\, v_{\text{ref}} \sin(\theta_{\text{ref}}) \\ 0 & 1 & T_s\, v_{\text{ref}} \cos(\theta_{\text{ref}}) \\ 0 & 0 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} T_s\, \cos(\theta_{\text{ref}}) & 0 \\ T_s\, \sin(\theta_{\text{ref}}) & 0 \\ 0 & T_s \end{bmatrix}. \tag{3.30}$$

Cost Function

The LQR controller minimizes the infinite-horizon quadratic cost function:

$$J = \sum_{k=0}^{\infty} \left(\delta\mathbf{x}_k^\top Q\,\delta\mathbf{x}_k + \delta\mathbf{u}_k^\top R\,\delta\mathbf{u}_k \right), \tag{3.31}$$

where Q is a positive semi-definite state weighting matrix, and R is a positive definite control weighting matrix.

Optimal Control Law

The optimal control correction is:

$$\delta\mathbf{u}_k = -K\,\delta\mathbf{x}_k, \tag{3.32}$$

where the feedback gain K is computed by solving the discrete algebraic Riccati equation (DARE):

$$P = A^\top PA - A^\top PB \left(B^\top PB + R \right)^{-1} B^\top PA + Q. \tag{3.33}$$

Then, the optimal gain is:

$$K = \left(B^\top PB + R \right)^{-1} B^\top PA. \tag{3.34}$$

Finally, the control input is applied as:

$$\mathbf{u}_k = \mathbf{u}_{\text{ref}} + \delta\mathbf{u}_k = \mathbf{u}_{\text{ref}} - K\,\delta\mathbf{x}_k. \tag{3.35}$$

3.3.3 Hybrid LQR Control with Pure Pursuit Inspired Look-Ahead Strategy

Based on the above LQR formulation, the look-ahead strategy selects a reference point $\mathbf{x}_{ref}$ along the trajectory at a predefined distance $d_{lookahead}$ from the current position:

$$|\mathbf{x}_{ref} - \mathbf{x}| \geq d_{lookahead}. \tag{3.36}$$

This reference state is then used to compute the control input via the LQR feedback law, ensuring that the vehicle follows the trajectory smoothly.

By integrating the Pure Pursuit look-ahead strategy with LQR-based feedback control, we enhance the system's ability to track a given trajectory efficiently. This approach ensures smooth control actions and better anticipation of future trajectory points, making it suitable for applications in autonomous navigation and mobile robotics.

3.3.4 Tracking Method Comparison

To achieve higher performance of different trajectory tracking strategies for AMRs, we compare Pure Pursuit and a hybrid LQR+Pure Pursuit controller. Each method offers distinct advantages depending on the system's dynamics, path complexity, and operating conditions. The comparison below highlights their differences in terms of control strategy, model dependency, tuning complexity, robustness, and practical performance.

Table 3.2: Comparison of Hybrid LQR and Pure Pursuit for Trajectory Tracking

Aspect	**Hybrid LQR**	**Pure Pursuit**
Control Strategy	Optimal control minimizing a quadratic cost over state and input deviations.	Geometric control using a lookahead point to compute curvature and steering.
Model Requirement	Requires a linear or linearized dynamic model.	No model required; purely kinematic.
Design Parameters	Weighting matrices Q and R.	Lookahead distance (fixed or adaptive).
Robustness	Sensitive to modeling errors and external disturbances.	Robust to model inaccuracies but may deviate on sharp turns or at high speeds.
Pros	Smooth and accurate tracking, stable even at higher speeds.	Simple to implement; effective at low speeds and for mild curvature.
Cons	May not converge under poor linearization or sharp maneuvers.	Struggles with backward or complex trajectories.

Both the hybrid LQR + Pure Pursuit controller and the standalone Pure Pursuit method are implemented in a switchable framework, allowing flexible deployment depending on the application scenario. While the hybrid approach provides improved stability and smoother tracking under dynamic conditions, the Pure Pursuit controller remains a lightweight and robust alternative for simpler or low-speed operations. This modular design ensures adaptability across varying operational requirements and computational constraints.

4

Evaluation and Results

This chapter presents a comprehensive evaluation of the multi-robot communication architecture using an automated testing framework, AutoTest, developed as a ROS2 module. AutoTest was designed to systematically assess system performance under varying network latency conditions by orchestrating simulation environments, managing process lifecycles, and collecting detailed metrics across numerous test iterations and latency levels. The evaluation focused on key performance indicators such as path deviation, execution efficiency, motion smoothness, MPC solver convergence, and system-level outcomes including success, collision, and timeout rates.

Five representative scenarios were examined, ranging from simple static obstacle navigation to complex multi-robot interactions in constrained environments. The experimental results highlight the system's robustness, demonstrating that it consistently maintained high success rates and stable control performance—even under severe communication latency. Notably, the impact of latency on system behavior varied depending on scenario complexity and the degree of resource contention, offering valuable insights into the resilience and scalability of the proposed architecture.

4.1 Evaluation Framework

4.1.1 AutoTest Module Architecture

To systematically evaluate the performance of our multi-robot communication architecture under varying network conditions, we developed an automated testing framework called AutoTest. This framework enables reproducible quantitative assessment of system behavior across different latency scenarios, providing empirical validation of our design's latency tolerance capabilities.
The AutoTest module is implemented as a ROS2 node that orchestrates the entire testing process, from environment initialization to performance data collection and analysis. Figure 4.1 illustrates the architectural components of the AutoTest framework.
The framework follows a modular design with several key components:

1. **Configuration Management**: The system loads test scenarios and parameters from YAML configuration files, allowing flexible definition of multiple testing scenarios with specific environmental settings, robot configurations,

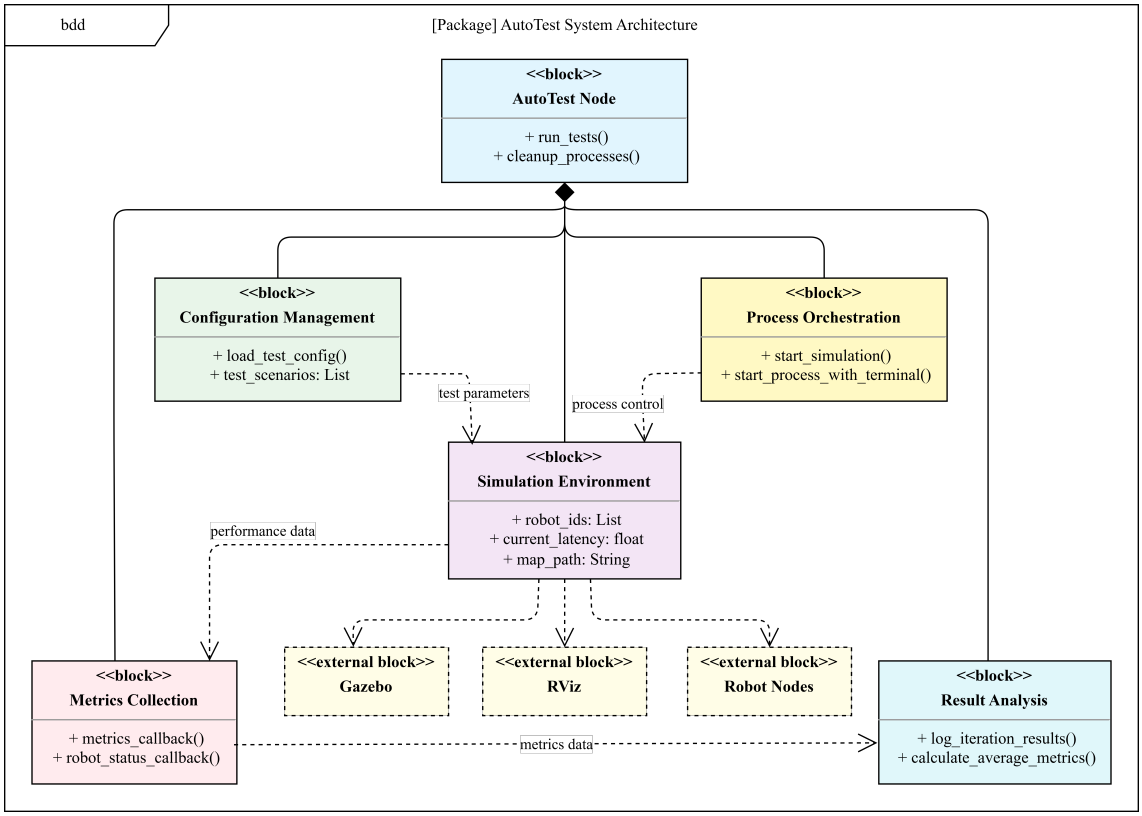


Figure 4.1: AutoTest architecture showing main components and their relationships

and test parameters.

2. **Simulation Environment**: The framework integrates multiple simulation tools, creating a comprehensive testing environment:
   - Gazebo physics simulation for realistic robot dynamics
   - RViz visualization for visual monitoring and verification
   - ROS2-based message buffer for precise network latency emulation
3. **Process Orchestration**: The framework automatically manages the lifecycle of all required processes, including:
   - Robot manager for high-level coordination
   - Individual robot nodes representing each agent in the system
   - Visualization and simulation processes
4. **Metrics Collection**: The system subscribes to performance metrics published by the robot system, collecting comprehensive data on:
   - Path execution accuracy (deviation area, normalized deviation)
   - Efficiency metrics (execution time, path length)
   - Motion quality metrics (linear and angular smoothness)
   - Control stability (convergence success rate)
5. **Result Analysis**: The framework includes automatic calculation of performance statistics, generating both detailed per-iteration logs and summary results for each latency condition.

4.1.2 Testing Methodology

Our testing approach systematically evaluates the system's performance across a spectrum of network latency conditions. The methodology implements a multi-layered structure:

1. **Scenario Selection**: Each test scenario defines a specific environment configuration, including map layout, robot placements, and mission parameters.

2. **Latency Variation**: For each scenario, the framework tests multiple latency values (typically ranging from 0ms to 1000ms), simulating different network conditions from ideal to severely degraded.

3. **Statistical Significance**: Each latency value is tested through multiple iterations (typically ranging from 20 to 50 per latency value) to ensure statistical validity of the results.

4. **Success Criteria**: The framework automatically detects and records:
   - Successful mission completion (all robots reaching targets)
   - Failure modes (collisions, timeouts)
   - Performance metrics for successful iterations

This methodical approach enables us to quantify the relationship between network latency and system performance, providing empirical evidence of our architecture's latency tolerance characteristics.

4.1.3 Data Collection and Analysis

The AutoTest framework implements comprehensive data collection and analysis capabilities:

1. **Per-Iteration Logging**: For each test iteration, the system logs:
   - Detailed performance metrics for each robot
   - Success/failure status and failure types
   - Visual screenshots for qualitative assessment
2. **Statistical Analysis**: The framework automatically computes:
   - Success rates under each latency condition
   - Average performance metrics across successful iterations
   - Failure distribution by type (timeouts vs. collisions)
3. **Result Visualization**: The collected data enables generation of performance curves showing the relationship between latency and various performance metrics.

4.2 Performance Metrics and Evaluation Criteria

To quantitatively assess the performance of our multi-robot system under varying latency conditions, we employ a comprehensive set of metrics designed to capture different aspects of system behavior. Each metric provides specific insights into how network latency affects the robots' ability to execute their tasks efficiently and accurately. Table 4.1 provides a summary of the metrics used in our evaluation framework.

4.2.1 Path Execution Accuracy

1. **Deviation Area:**
   The deviation area metric quantifies the cumulative spatial error between the planned path and the actual path taken by the robot, providing a measure of path following accuracy.

   For a robot following an actual path $P_a = \{(x_i, y_i)\}_{i=1}^{n}$ when given a planned path $P_p = \{(x_j, y_j)\}_{j=1}^{m}$, the deviation area is defined as:

   $$A_{dev} = \sum_{i=1}^{n-1} \frac{1}{2}(d_i + d_{i+1}) \cdot l_i \tag{4.1}$$

   where d_i is the minimum distance from point (x_i, y_i) to any line segment in the planned path, and l_i is the length of the segment between consecutive points in the actual path.

   This metric calculates the approximate area between the actual and planned paths, with larger values indicating greater deviation from the planned trajectory.
2. **Normalized Deviation:**

Table 4.1: Summary of Evaluation Metrics

Category	**Metric**	**Description**
Path Execution Accuracy	Deviation Area (A_{dev})	Area between planned and actual paths
	Normalized Deviation (D_{norm})	Deviation area normalized by path length
Efficiency	Execution Time (T_{exec})	Total time to complete the path
	Path Length (L_p)	Total length of the planned path
Motion Quality	Linear Smoothness (S_l)	Median absolute linear acceleration
	Angular Smoothness (S_a)	Median absolute angular acceleration
Control Stability	Convergence Success Rate (R_{conv})	Ratio of successful MPC solver convergences
System-Level Performance	Success Rate ($R_{success}$)	Ratio of successful test iterations
	Collision Rate ($R_{collision}$)	Ratio of iterations with collisions
	Timeout Rate ($R_{timeout}$)	Ratio of iterations exceeding time limit

To facilitate comparison across different path lengths, we normalize the deviation area by the planned path length:

$$D_{norm} = \frac{A_{dev}}{L_p} \tag{4.2}$$

where L_p is the total length of the planned path. This normalization allows for fair comparison between robots traveling different distances or across different scenarios.

4.2.2 Efficiency Metrics

1. **Execution Time:** The execution time measures the total duration required for a robot to travel from its starting position to its destination:

$$T_{exec} = t_{end} - t_{start} \tag{4.3}$$

where t_{start} is the timestamp when the robot begins its journey and t_{end} is when it reaches its destination. Longer execution times under increased latency conditions indicate degraded system efficiency.

2. **Path Length:** While the planned path length is predetermined, we track this metric to provide context for other measurements and to verify that robots

are attempting to follow optimal trajectories:

$$L_p = \sum_{j=1}^{m-1} \sqrt{(x_{j+1} - x_j)^2 + (y_{j+1} - y_j)^2} \tag{4.4}$$

4.2.3 Motion Quality Metrics

1. **Linear Smoothness:** Linear smoothness quantifies the stability of a robot's linear motion by measuring the median absolute linear acceleration:

$$S_l = \text{median}\{|a_i|\}_{i=1}^{k} \tag{4.5}$$

where a_i represents the absolute linear acceleration calculated between consecutive velocity measurements. Lower values indicate smoother motion with fewer abrupt changes in velocity, which is generally desirable for both energy efficiency and passenger comfort in transportation applications.

2. **Angular Smoothness:** Similarly, angular smoothness evaluates the stability of a robot's rotational motion, measured as the median absolute angular acceleration:

$$S_a = \text{median}\{|\alpha_i|\}_{i=1}^{k} \tag{4.6}$$

where α_i represents the absolute angular acceleration. Lower values indicate smoother turning behavior with less jerky rotations, which contributes to overall system stability and reduces mechanical stress on the robots.

4.2.4 Control Stability Metrics

1. **Convergence Success Rate:** To evaluate the robustness of our control algorithm, we track the convergence success rate of the MPC solver:

$$R_{conv} = \frac{N_{success}}{N_{success} + N_{failure}} \tag{4.7}$$

where $N_{success}$ is the number of successful MPC solver convergences and $N_{failure}$ is the number of failed convergences. Higher rates indicate more reliable control performance under the given conditions.

4.2.5 System-Level Performance Indicators

In addition to the robot-specific metrics described above, we evaluate overall system performance using the following indicators:

1. **Success Rate:** The ratio of test iterations where all robots successfully reach their destinations without collisions or timeouts:

$$R_{success} = \frac{N_{successful\_iterations}}{N_{total\_iterations}} \tag{4.8}$$

2. **Collision Rate:** The fraction of test iterations where at least one robot collision occurs:
$$R_{collision} = \frac{N_{collision\_iterations}}{N_{total\_iterations}} \tag{4.9}$$

3. **Timeout Rate:** The proportion of test iterations that exceed the maximum allowed execution time:
$$R_{timeout} = \frac{N_{timeout\_iterations}}{N_{total\_iterations}} \tag{4.10}$$

These system-level indicators provide a comprehensive view of how increasing network latency affects the overall reliability and safety of the multi-robot system, complementing the detailed performance metrics collected for individual robots.

Through this systematic evaluation framework, we can objectively assess how our multi-robot communication architecture performs under different network conditions, providing quantitative evidence for the effectiveness of our latency-tolerant design approaches.

4.3 Evaluation Scenarios

To systematically assess the performance and robustness of the proposed control architecture, we designed five distinct test scenarios, each targeting specific aspects of multi-robot coordination and real-world applicability.

Scene 1 (Figure4.2) is designed to evaluate a robot's ability to navigate around static obstacles. This basic setup verifies both the core navigation functionality and the correct integration of the control architecture within a static environment.

Scene 2 (Figure4.3) introduces a typical interaction scenario where two robots encounter each other in a narrow corridor. This setup tests the system's ability to resolve conflicts and coordinate motion in constrained spaces, a common challenge in real-world deployments.

Scene 3 (Figure4.4) evaluates multi-robot interaction at a crossing point. We consider two variations: a balanced crossing (Scene 3A), where both robots approach the intersection from equal distances, and an unbalanced crossing (Scene 3B), where one robot is closer than the other. These variations test the coordination mechanism under shared-resource constraints, a situation commonly encountered in indoor logistics.

Scene 4 (Figure4.5) replicates a real-world indoor logistics environment. It includes both encounter and crossing conditions and is used to assess the practical applicability of the system under near-deployment configurations.

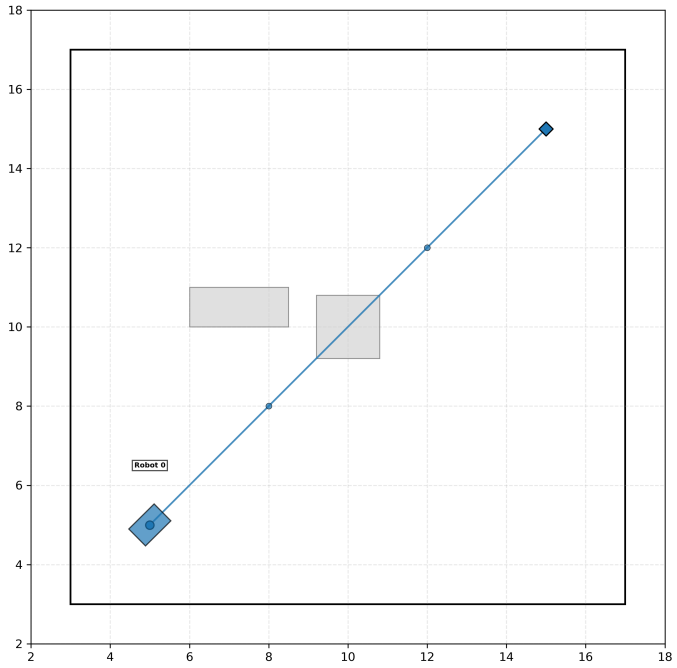


Figure 4.2: Scene 1. This scene evaluates the robot's ability to navigate static obstacles and serves as a test for both the environment and system performance.

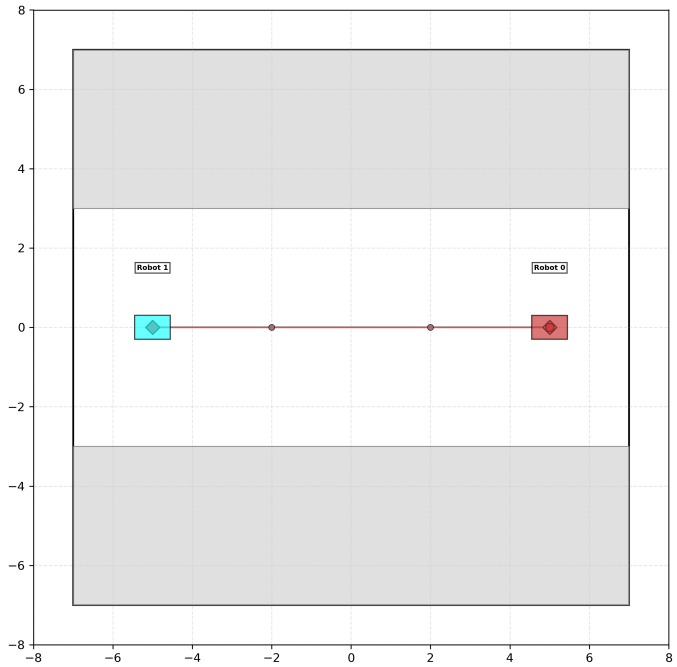
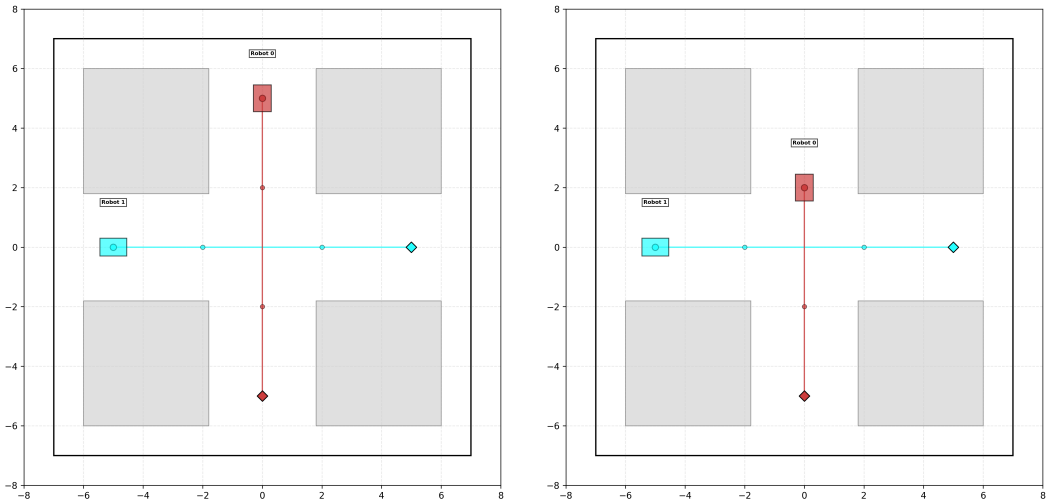


Figure 4.3: Scene 2. In this scene, two robots encounter each other in a narrow corridor and attempt mutual avoidance. This scenario evaluates the coordination strategies in constrained environments.



A) Balanced Crossing B) Unbalanced Crossing

Figure 4.4: Scene 3. These scenes evaluate the robots' ability to avoid each other during a crossing. The left (Scene 3A) depicts a balanced crossing scenario, while the right (Scene 3B) shows an unbalanced crossing scenario.

Scene 5 (Figure 4.6) tests the system's scalability and performance under a high-load condition. This scenario involves ten robots, multiple crossing events, and static obstacles. It serves to verify the system's ability to coordinate a large fleet of AMRs in complex environments.

4.4 Results

In this evaluation, all test cases are conducted using consistent MPC parameters and identical local control strategies to ensure a fair comparison. The latency value is selected based on cross-device testing, which showed that within the same local area network (LAN), the natural communication delay remains within 200ms. Therefore, setting the latency to 1000ms in our experiments represents a deliberately extended delay and serves as a reasonable approximation of a long-latency scenario.

The primary MPC solver parameters are defined as:

$$T_s = 0.2, \quad N = 30, \quad \alpha = 0.5, \quad Q_0 = 1, \quad R = 0, \quad S = \begin{bmatrix} 5 & 0 \\ 0 & 0.5 \end{bmatrix}, \tag{4.11}$$

$$Q_{\mathcal{F}} = 100, \quad Q_{\mathcal{O}} = 10, \quad Q_{\mathcal{D}} = \begin{bmatrix} 100 & 100 & 100 & 100 & 100 & 80 & 80 & 80 & 80 & 80 \\ 50 & 50 & 50 & 50 & 20 & 20 & 20 & 20 & 10 & 10 \\ 10 & 10 & 5 & 5 & 5 & 2 & 2 & 2 & 1 & 1 \end{bmatrix} \tag{4.12}$$

(4.13)

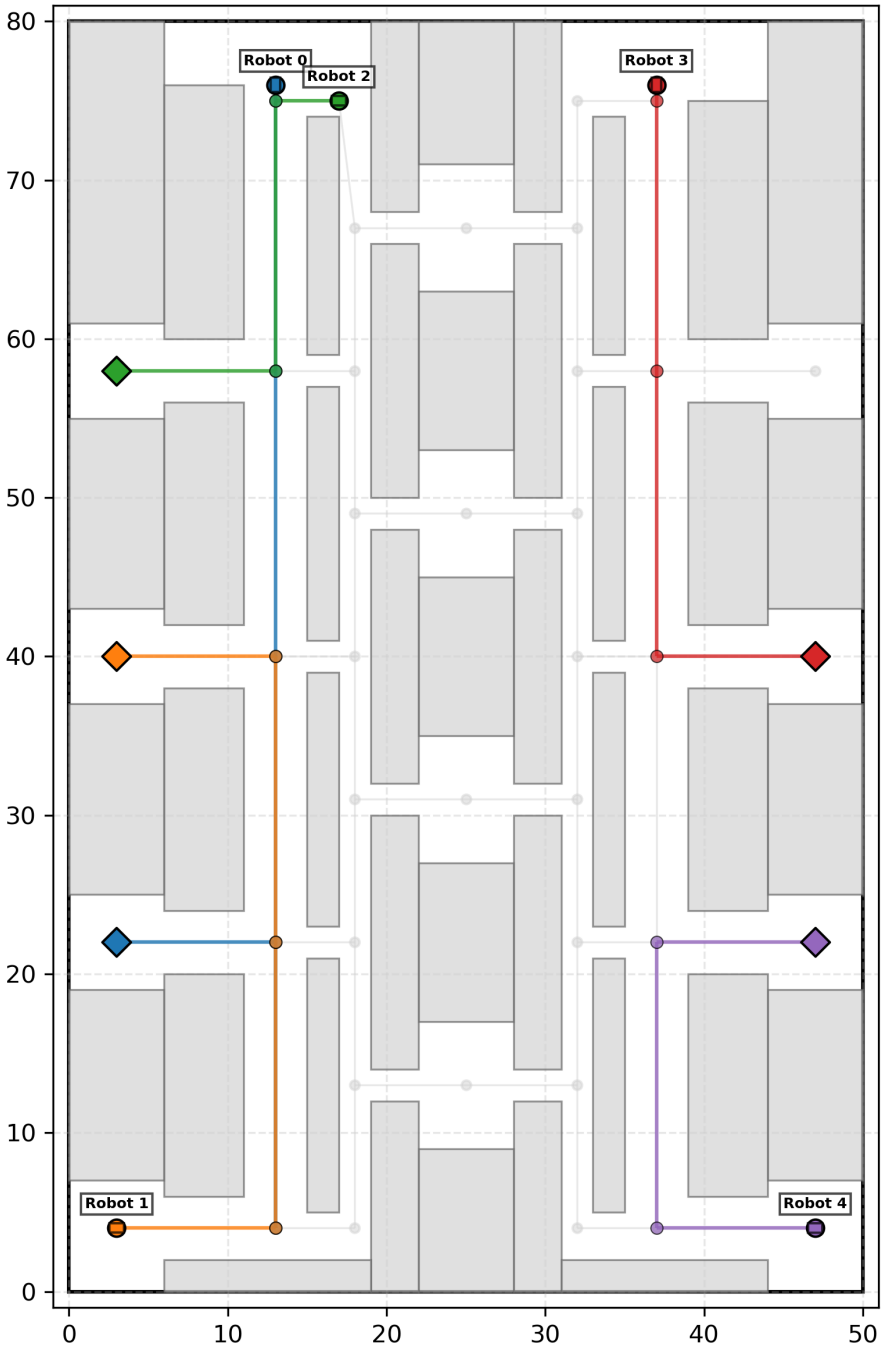


Figure 4.5: Scene 4. This scene represents a typical production line scenario, including an encounter and crossing condition, and evaluates the system's suitability for real-world deployment.

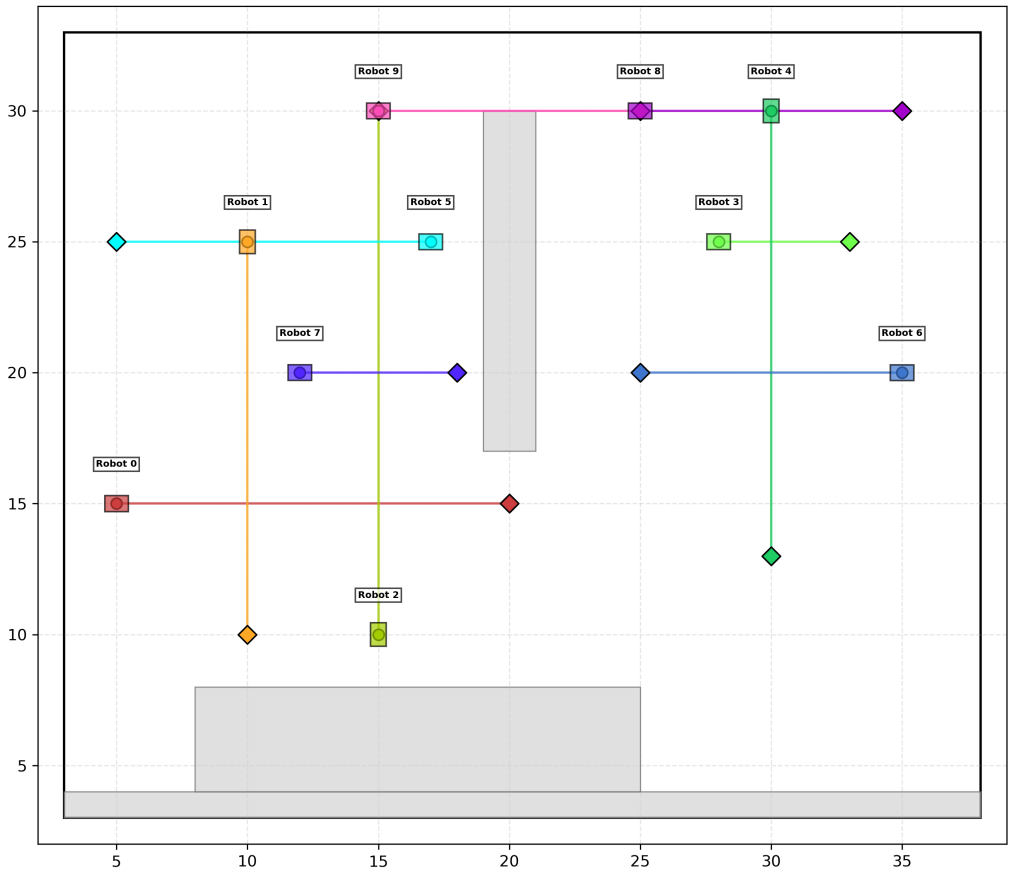


Figure 4.6: Scene 5. This scene involves 10 robots and includes multiple crossing events and static obstacle navigation. It demonstrates the system's scalability and ability to handle complex conditions.

The Hybrid LQR controller uses the following weight matrices and lookahead distance:

$$Q = \begin{bmatrix} 1000 & 0 & 0 \\ 0 & 1000 & 0 \\ 0 & 0 & 10 \end{bmatrix}, \quad R = \begin{bmatrix} 0.001 & 0 \\ 0 & 0.1 \end{bmatrix}, \quad d_{\text{lookahead}} = 0.5 \tag{4.14}$$

The Pure Pursuit controller is configured with the following parameters:

$$L = 0.5, \quad \alpha = 0.1 \tag{4.15}$$

To ensure statistical significance, each test case is executed 20 times on identical hardware. The performance metrics for each scenario are summarized in Table 4.2. As shown in Table 4.2, the success rates across all scenarios remain consistently

Table 4.2: Performance metrics across scenarios with and without latency

Scenario	Latency (ms)	Success Rate	Failure		Avg Execution Time(s)	Deviation (m)	Smoothness		Solver	
			Timeout Rate	Collision Rate			Linear Acc(m/s²)	Angular Acc(rad/s²)	Solver Call Count	Convergence Rate
Scene 1	0	100%	0%	0%	33.38	1.19	0.0939	0.0310	19	98%
	1000	100%	0%	0%	35.61	1.47	0.0787	0.0362	22	99%
Scene 2	0	100%	0%	0%	35.39	0.43	0.0006	0.0022	27	98%
	1000	85%	15%	0%	38.73	0.55	0.0005	0.0017	29	97%
Scene 3A	0	95%	0%	5%	30.16	0.39	0.0007	0.0030	23	93%
	1000	95%	5%	0%	37.59	0.36	0.0007	0.0023	30	84%
Scene 3B	0	95%	5%	0%	26.41	0.18	0.0130	0.0170	15	86%
	1000	100%	0%	0%	21.17	0.09	0.0138	0.0283	8	91%
Scene 4	0	95%	5%	0%	116.14	5.26	0.0467	0.0001	75	98%
	1000	100%	0%	0%	121.15	7.71	0.0434	0.0000	90	98%
Scene 5	0	95%	0%	5%	119.49	47.10	0.0293	0.0007	575	100%
	1000	95%	5%	0%	121.23	48.59	0.0328	0.0012	563	99%

high, indicating the robustness of the control strategies under both normal and high-latency conditions. For a clearer comparison of success rates across scenarios, refer to Figure 4.7. In the simplest scenario, Scene 1, the system achieves a 100%

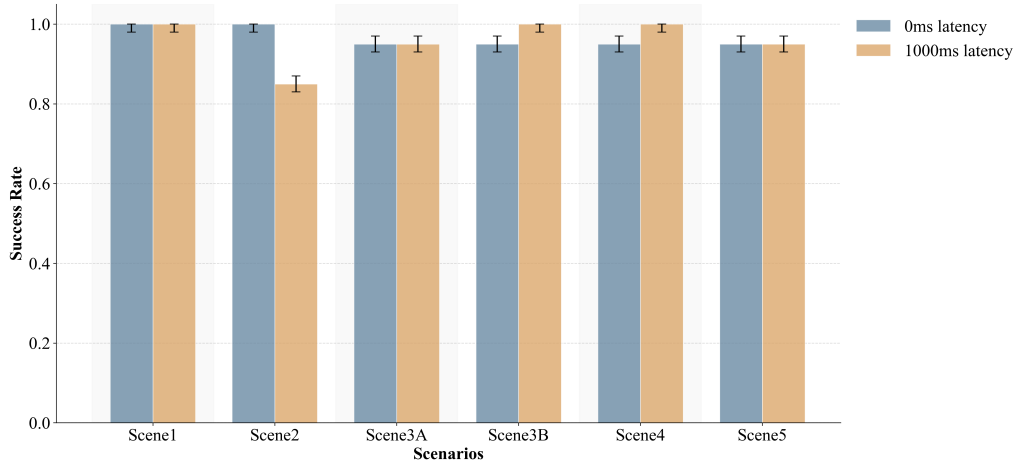


Figure 4.7: The success rate comparison

success rate under both no-latency and high-latency conditions, demonstrating its reliability in straightforward environments.

Comparing Scene 2, Scene 3A, and Scene 3B, we observe different patterns of resource competition. Scene 2 and Scene 3A represent balanced scenarios, where the competing agents have symmetric access to resources. In contrast, Scene 3B is characterized by unbalanced competition, where one agent has a positional or temporal advantage over the other.

From the success rates, we observe that high latency tends to negatively impact the balanced scenarios (Scene 2 and Scene 3A) more than the unbalanced one (Scene 3B). Interestingly, Scene 3B shows a slight improvement under high latency, which may be attributed to one robot seizing the shared resource earlier, while the other defers its action due to delayed communication—effectively reducing direct conflict.

In the more complex environments—Scene 4 (balanced competition) and Scene 5 (unbalanced competition)—the success rates appear largely unaffected by the introduction of latency. This can be explained by the increased spatial flexibility in these scenarios, allowing the system more opportunities to invoke the solver and adjust robot trajectories to avoid conflicts.

Further insights into the difficulty of robot coordination under different conditions

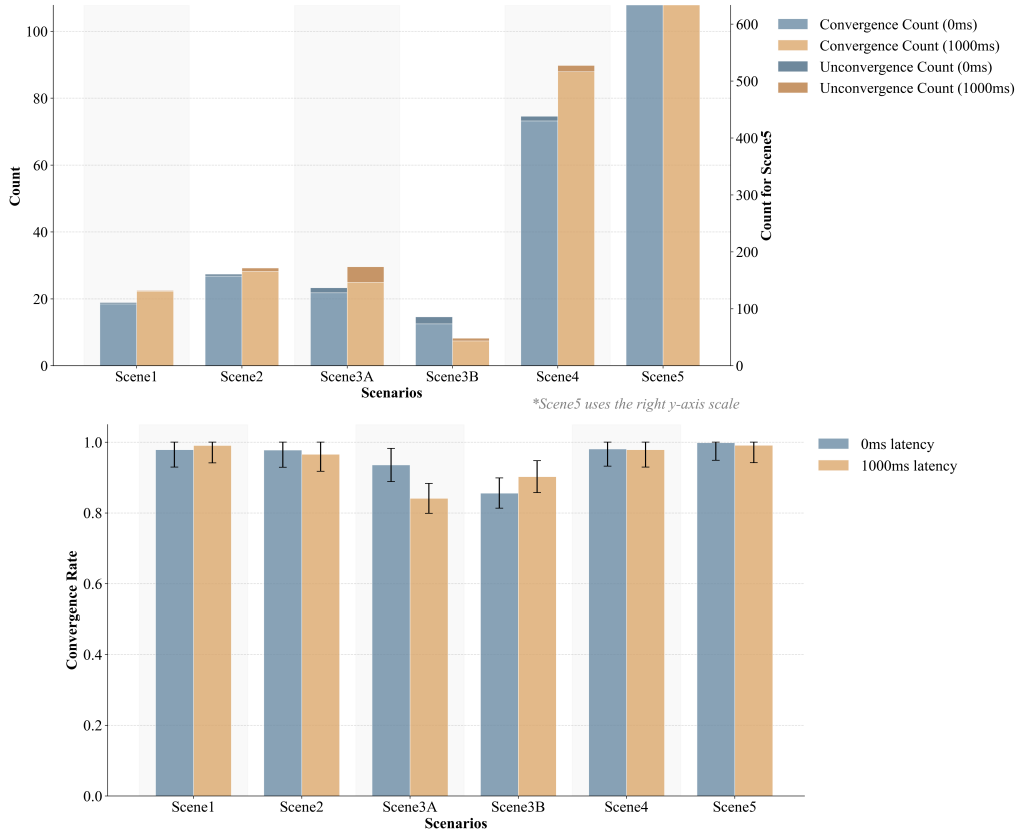


Figure 4.8: MPC solver's convergence condition

can be obtained by analyzing the convergence behavior of the MPC solver shown in Figure 4.8. In Scene 1, a relatively simple scenario, the solver exhibits a high and consistent convergence rate regardless of the presence of latency. However, under high-latency conditions, the solver is invoked more frequently, indicating a longer overall execution time. Despite this, latency does not appear to impose additional difficulty on the solver's ability to find a solution.

In the competitive scenarios—Scene 2, Scene 3A, and Scene 3B—we observe differing impacts based on the nature of resource competition. Under balanced competition (Scenes 2 and 3A), latency leads to more frequent solver invocations and a reduced convergence rate, suggesting increased difficulty in coordination. In contrast, the unbalanced competition in Scene 3B shows fewer solver calls and a higher convergence rate under latency. This implies that latency can actually simplify coordination in unbalanced situations, as one robot may dominate resource access while the other adapts accordingly. This observation is consistent with our earlier findings on success rates.

For the more complex scenarios—Scene 4 (balanced competition) and Scene 5 (unbalanced competition)—the effects of latency are mixed. In Scene 4, latency results in more solver calls, but the convergence rate remains nearly unchanged due to the inherently high number of solver invocations in this scenario, which makes differences less pronounced. In Scene 5, however, we observe a significantly reduced number of solver calls under the latency setting, further supporting the notion that latency can be advantageous in certain unbalanced competitive situations.

In summary, the experimental results demonstrate that the proposed coordination framework is robust across a variety of scenarios, including those with significant communication latency. While latency increases solver invocation frequency and slightly degrades performance in balanced competition scenarios, it has limited impact—or even slight benefits—in unbalanced or more complex environments. These findings suggest that the system can adapt effectively to real-world network conditions and dynamic interactions, maintaining high success rates and reasonable computational demands. This robustness under varying levels of difficulty and latency confirms the practicality and scalability of the proposed approach for multi-robot coordination.

5
Conclusion

This thesis investigates the integration of cloud robotics into AMR coordination, with a primary focus on managing network latency. We have developed a complete system from the ground up, addressing architecture, inter-robot communication, coordination control, message sharing, local decision-making, and trajectory tracking. The system is validated in a physics-based simulation environment under various latency conditions.

5.1 Contributions

Multilayer Control Architecture: A hierarchical control structure is designed and implemented, distributing responsibilities between local robot controllers and cloud-based components. This architecture enables trajectory generation in the cloud while keeping only essential tasks onboard, enhancing system robustness in the presence of latency.

Switchable Trajectory Generation: We propose an approach to trajectory planning by combining traditional MPC with adaptive local adjustments to cloud-optimized trajectories. Comparative evaluations show that our hybrid approach achieves smoother execution and higher computational efficiency.

Flexible Trajectory Tracking Strategies: The local robot control design emphasizes a clear and modular onboard architecture. A key feature of this design is the integration of a switchable controller framework, enabling dynamic selection between different tracking strategies—such as pure pursuit and hybrid LQR—depending on the operational context. This flexibility allows the system to adapt effectively to varying task demands and environmental conditions, ensuring reliable and responsive performance in diverse scenarios.

Well-designed Local Decision-making: To ensure operational safety during communication delays or interruptions, an FSM governs well-defined state transitions, while the control logic specifies the real-time behavior executed within each state. This local fallback strategy allows robots to maintain safe operation independently of the cloud, enhancing system reliability in real-world deployments.

Comprehensive Simulation Platform: We developed an integrated simulation

environment combining RViz for visualization, Gazebo for physics-based simulation, and an automated testing framework. This platform enables systematic testing and evaluation of multi-robot coordination strategies under varying levels of network latency, allowing for consistent performance benchmarking.

To sum up, this thesis contributes a robust, latency-tolerant multi-robot control architecture, validated through a custom-developed AutoTest evaluation framework. By simulating a range of real-world scenarios under controlled latency conditions, we demonstrated that the system sustains high success and convergence rates while maintaining efficiency and motion quality. The empirical results confirm the architecture's resilience and adaptability in the face of degraded communication conditions, reinforcing its applicability to real-world multi-agent robotic deployments where network quality may fluctuate.

5.2 Research Outcomes

To provide a structured and comprehensive summary of our findings, we revisit and address the research questions posed in the introduction. By systematically analyzing each question in light of the experimental results and system design choices, we aim to clarify the contributions of our work and highlight the effectiveness of the proposed architecture and strategies in managing communication latency and coordination challenges in multi-robot systems.

RQ1: How to design cloud-based architecture for efficient AMR fleet coordination?

Our research addresses this question through a hierarchical cloud architecture that distributes responsibilities between centralized coordination and local control. The cloud component consists of a central Manager node responsible for global state management and multiple Cluster nodes that handle individual robot planning. This design effectively balances the benefits of centralized information sharing with the scalability of distributed computation.

The Manager node serves as a lightweight coordination hub, maintaining global state consistency and handling robot registration while delegating the computationally intensive trajectory planning to dedicated Cluster nodes. Each Cluster node is paired with a specific robot, creating a one-to-one mapping that enables parallel processing and reduces system latency. This approach prevents the bottlenecks associated with sequential processing while maintaining a coherent system view.

Communication between cloud components is designed through a selective information-sharing strategy, where only essential coordination data traverses the network, minimizing bandwidth requirements. The architecture implements robust state synchronization mechanisms and heartbeat protocols to maintain system integrity despite network fluctuations. By structuring cloud resources in this hierarchical manner, our design achieves efficient fleet coordination while accommodating the inherent

challenges of cloud-based robotics, including variable network conditions and computation load balancing.

RQ2: How can DMPC be applied to coordinate robot fleets effectively under high-latency conditions?

Effective coordination in latency-prone environments is achieved through a combination of distributed information sharing, modular trajectory planning, and robust state synchronization. In our system, critical information—such as robot identities and their planned trajectories—is shared with a centralized Robot Manager, which functions as both a fleet-level information distributor and collector. Concurrently, the vision system continuously updates each robot's current state within the Robot Manager, ensuring situational awareness across the fleet.

Thanks to ROS2's support for precise time stamping, information updates are synchronized and can be reliably interpreted despite communication delays. The use of DMPC enables each robot to independently compute its own trajectory using only locally available or shared information, rather than requiring full system-wide knowledge. This decentralized approach significantly reduces communication dependencies and improves system resilience under latency, while still ensuring coordinated behavior across the fleet.

RQ3: What strategies can be employed on individual AMRs to manage outdated trajectory information when latency and computation bottlenecks exist?

To handle outdated trajectory information under latency and computational constraints, we employ distance-based lookahead strategies for both hybrid LQR and pure pursuit trajectory tracking. These methods allow the AMR to continue following the last known trajectory until its endpoint, effectively buying time for the system to recompute and update trajectories based on the latest state estimates. This buffering mechanism gives the centralized cluster additional opportunities to refine trajectory plans under fluctuating conditions.

Moreover, each AMR is equipped with a local safety module that uses onboard laser scan data to perform reactive collision avoidance. This module ensures safety even when global trajectory updates are delayed. At the same time, emergency behaviors are implemented to handle abnormal cases—such as executing a stop when receiving a trajectory with an excessively distant start point, or halting operation when communication errors are detected (e.g., missing updates or dropped connections).

RQ4: What is the quantitative impact of varying communication latency conditions on multi-robot system performance?

Through various architectural and algorithmic efforts, we have significantly mitigated or bypassed the impact of latency. Nonetheless, its influence on multi-

robot system performance remains nuanced and scenario-dependent. In simple, low-conflict environments (e.g., Scene 1), latency has minimal impact, with the system maintaining a 100% success rate and stable solver convergence. In contrast, balanced competition scenarios (e.g., Scene 2 and Scene 3A) experience increased solver invocations and reduced convergence rates under latency, reflecting greater coordination difficulty. Interestingly, in unbalanced competition settings (e.g., Scene 3B and Scene 5), latency can actually ease coordination by staggering access to shared resources, resulting in fewer solver calls and, in some cases, improved success rates. In more complex environments with greater spatial flexibility (e.g., Scene 4 and Scene 5), the system's adaptive trajectory generation further mitigates latency effects.

However, latency can occasionally cause robots to enter non-convex regions of the solution space, especially when error-handling mechanisms are triggered during delayed communication. This can prevent the MPC solver from converging in specific instances. Despite these challenges, the system consistently demonstrates robustness and adaptability, maintaining high success rates and reliable performance across a wide range of conditions.

5.3 Current Limitations and Future Works

Although our system and control strategies have demonstrated strong robustness and reliability across a range of scenarios, several limitations were identified during development that warrant further investigation in future work:

- **Lack of real-world validation**: While we utilized the Gazebo simulation environment to approximate real-world conditions through its physics engine, the system has not yet been tested on physical robots. Real-world deployment is necessary to fully assess system performance under unpredictable environmental factors and hardware constraints.

- **Limited MPC convergence under specific conditions**: As discussed in RQ4, the MPC solver may struggle to converge in certain local situations—particularly under latency-induced disturbances or non-convex conditions. This limitation stems not only from solver capabilities but also from the formulation itself, which currently does not account for feedback from local control actions in the optimization loop.

- **Suboptimal trajectory quality and coordination in competitive scenarios**: In resource-constrained or competitive settings, the MPC may generate inefficient or non-intuitive trajectories—such as unnecessary forward–backward movements or suboptimal yielding behaviors (e.g., passing in front rather than behind another AMR). These behaviors highlight the lack of higher-level fleet coordination. Incorporating traffic management strategies, such as heatmap-based conflict avoidance, could help mitigate these issues.

- **Insufficient evaluation hardware**: The current cloud-based cluster is limited in computational capacity. In practice, such modules should ideally run on high-performance computing clusters equipped with intelligent load balancing to better reflect the intended deployment conditions.

Therefore, several promising directions remain for future work:

- **Game-theoretic traffic coordination**: Incorporate game-theoretic approaches at the cluster level to enable strategic fleet-level traffic balancing. This can proactively mitigate or avoid competitive scenarios by optimizing the behavior of individual AMRs within a shared environment.

- **Reinforcement learning for trajectory optimization**: Explore the use of reinforcement learning to improve the quality of generated trajectories. By learning from interactions with the environment, AMRs can discover more efficient and adaptive motion strategies that outperform rule-based or purely optimization-driven methods in complex or dynamic settings.

- **Comprehensive real-world validation**: Extend the current system to real-world deployments for thorough testing under physical constraints and uncertainties. This step is critical to validate the scalability, robustness, and safety of the proposed architecture in practical applications.

This thesis contributes a robust cloud-based control architecture for multi-robot coordination that withstands communication latency. Through trajectory planning, modular control, and a strong simulation evaluation framework, we demonstrate high adaptability and reliability under diverse and challenging conditions. These findings pave the way for deploying latency-tolerant multi-robot systems in dynamic, real-world environments such as logistics, manufacturing, and beyond.

Bibliography

[1] J. Al-Jaroodi. (2021) Multi-robot systems. Encyclopedia. [Online]. Available: https://encyclopedia.pub/entry/17215

[2] Technavio, "Robotics market analysis apac, europe, north america, middle east and africa, south america - us, china, japan, india, canada, south korea, germany, uk, italy, france - size and forecast 2025-2029," Technavio, Tech. Rep., 2 2025, report code: IRTNTR44162. [Online]. Available: https://www.technavio.com/report/robotics-market-industry-analysis

[3] S.-J. Chung, A. A. Paranjape, P. Dames, S. Shen, and V. Kumar, "A survey on aerial swarm robotics," *IEEE Transactions on Robotics*, vol. 34, no. 4, pp. 837–855, 2018.

[4] J. Wang, Y. Lin, R. Liu, and J. Fu, "Odor source localization of multi-robots with swarm intelligence algorithms: A review," *Frontiers in Neurorobotics*, vol. 16, p. 949888, 2022.

[5] K. Kamei, S. Nishio, N. Hagita, and M. Sato, "Cloud networked robotics," *IEEE Network*, vol. 26, no. 3, pp. 28–34, 2012.

[6] L. Wang, M. Liu, and M. Q.-H. Meng, "Towards cloud robotic system: A case study of online co-localization for fair resource competence," in *2012 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, 2012, pp. 2132–2137.

[7] V. K. Sarker, J. P. Queralta, T. N. Gia, H. Tenhunen, and T. Westerlund, "Offloading slam for indoor mobile robots with edge-fog-cloud computing," in *2019 1st international conference on advances in science, engineering and robotics technology (ICASERT)*. IEEE, 2019, pp. 1–6.

[8] A. Rahman, J. Jin, A. Rahman, A. Cricenti, M. Afrin, and Y.-n. Dong, "Energy-efficient optimal task offloading in cloud networked multi-robot systems," *Computer Networks*, vol. 160, pp. 11–32, 2019.

[9] B. Kehoe, S. Patil, P. Abbeel, and K. Goldberg, "A survey of research on cloud robotics and automation," *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 2, pp. 398–409, 2015.

[10] G. Hu, W. P. Tay, and Y. Wen, "Cloud robotics: architecture, challenges and applications," *IEEE network*, vol. 26, no. 3, pp. 21–28, 2012.

[11] M. Waibel, M. Beetz, J. Civera, R. d'Andrea, J. Elfring, D. Galvez-Lopez, K. Häussermann, R. Janssen, J. Montiel, A. Perzylo *et al.*, "Roboearth," *IEEE Robotics & Automation Magazine*, vol. 18, no. 2, pp. 69–82, 2011.

[12] S. Shukla, M. F. Hassan, M. K. Khan, L. T. Jung, and A. Awang, "An analytical model to minimize the latency in healthcare internet-of-things in fog computing environment," *PloS one*, vol. 14, no. 11, p. e0224934, 2019.

[13] A. Brogi and S. Forti, "Qos-aware deployment of iot applications through the fog," *IEEE internet of Things Journal*, vol. 4, no. 5, pp. 1185–1192, 2017.

[14] K. Gillani, J. Martín-Pérez, M. Groshev, A. De La Oliva, and R. Gazda, "Don't let me down! offloading robot vfs up to the cloud," in *2023 IEEE 9th International Conference on Network Softwarization (NetSoft)*. IEEE, 2023, pp. 232–236.

[15] J. Gao and D. Wang, "Robot cloud computing and al services-state-of-the-art solutions, challenges, and needs," in *2022 IEEE International Conference on Service-Oriented System Engineering (SOSE)*. IEEE, 2022, pp. 105–116.

[16] W. Chen, W. Chi, S. Ji, H. Ye, J. Liu, Y. Jia, J. Yu, and J. Cheng, "A survey of autonomous robots and multi-robot navigation: Perception, planning and collaboration," *Biomimetic Intelligence and Robotics*, p. 100203, 2024.

[17] R. M. Francos and A. M. Bruckstein, "On the role and opportunities in teamwork design for advanced multi-robot search systems," *Frontiers in Robotics and AI*, vol. 10, p. 1089062, 2023.

[18] A. Gautam and S. Mohan, "A review of research in multi-robot systems," in *2012 IEEE 7th international conference on industrial and information systems (ICIIS)*. IEEE, 2012, pp. 1–5.

[19] Z. Liu, H. Wang, S. Zhou, Y. Shen, and Y.-H. Liu, "Coordinating large-scale robot networks with motion and communication uncertainties for logistics applications," *arXiv preprint arXiv:1904.01303*, 2019.

[20] Z. Liu, S. Zhou, H. Wang, Y. Shen, H. Li, and Y.-H. Liu, "A hierarchical framework for coordinating large-scale robot networks," in *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 6672–6677.

[21] B. Hu and R. Yang, "A kubeedge-based multi-robot collaboration framework for perception, planning and navigation," in *2022 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, 2022, pp. 1186–1191.

[22] A. Singhal, P. Pallav, N. Kejriwal, S. Choudhury, S. Kumar, and R. Sinha, "Managing a fleet of autonomous mobile robots (amr) using cloud robotics platform," in *2017 European Conference on Mobile Robots (ECMR)*, 2017, pp. 1–6.

[23] E. Dey, M. Walczak, M. S. Anwar, N. Roy, J. Freeman, T. Gregory, N. Suri, and C. Busart, "A novel ros2 qos policy-enabled synchronizing middleware for co-simulation of heterogeneous multi-robot systems," in *2023 32nd International Conference on Computer Communications and Networks (ICCCN)*. IEEE, 2023, pp. 1–10.

[24] J. Wan, S. Tang, H. Yan, D. Li, S. Wang, and A. V. Vasilakos, "Cloud robotics: Current status and open issues," *Ieee Access*, vol. 4, pp. 2797–2807, 2016.

[25] T. Kronauer, J. Pohlmann, M. Matthé, T. Smejkal, and G. Fettweis, "Latency analysis of ros2 multi-node systems," in *2021 IEEE international conference on multisensor fusion and integration for intelligent systems (MFI)*. IEEE, 2021, pp. 1–7.

[26] A. Kaknjo, M. Rao, E. Omerdic, L. Robinson, D. Toal, and T. Newe, "Real-time video latency measurement between a robot and its remote control station: Causes and mitigation," *Wireless Communications and Mobile Computing*, vol. 2018, no. 1, p. 8638019, 2018.

[27] Y. F. Chen, M. Liu, M. Everett, and J. P. How, "Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 285–292.

[28] S. H. Semnani, H. Liu, M. Everett, A. de Ruiter, and J. P. How, "Multi-agent motion planning for dense and dynamic environments via deep reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3221–3226, 2020.

[29] K. Ceder, Z. Zhang, A. Burman, I. Kuangaliyev, K. Mattsson, G. Nyman, A. Petersén, L. Wisell, and K. Åkesson, "Bird's-eye-view trajectory planning of multiple robots using continuous deep reinforcement learning and model predictive control," in *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2024, pp. 8002–8008.

[30] J. Choi, F. Castaneda, C. J. Tomlin, and K. Sreenath, "Reinforcement learning for safety-critical control under model uncertainty, using control lyapunov functions and control barrier functions," *arXiv preprint arXiv:2004.07584*, 2020.

[31] L. Wang, A. D. Ames, and M. Egerstedt, "Safety barrier certificates for collisions-free multirobot systems," *IEEE Transactions on Robotics*, vol. 33, no. 3, pp. 661–674, 2017.

[32] J. Berlin, G. Hess, A. Karlsson, W. Ljungbergh, Z. Zhang, K. Åkesson, and P.-L. Götvall, "Trajectory generation for mobile robots in a dynamic environment using nonlinear model predictive control," in *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*. IEEE, 2021, pp. 942–947.

[33] C. E. Luis, M. Vukosavljev, and A. P. Schoellig, "Online trajectory generation with distributed model predictive control for multi-robot motion planning," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 604–611, 2020.

[34] H. Zhu, F. M. Claramunt, B. Brito, and J. Alonso-Mora, "Learning interaction-aware trajectory predictions for decentralized multi-robot motion planning in dynamic environments," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 2256–2263, 2021.

[35] L. Ferranti, L. Lyons, R. R. Negenborn, T. Keviczky, and J. Alonso-Mora, "Distributed nonlinear trajectory optimization for multi-robot motion planning," *IEEE Transactions on Control Systems Technology*, vol. 31, no. 2, pp. 809–824, 2023.

[36] L. Li, J. Li, and S. Zhang, "Review article: State-of-the-art trajectory tracking of autonomous vehicles," *Mechanical Sciences*, vol. 12, no. 1, pp. 419–432, 2021. [Online]. Available: https://ms.copernicus.org/articles/12/419/2021/

[37] H. Li, J. Luo, S. Yan, M. Zhu, Q. Hu, and Z. Liu, "Research on parking control of bus based on improved pure pursuit algorithms," in *2019 18th International Symposium on Distributed Computing and Applications for Business Engineering and Science (DCABES)*, 2019, pp. 21–26.

[38] E. Alcala, V. Puig, J. Quevedo, T. Escobet, and R. Comasolivas, "Autonomous vehicle control using a kinematic lyapunov-based technique with lqr-lmi tuning," *Control Engineering Practice*, vol. 73, pp. 1–12,

2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0967066117302721

[39] A. S. Tanenbaum and M. Van Steen, *Distributed systems*. CreateSpace Independent Publishing Platform, 2017.

[40] S. Santra and P. P. Acharjya, "A study and analysis on computer network topology for data communication," *International Journal of Emerging Technology and Advanced Engineering*, vol. 3, no. 1, pp. 522–525, 2013.

[41] Y. Leng, C. Yu, W. Zhang, Y. Zhang, X. He, and W. Zhou, "Task-oriented hierarchical control architecture for swarm robotic system," *Natural Computing*, vol. 16, pp. 579–596, 2017.

[42] H. Ebel, "Distributed control and organization of communicating mobile robots: design, simulation, and experimentation," 2021.

[43] F. Bertilsson, M. Gordon, J. Hansson, D. Möller, D. Söderberg, Z. Zhang, and K. Åkesson, "Centralized versus distributed nonlinear model predictive control for online robot fleet trajectory planning," in *2022 IEEE 18th International Conference on Automation Science and Engineering (CASE)*. IEEE, 2022, pp. 701–706.

[44] A. Sathya, P. Sopasakis, R. Van Parys, A. Themelis, G. Pipeleers, and P. Patrinos, "Embedded nonlinear model predictive control for obstacle avoidance using panoc," in *2018 European control conference (ECC)*. IEEE, 2018, pp. 1523–1528.

[45] Wikipedia contributors, "Model predictive control — Wikipedia, the free encyclopedia," 2025. [Online]. Available: https://en.wikipedia.org/wiki/Model_predictive_control

A

Appendix 1

Source Code Repository

The implementation code for this thesis is available in a public GitHub repository:
`https://github.com/lutzzzzz/Distribute_mpc_ros2`
The repository "Toward Efficient Collaboration in Autonomous Mobile Robot Fleets: Addressing Latency and Distributed Model Predictive Control" contains all necessary components to reproduce the experiments described in this work.

The MPC implementation in this work draws significant inspiration from the methodologies presented in the repository
`https://github.com/Woodenonez/DyObAv-MPCnEBM-Warehouse`
The AMR robot in the Gazebo model originates from MiR100 from the repository:
`https://github.com/DFKI-NI/mir_robot`

System Requirements and Environment

The implementation requires the following environment:

- Operating System: Ubuntu 20.04
- ROS2: Foxy distribution
- Python: Version 3.8
- OpEn: Optimization Engine for MPC solver

Hardware Specifications

All simulations in this thesis were conducted on a workstation with the following specifications:

- **Processor:** Intel Core i5-13600KF (14 cores, 5.1 GHz max turbo frequency)
- **Graphics Processing Unit:** NVIDIA GeForce RTX 4070 with 12GB GDDR6X VRAM
- **Memory:** 32GB DDR5 RAM operating at 6400 MHz
- **Storage:** 1TB SSD read speeds up to 7200 MB/s and write speeds up to 6300 MB/s

DEPARTMENT OF Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY