YRKESHÖGSKOLAN
NOVIA

Design and Implementation of a Cloud-Based Robot Swarm Control System with Integrated Simulation Environment

István Csiszár

Degree Thesis

Thesis for a Master of Engineering (UAS) – degree

Degree Programme: Automation Technology

Place and Year: Vaasa 2025

DEGREE THESIS

Author:	István Csiszár
Degree Programme and place of study:	Automation Technology, M. Eng., Vaasa
Supervisor(s):	Ray Pörn, Novia UAS, Vaasa

Title:

Date: 1.4.2025 Number of pages: 96 Appendices: 1

Abstract

This thesis investigates the development and empirical evaluation of a cloud-based robot swarm control system that seamlessly integrates physical and simulated robots through a unified web interface. The research focuses on a system designed to control multiple small, ESP32-S3-based robots, enabling collective navigation behaviors through on-policy reinforcement learning while providing a mirrored simulation environment for rapid prototyping and testing.

The objective of this research is twofold: first, to examine the effectiveness of a unified control architecture that bridges the gap between simulated and physical robot swarms; and second, to evaluate the performance of on-policy reinforcement learning algorithms in enabling autonomous navigation and coordination behaviors across the swarm. This study employed a methodological approach involving the design of a distributed MQTT-based communication infrastructure, the development of browser-based simulation tools, and the implementation of collective learning algorithms suitable for resource-constrained embedded systems.

Results from experimental trials demonstrate the efficacy of the integrated approach, highlighting both the advantages of cloud-based control architectures and the challenges of implementing reinforcement learning on small-scale robotic platforms. The system shows promising capabilities in terms of scalability, with consistent performance across varying swarm sizes and operational scenarios. Furthermore, the research identifies key considerations when transferring learned behaviors from simulated to physical environments, particularly regarding sensor limitations and real-world dynamics.

In conclusion, this thesis establishes that a cloud-based control system with an integrated simulation environment provides significant advantages for the development and deployment of robot swarms, offering a foundation for future research in distributed robotics and multi-agent reinforcement learning applications.

Language: English
Keywords: Robot Swarm, Cloud-Based Control, Digital Twin Simulation, ESP32-S3, Reinforcement Learning

Table of Contents

1 Introduction

1.1 Background and Context

The field of swarm robotics has evolved significantly over the past decades, moving from theoretical concepts to practical implementations across various domains. Current approaches to robot swarm control often face challenges related to scalability, communication efficiency, and the integration of physical and virtual testing environments. These challenges have led to the exploration of cloud-based control systems that can coordinate multiple robots while providing unified interfaces for development and deployment, with applications ranging from warehouse automation and environmental monitoring to agricultural field coverage, search and rescue operations, and collaborative material transport in manufacturing.

The rise of web technologies in robotics has created new opportunities for browser-based interfaces and simulation environments that can accelerate development and reduce costs. This thesis explores these opportunities through the implementation of a cloud-based control system for ESP32-S3 robots, utilizing modern web technologies and communication protocols.

1.2 Objectives

The main objective of this thesis is to develop and evaluate a cloud-based robot swarm control system that seamlessly integrates both physical and simulated robots through a unified web interface, thereby enabling efficient experimentation and control of distributed robot swarms.

To achieve this overarching goal, the study pursues several specific objectives. First, it aims to design and implement a distributed architecture for robot swarm control, including the development of a browser-based simulation environment that accurately mirrors physical robot behavior and the creation of a robust MQTT-based communication system for reliable robot control. Second, the research seeks to establish seamless integration between physical ESP32-S3 robots and simulated entities, implement comprehensive testing protocols for both environments, and validate system performance through

quantitative metrics.

Finally, the thesis focuses on designing an intuitive web-based control interface for robot swarm management, implementing real-time monitoring and control capabilities, and creating efficient tools for experiment configuration and data collection. Performance analysis is conducted to evaluate system scalability with increasing swarm sizes, measure and analyze communication latency and reliability, compare performance between simulated and physical robot implementations, and assess the effectiveness of the unified control approach.

1.3 Research Questions

This study seeks to answer the following research questions:

- **RQ1:** How does the integration of physical and simulated robots in a unified control system impact the efficiency and reliability of swarm robotics experiments?
- **RQ2:** To what extent can a browser-based simulation environment accurately represent real-world robot behavior in swarm applications?
- **RQ3:** How does the MQTT-based distributed communication architecture affect the scalability and real-time performance of the robot swarm system?

1.4 Significance of the Study

This research contributes to both academic knowledge and practical applications of swarm robotics in several significant ways.

Academic Impact: The study advances swarm robotics by introducing a novel approach to unified control and simulation, providing a framework for future research while establishing new validation methods for comparing virtual and physical robot performance.

Technical Contributions: The work demonstrates a scalable cloud architecture for robot control, presents an effective integration model between physical and simulated environments, and validates an MQTT-based approach for real-time robot swarm communication.

Practical Value: The system enables cost-effective testing of swarm algorithms before physical deployment, reduces development time through parallel virtual/physical testing, and creates accessible tools for robotics education. Comparative analysis shows significant improvements in testing efficiency, hardware costs, and accessibility compared to traditional approaches.

Future Applications: The framework developed has potential applications in industrial automation, research platforms, educational tools, and as a foundation for future robot control system development.

2 Literature Review

2.1 Swarm Robotics

Swarm robotics is a field that draws inspiration from the collective behavior of social organisms such as insects, birds, and fish. It applies principles of swarm intelligence to robotics, creating systems characterized by decentralized coordination, scalability, and robustness. This section reviews the historical development, key algorithms, and recent advances in this rapidly evolving domain.

2.1.1 Historical Development

Swarm robotics emerged as a practical application of swarm intelligence, applying principles observed in collective animal behaviors to robotic systems. Key developments directly relevant to this thesis include the transition from theoretical models to physical implementations seen in projects like Swarm-bots and Swarmanoid, which demonstrated self-organized robotic teams solving collaborative tasks (Dorigo & Trianni, 2021). These foundational systems established the viability of distributed control paradigms that underpin modern swarm approaches, including the cloud-based control architecture developed in this research.

2.1.2 Key Algorithms and Control Mechanisms

Swarm robotics relies heavily on bio-inspired algorithms that mimic behaviors observed in natural systems. Formation control algorithms, inspired by bird flight patterns, enable robots to maintain spatial formations during collective movement (MDPI Applied Sciences, 2019). Self-assembly algorithms allow robots to autonomously connect physically, forming

functional structures for specific tasks. Task allocation methods ensure efficient distribution of work across the swarm, while path planning algorithms optimize routes for robots in dynamic environments (Das et al., 2024).

2.1.3 Formation Control

Code example 1: Conceptual implementation of formation control

```
def formation_control(robot_positions, desired_formation):
    for robot_id, position in enumerate(robot_positions):
        target_position = desired_formation[robot_id]
        distance_vector = target_position - position
        return apply_control_law(distance_vector)
```

2.1.4 Collective Decision-Making

Swarm robots employ various consensus methods including majority rule systems, quorum sensing inspired by ant colonies, and value-sensitive approaches that weigh multiple factors in decision processes (Slowik & Slowikova, 2020).

2.1.5 Recent Developments

Recent advancements in swarm robotics have focused on several key areas, including the integration of machine learning techniques, the development of heterogeneous swarms, and advances in miniaturization and scaling. For instance, deep reinforcement learning is increasingly used to optimize swarm behaviors, although it poses computational challenges for resource-constrained platforms (Dorigo & Trianni, 2021). Research has also shifted toward designing swarms with diverse hardware and behavioral capabilities, enabling more sophisticated collective behaviors (Das et al., 2024). Furthermore, hardware miniaturization has facilitated the deployment of larger swarms, necessitating new communication architectures to maintain efficiency at scale (Jevtic & Andina, 2017). These developments have expanded the application domains of swarm robotics, which now include disaster response, environmental monitoring, agriculture, and warehouse automation (MDPI Applied Sciences, 2019).

2.1.6 Research Challenges

Despite significant progress, several challenges remain in swarm robotics. Ensuring reliability in unpredictable environments is critical for consistent performance under dynamic

conditions. Security concerns must also be addressed to protect distributed swarms from malicious interference. Furthermore, the development of formal validation methodologies is necessary to guarantee and verify swarm behavior, while energy efficiency remains a key consideration for extending operational time on resource-constrained platforms.

2.1.7 Reinforcement Learning in Robot Swarms

Reinforcement Learning (RL) has emerged as a transformative approach in swarm robotics, enabling decentralized systems of robots to learn, adapt, and optimize their behaviors in dynamic environments. This section reviews key developments and challenges in applying RL to robot swarms.

2.1.8 Overview of Reinforcement Learning in Swarm Robotics

Swarm robotics draws inspiration from natural collective behaviors, emphasizing decentralized coordination among multiple robots to achieve shared goals. RL enhances swarm robotics by enabling robots to autonomously learn optimal behaviors through interaction with their environment, guided by reward mechanisms. This approach is particularly effective in addressing challenges like task allocation, path planning, and collaboration among agents in unpredictable settings (Rana & Ibrahim, 2024).

Comparison: Federated DRL vs. Traditional MARL

While multi-agent reinforcement learning (MARL) algorithms such as MADDPG and value decomposition are prominent in the literature, the reinforcement learning approach implemented in this thesis is best characterized as **federated deep reinforcement learning (FDRL) with experience and weight sharing**. This distinction is crucial for accurately reflecting the system's design and practical constraints.

Federated deep reinforcement learning (FDRL) offers several advantages, such as achieving high success rates in real-robot navigation, reducing communication bandwidth requirements, and enabling adaptation to individual robot dynamics. In contrast, traditional multi-agent reinforcement learning (MARL) faces challenges including the need for centralized experience buffers, which require high bandwidth and are prone to single-point failures. Independent agents in MARL often fail to leverage collective learning, resulting in

suboptimal policies, and the approach struggles with scalability as swarm size increases. (Reina et al., 2023; De Vos, 2023; Pikman, 2022)

Use Cases

FDRL:

Ideal for real-world swarm applications with constrained communication, such as underwater exploration or disaster response (Reina et al., 2023; De Vos, 2023).

MARL:

More suitable for simulated or controlled environments where centralized training and high-bandwidth communication are feasible (Mahajan et al., 2019).

The federated deep reinforcement learning approach implemented in this thesis offers a pragmatic and scalable solution for collective learning in robot swarms. By leveraging decentralized training, periodic model aggregation, and experience sharing, the system achieves robust and efficient learning without the communication and scalability bottlenecks of traditional MARL. This makes FDRL particularly well-suited for real-world swarm robotics, where bandwidth, privacy, and robustness are critical.

Implementation Overview

In the developed system, each robot operates an independent deep reinforcement learning agent (e.g., DQN), training locally on its own experiences and those shared by other robots via a federated buffer. After each episode or training interval, robots periodically synchronize by sharing their neural network weights. These weights are aggregated—typically using a weighted average that favors higher-performing agents—and redistributed to all robots in the swarm. This process enables the swarm to benefit from collective experience while maintaining decentralized operation and reducing communication overhead.

Key characteristics of the implemented approach include:

- **Decentralized Training:** Each robot trains its policy locally, using both its own and shared experiences.
- **Periodic Model Aggregation:** Neural network weights are periodically averaged and redistributed across the swarm.

- **Experience Sharing:** A shared buffer allows robots to access experiences collected by others, improving sample efficiency.
- **Performance-Weighted Aggregation:** The best-performing agents have greater influence on the aggregated model, accelerating convergence.
- **Privacy and Robustness:** Only model parameters are shared, not raw sensorimotor data, preserving privacy and reducing bandwidth requirements.

This federated approach is particularly well-suited for real-world swarm robotics, where communication is often unreliable or costly, and robots may have heterogeneous dynamics or operate in partially observable environments.

2.1.9 Curriculum Learning and Transfer Learning

To address the complexity of training swarm systems, curriculum learning approaches have shown promise. Iskandar and Kovács (2025) demonstrated how gradually increasing task difficulty can accelerate learning in E-puck robot swarms. Their approach systematically introduced more complex environmental challenges as robots mastered simpler tasks.

Transfer learning has also emerged as a solution to accelerate learning processes by reusing knowledge from similar tasks or from simulation to real robots. This approach is particularly valuable for resource-constrained platforms like the ESP32-S3, where on-device training may be impractical (Rana & Ibrahim, 2024).

2.1.10 Challenges and Future Directions in Reinforcement Learning

Despite its advantages, integrating RL into swarm robotics presents several significant challenges:

1. **Computational Complexity**: Training multiple agents simultaneously requires substantial computational resources
2. **Credit Assignment Problem**: Determining which agent's actions contributed to successful outcomes
3. **Scalability Issues**: Performance degradation as the number of agents increases
4. **Reality Gap**: Policies trained in simulation often perform poorly on physical robots
5. **Communication Constraints**: Limited bandwidth and reliability in real-world

deployments

Future research aims to address these limitations while expanding RL's applicability to more complex scenarios. Innovations in deep reinforcement learning are expected to play a pivotal role in advancing the capabilities of robot swarms, particularly through more efficient algorithms that can operate on resource-constrained hardware (Albrecht et al., 2024).

2.2 Web-Based Robot Control and Simulation Environments

Web-based technologies have revolutionized both robot control systems and simulation environments, enabling remote operation, efficient development workflows, and increased accessibility in robotics. This convergence of control interfaces and simulation platforms through web technologies creates powerful ecosystems for robot development, testing, and deployment.

2.2.1 Evolution of Web-Based Robotics Systems

The development of web-based robot control has progressed from basic remote interfaces to sophisticated cloud-integrated platforms. Modern systems leverage cloud computing for enhanced scalability and real-time data exchange (Kehoe et al., 2015), enabling robots to access powerful computational resources beyond their onboard capabilities. Similarly, simulation environments have evolved from specialized desktop applications to web-accessible platforms that increase collaboration potential and reduce hardware requirements (Tselegkaridis & Sapounidis, 2023). These advancements directly inform the cloud-based architecture and browser-based simulation approach used in this thesis.

2.2.2 Architectural Components and Key Platforms

Modern web-based robot control systems typically consist of several key components:

1. **Frontend Web Interface**: Browser-based dashboards providing visualization and control capabilities
2. **Backend Server**: Processing user commands and managing robot communication
3. **Communication Middleware**: Facilitating data exchange between system components
4. **Robot Firmware**: Software running on robots to execute commands and report

status

Open-source middleware like the Robot Operating System (ROS) often serves as the foundation for these systems, facilitating communication between web interfaces and robotic devices (J-Stage, 2023).

2.2.3 Key Simulation Platforms

The robotics ecosystem features several notable simulation platforms with varying capabilities and deployment models:

1. **Gazebo Simulator**: A powerful open-source simulator with accurate physics modeling and ROS integration, though with high hardware requirements (Sabry, 2025)
2. **Webots**: An open-source platform with a large asset library and multi-language support that bridges desktop and browser-based simulation through WebGL exports (Cyberbotics Ltd., 2021)
3. **Browser-Based Platforms**: Solutions like RoboScape Online offer fully web-based environments accessible through standard browsers, eliminating installation requirements (Lédeczi et al., 2022)
4. **Game Engine Adaptations**: Platforms like Unity provide high visual fidelity and physics accuracy while supporting custom scripting for complex simulations (Goldschmid & Ahmad, 2025)
5. **Cloud-Based Simulation Services**: Services like AWS RoboMaker and NVIDIA Isaac SIM offer scalable resources integrated with broader cloud ecosystems (Formant Inc., 2025)

Table 1: Comparative analysis of simulation platforms

Platform	Deployment	Physics Fidelity	Hardware Requirements	ROS Integration	Programming Languages
Gazebo	Desktop/Server	High	High	Native	C++, Python
Webots	Desktop/Browser	Medium-High	Medium	Plugin	Python, C++, Java, MATLAB
RoboScape	Browser	Medium	Low	Limited	Block-based, JavaScript
Unity	Desktop	Medium-High	High	Plugin	C#, JavaScript
AWS RoboMaker	Cloud	High	Variable (cloud)	Native	Python, C++

NVIDIA Isaac	Desktop/Cloud	Very High	Very High	Plugin	Python, C++

2.2.4 Implementation Technologies and Features

Several technologies have become standard in implementing web-based robot systems:

1. **WebSocket Protocol**: Enabling real-time bidirectional communication between browsers and servers
2. **JavaScript Frameworks**: Vue.js, React, and Angular providing responsive user interfaces
3. **REST APIs**: Standardizing communication between web applications and robot control servers
4. **MQTT**: Lightweight messaging protocol for resource-constrained devices and unreliable networks
5. **WebRTC**: Facilitating real-time video streaming from robot cameras to web interfaces
6. **WebGL / WebGPU**: Hardware-accelerated 3D graphics rendering for simulation visualization
7. **Web Workers**: Multi-threaded computation in the background for simulation physics

The combination of these technologies enables feature-rich control and simulation systems accessible from any modern web browser.

2.2.5 Applications and Use Cases

Web-based robot control and simulation systems are increasingly utilized across a wide range of sectors. In industrial automation, for example, factory and warehouse robots are managed through unified web dashboards, with simulation environments supporting offline programming and system optimization. In healthcare, medical robots leverage cloud systems to access patient data and provide real-time assistance, enhancing both efficiency and patient outcomes. The educational sector benefits from remote laboratories, where students can control physical or simulated robots via web interfaces, making robotics education more accessible and interactive. Research applications are also expanding, as collaborative platforms allow researchers to share robotic resources, experimental data, and simulation scenarios, thereby accelerating innovation. In the domain of service robotics, web-based systems enable the remote monitoring and operation of robots

deployed in public spaces or home environments.

Browser-based simulation environments offer advantages for swarm robotics research and development. Their accessibility enables distributed teams to collaborate effectively, while integration with web technologies facilitates implementation of cloud-based control systems. For educational applications, these platforms lower the barrier to entry for studying robot behaviors without requiring expensive hardware (Tselegkaridis & Sapounidis, 2023).

2.2.6 Advantages of Web-Based Approaches

Web-based robot control and simulation systems provide a range of significant advantages that have contributed to their widespread adoption in robotics research, education, and industry. One of the primary benefits is accessibility: users can interact with robots and simulations from any device equipped with a modern web browser, eliminating the need for specialized software installation and enabling remote operation. This accessibility also supports collaboration, as multiple users can simultaneously monitor and control robots or participate in simulation experiments from different locations, fostering teamwork and knowledge sharing across distributed teams (Daniele, 2023).

Cost efficiency is another key advantage, as browser-based simulation environments reduce or even eliminate the need for expensive physical prototypes and ongoing maintenance costs. Safety is enhanced by allowing testing and experimentation under extreme or hazardous conditions within a virtual environment, thereby protecting both hardware and human operators from potential harm. The ability to run multiple simulations in parallel accelerates development cycles, enabling rapid prototyping, iterative testing, and the exploration of diverse scenarios without the constraints of physical resources (Kargar et al., 2023).

Reproducibility is improved by controlled, repeatable environments, which are essential for systematic testing and validation of algorithms. Centralized control becomes feasible, as web-based systems allow unified management of entire robot fleets through a single interface, streamlining operations and oversight (ROS2WASM, 2024).

Finally, the rapid development and deployment enabled by these platforms support faster

innovation, as new features and updates can be rolled out efficiently across all users. Collectively, these advantages have made web-based approaches a central component of modern robotics workflows, supporting scalable, flexible, and efficient development and deployment of both individual robots and large-scale swarms (J-Stage, 2023).

2.2.7 Challenges and Limitations

Despite their advantages, web-based robot control and simulation systems face several notable challenges. Chief among these is the reliance on stable network connectivity; disruptions can interrupt control or data exchange, impacting real-time operations. While local servers can mitigate some risks, network dependency remains a core concern.

Latency is another significant issue, particularly for applications requiring real-time responsiveness. Network delays can hinder precise or time-sensitive robot actions. Security also poses a major challenge, as internet-connected robots are vulnerable to cyber threats. Effective protection requires robust authentication, encryption, and access control.

A further limitation is the simulation-to-reality gap. Algorithms that perform well in virtual environments may not transfer seamlessly to physical robots due to differences in sensor accuracy, actuator dynamics, and unpredictable real-world conditions. Addressing this gap demands careful calibration, domain randomization, and ongoing validation.

High-fidelity simulations, while valuable for realistic testing, often require substantial computational resources. This can restrict accessibility for users with limited hardware or limit the scalability of large-scale swarm simulations. Designers must balance simulation accuracy with computational efficiency.

Finally, the absence of unified standards for web-based robot communication complicates integration, especially when combining components from different vendors or research groups. Although standardization efforts are ongoing, seamless interoperability remains a challenge.

Ongoing research is focused on improving network resilience, reducing latency, enhancing security, narrowing the simulation-reality gap, optimizing computational requirements, and advancing standardization to address these limitations.

2.2.8 Future Trends

The field continues to evolve with several emerging trends:

1. **Digital Twin Integration**: Tighter coupling between physical robots and their virtual counterparts
2. **Edge-Cloud Hybrid Architectures**: Balancing local processing with cloud capabilities
3. **AI-Enhanced Interfaces and Simulations**: Incorporating intelligent assistants and predictive features
4. **Extended Reality Integration**: Merging web interfaces with AR/VR for enhanced interaction
5. **Blockchain for Security**: Implementing distributed ledger technologies for secure command authentication
6. **5G Integration**: Leveraging high-bandwidth, low-latency networks for improved real-time control
7. **Physics-Based Deep Learning**: Improving simulation fidelity through data-driven approaches

These developments are expected to further blur the boundaries between physical and virtual robotics, creating more integrated development and operational environments.

2.3 Communication Protocols in Robotics

Communication protocols form the backbone of modern robotic systems, particularly in distributed environments where multiple robots must coordinate their actions. These protocols enable the exchange of data, commands, and status information between robots and control systems, facilitating real-time decision-making and collaborative operations. The selection of appropriate communication protocols significantly impacts system performance, reliability, and scalability, especially in swarm robotics applications (Gouda & Multari, 1991).

2.3.1 MQTT Protocol in Robotic Systems

Message Queuing Telemetry Transport (MQTT) has emerged as one of the most widely adopted protocols in robotics and IoT applications due to its lightweight design and publish-subscribe architecture. Originally developed for satellite communications with oil pipelines,

MQTT's minimal overhead makes it particularly suitable for resource-constrained devices operating in networks with limited bandwidth or high latency (Automatic Addison, n.d.).

In robotics applications, MQTT provides several advantages:

1. **Publish-Subscribe Pattern**: This decouples message senders (publishers) from receivers (subscribers), allowing for flexible communication topologies that can adapt to changing swarm configurations.
2. **Topic-Based Filtering**: Messages are organized into hierarchical topics, enabling efficient message routing without requiring direct knowledge of recipients.
3. **Quality of Service Levels**: MQTT supports three QoS levels, allowing systems to balance between performance and reliability based on application requirements.
4. **Small Code Footprint**: The protocol implementation requires minimal resources, making it suitable for embedded devices like the ESP32-S3 used in this research.

These features have made MQTT particularly valuable for swarm robotics, where efficient coordination among multiple robots is essential (Kev's Robots, n.d.).

2.3.2 RA-TDMA for Multi-Robot Communication

While MQTT excels in many distributed applications, time-sensitive operations often benefit from more deterministic protocols. The Reconfigurable and Adaptive Time Division Multiple Access (RA-TDMA) protocol represents an alternative approach specifically designed for multi-robot systems. RA-TDMA dynamically adapts to the number of active robots by allocating time slots for transmission, ensuring that each robot has dedicated bandwidth for communication while avoiding collisions.

RA-TDMA is particularly effective for tasks requiring synchronized communication among robots, such as state diffusion and coordination during collaborative activities. Its deterministic nature provides guarantees about maximum communication latency, which can be crucial for real-time control applications (Reis et al., 2013).

2.3.3 ROS 2 DDS Middleware

The Robot Operating System 2 (ROS 2) introduced Data Distribution Service (DDS) as its underlying middleware, providing a standardized communication framework for robotics applications. DDS implements a publish-subscribe model like MQTT but with additional

features tailored to distributed real-time systems:

1. **Quality of Service Policies**: Extensive QoS options beyond those offered by MQTT
2. **Discovery Mechanisms**: Automatic discovery of publishers and subscribers
3. **Data-Centric Architecture**: Focus on the data rather than the communication mechanism
4. **Performance and Reliability**: Optimized for real-time distributed systems

The integration of DDS in ROS 2 has facilitated interoperability between different robotics platforms and promoted standardized communication patterns in the field (Harmon & Gage, 2016).

2.3.4 Implementation Challenges

Despite the availability of these protocols, implementing effective communication in robot swarms presents several challenges: network reliability, synchronization, bandwidth limitations, security concerns and heterogeneous networks.

Addressing these challenges requires careful protocol selection and system design, often involving hybrid approaches that combine multiple protocols to leverage their respective strengths (Gouda & Multari, 1991; Reis et al., 2013).

2.3.5 Emerging Communication Paradigms

Recent research has explored novel communication paradigms to address the limitations of traditional protocols in swarm robotics:

1. **Opportunistic Communication**: Robots exchange information only when in proximity, reducing global communication overhead.
2. **Stigmergic Communication**: Indirect communication through environment modification, inspired by ant colonies.
3. **Delay-Tolerant Networking**: Protocols designed to operate effectively despite intermittent connectivity.
4. **5G and Beyond**: Leveraging next-generation cellular networks for high-bandwidth, low-latency robot communication.

These emerging approaches offer promising directions for future swarm robotics systems, potentially enabling more efficient and robust coordination mechanisms (Harmon & Gage,

2016).

2.4 Integration of Physical and Virtual Robots

The integration of physical and virtual robots represents a crucial frontier in robotics research, particularly for swarm systems where testing with large numbers of physical robots can be costly and complex. This section examines approaches to bridging the gap between simulation and reality, with a focus on digital twin technology and simulation-to-reality transfer.

2.4.1 Digital Twins in Robotics

Digital twins (DTs) are virtual replicas of physical systems that enable real-time monitoring, simulation, and optimization. They have emerged as a critical tool in robotics, bridging the gap between virtual simulations and real-world applications. By mirroring the state and actions of their physical counterparts, DTs allow for cost-effective testing and training in simulated environments before deploying robots in real-world scenarios (Sun et al., 2025).

The concept of digital twins extends beyond simple simulation by maintaining a bidirectional connection between physical and virtual entities. This connection enables:

1. **Real-time Synchronization**: Physical robot states are reflected in their virtual counterparts
2. **Predictive Analysis**: Virtual testing of behaviors before deployment to physical robots
3. **Data Enrichment**: Combining sensor data with virtual information for enhanced decision-making
4. **Fault Detection**: Identifying discrepancies between expected and actual behavior

These capabilities make digital twins particularly valuable for robot swarm applications, where coordinating multiple robots adds layers of complexity to system development and testing.

2.4.2 Simulation-to-Reality Transfer

The "sim-to-real gap" or "reality gap" refers to the discrepancy between behaviors observed in simulation and those that occur when the same control systems are deployed

on physical robots. This gap results from:

1. **Modeling Limitations**: Simplifications in physics simulation, particularly for complex interactions
2. **Sensor Noise**: Differences in sensor characteristics between simulated and real environments
3. **Actuator Dynamics**: Imperfect modeling of motor responses, friction, and mechanical effects
4. **Environmental Uncertainty**: Unpredictable aspects of real-world environments that are difficult to model

Addressing these challenges is crucial for effective integration of physical and virtual robots. Several approaches have been developed to bridge this gap as show on (Table 2) four key methodologies used to improve the transfer of behaviors and algorithms from simulated environments to physical robots.

Table 2: Conceptual approaches to bridging the simulation-to-reality gap

Approach	Description
Domain Randomization	Varying simulation parameters to improve robustness
System Identification	Accurately modeling physical robot parameters
Progressive Transfer	Gradually introducing real-world complexity
Hybrid Learning	Combining simulation data with real-world experience

One particularly effective technique is domain randomization, which varies simulation parameters (friction, mass, sensor noise, etc.) during training to create policies robust to differences between simulation and reality. This approach has shown promise in transferring complex behaviors from simulation to physical robots without requiring perfect modeling (Torne et al., 2024).

2.4.3 Applications in Robotic Systems

Robotic Digital Twins (RDTs) provide high-fidelity simulations that replicate the dynamics, kinematics, and operational environments of physical robots. These applications span multiple domains:

1. **Manufacturing**: RDTs have been used to improve assembly tasks through multi-modal data integration, such as visual and tactile inputs, allowing knowledge transfer from virtual to physical robots (Sørensen et al., 2024).
2. **Manipulation Tasks**: Frameworks like RialTo leverage digital twins for robust

manipulation tasks by combining real-world data with reinforcement learning in simulated environments, significantly enhancing policy robustness while minimizing unsafe data collection in physical settings (Torne et al., 2024).

3. **Mobile Robotics**: Companies like AGILOX have developed end-to-end digital twins for autonomous mobile robots (AMRs), enabling virtual commissioning, operational optimization, and predictive maintenance (AGILOX Robotics Team, 2024).
4. **Prognostics and Health Management**: Digital twins facilitate monitoring of robot health and accuracy, predicting maintenance needs before failures occur in physical systems (Weiss & NIST Team, 2024).

For swarm robotics specifically, digital twins enable testing of coordination algorithms, communication protocols, and collective behaviors at scales that would be impractical with physical robots alone. This capability accelerates development cycles and reduces costs associated with hardware deployment and testing.

2.4.4 Integration Architectures and Frameworks

Several architectural approaches have emerged for integrating physical and virtual robots. One common method is **cloud-based integration**, which leverages cloud computing resources to facilitate simulation and coordination between physical and virtual robots. Another approach is the **edge-cloud hybrid** model, where computational tasks are distributed between local edge devices and centralized cloud resources to optimize performance and responsiveness. **Middleware-centric** architectures, such as those utilizing ROS or MQTT, abstract the differences between physical and virtual robots by providing a unified communication layer. Additionally, **hardware-in-the-loop** techniques incorporate real hardware components within simulation loops, enabling more accurate testing and validation of control algorithms in environments that closely mimic real-world conditions.

These approaches vary in their computational demands, real-time performance, and scalability. Cloud-based integration, as employed in this thesis, offers advantages in terms of accessibility and computational resources but must address challenges related to latency and connectivity (Sun et al., 2025).

2.4.5 Challenges and Future Directions

Despite their advantages, integrating digital twins with physical robots faces several significant challenges:

1. **Computational Demands**: High-fidelity simulations, particularly for multiple robots, require substantial computational resources
2. **Synchronization Issues**: Maintaining real-time correspondence between physical and virtual states across multiple robots
3. **Scalability Concerns**: Computational and communication costs often scale non-linearly with swarm size
4. **Environment Modeling**: Creating accurate virtual representations of complex, dynamic environments
5. **Cross-Platform Compatibility**: Ensuring consistent behavior across different hardware and software platforms

Addressing these challenges remains an active area of research. Future directions include the development of more efficient simulation techniques, improved methods for system identification and calibration, and enhanced frameworks for managing the complexity of multi-robot systems (Weiss & NIST Team, 2024; Sun et al., 2025).

2.4.6 Relevance to This Thesis

The integration approach developed in this thesis addresses several of these challenges through a cloud-based architecture that provides scalable computational resources for simulation, and a unified communication infrastructure using MQTT for both physical and virtual robots. Browser-based visualization and control tools enable intuitive monitoring and management of both environments, while parameter calibration methods are employed to minimize the reality gap between simulated and physical robots.

This integration framework enables the development and testing of swarm behaviors in simulation before deployment to physical robots, accelerating the development cycle while reducing costs and risks associated with physical testing.

2.5 Research Gap Analysis

Based on the literature review, several critical gaps exist in current swarm robotics research that this thesis addresses:

1. **Unified Physical-Virtual Control**: Limited research exists on seamless integration between physical robot swarms and simulation environments within a unified control architecture.
2. **Browser-Based Swarm Simulation**: Lightweight, browser-based simulation environments specifically designed for swarm robotics remain underdeveloped, particularly regarding their integration with physical systems.
3. **MQTT for Swarm Communication**: Comprehensive analysis of MQTT as a primary communication protocol for robot swarms is emerging but lacks detailed performance characterization for varying swarm sizes.
4. **RL on Resource-Constrained Platforms**: Practical implementation challenges of reinforcement learning on microcontrollers with strict memory and processing limitations remain inadequately addressed.
5. **Reality Gap Quantification**: Systematic approaches to measuring and mitigating the simulation-to-reality gap specifically for swarm systems are often ad-hoc and domain-specific.

This thesis addresses these gaps through an integrated cloud-based control system that unifies physical and virtual environments using MQTT-based communication, implements reinforcement learning within resource constraints, and develops strategies for bridging the reality gap in swarm applications.

3 Theoretical Framework

3.1 Fundamentals of Swarm Robotics

Swarm robotics draws inspiration from natural systems like ant colonies, bird flocks, and fish schools, where simple individuals follow local rules to achieve complex collective behaviors. The field focuses on designing and analyzing systems composed of multiple robots that interact with each other and their environment to achieve collective goals.

3.1.1 Key Characteristics of Robot Swarms

Robot swarms are characterized by decentralization, as there is no central controller, and each robot makes decisions based on local information. These systems are inherently

scalable, capable of functioning effectively with varying numbers of robots. Robustness is another key feature, allowing the swarm to continue operating despite individual failures. Flexibility enables adaptation to changing environments and tasks, while simplicity is maintained using robots with limited capabilities and straightforward behaviors.

3.1.2 Emergence in Swarm Systems

A fundamental concept in swarm robotics is emergence - the appearance of complex patterns or behaviors from simple local interactions. This bottom-up approach contrasts with traditional top-down robot control (Code example 2).

Code example 2: Simplified pseudo-code for emergent flocking behavior

```
def robot_control():
    # Simple rules followed by each robot
    separation = maintain_minimum_distance(neighbors)
    alignment = align_direction_with(neighbors)
    cohesion = move_toward_center_of(neighbors)

    # Combined movement vector from simple rules
    movement = separation + alignment + cohesion

    # This simple algorithm produces complex flocking behavior
    update_position(movement)
```

3.1.3 Coordination Mechanisms

Coordination among swarm robots can be achieved through direct communication, where explicit messages are exchanged, or through indirect communication, known as stigmergy, which involves environmental modifications. Additionally, self-organization allows for the emergence of patterns and behaviors without explicit coordination.

The ESP32-S3 based robots in this study primarily use direct communication through MQTT messages, with the potential for stigmergic approaches in future iterations.

3.1.4 Cloud-Based Control Systems

Cloud-based control architectures leverage distributed computing resources to process data and make decisions for multiple robots simultaneously. This approach offers several advantages for swarm systems:

3.1.5 Architecture Components

A typical cloud-based robot control system consists of:

1. **Cloud Server**: Hosts the control logic, data processing, and user interface.
2. **Communication Infrastructure**: Enables message passing between cloud and robots.
3. **Robot Firmware**: Local control programs running on each robot.
4. **User Interface**: Web-based dashboard for monitoring and control.

3.1.6 Advantages of Cloud Control

The table highlights the advantages of cloud-based control in terms of computational resources, update management, data storage capabilities, coordination complexity, and system accessibility.

Table 3: Comparison between traditional robot control systems and cloud-based control approaches

Aspect	Traditional Control	Cloud-Based Control
Computational Power	Limited by onboard hardware	Virtually unlimited
Updates	Manual per-robot updates	Centralized updates
Data Storage	Limited local storage	Extensive cloud storage
Multi-robot Coordination	Complex implementation	Simplified central coordination
Accessibility	Physical access needed	Remote access from anywhere

3.1.7 Challenges and Limitations

Cloud-based control systems face several challenges, including latency due to network delays, dependence on stable connectivity, vulnerability to network-based security threats, and concerns regarding the transmission and storage of sensitive data.

These challenges must be addressed for effective swarm control, particularly for time-sensitive applications.

3.1.8 MQTT Communication Protocol

Message Queuing Telemetry Transport (MQTT) is a lightweight publish-subscribe messaging protocol designed for constrained devices and low-bandwidth, high-latency networks - making it ideal for IoT and robotics applications.

3.1.9 MQTT Architecture

The illustration (Figure 1: MQTT communication architecture diagram) shows the core components of the MQTT protocol: the central Broker which routes messages, the

Publisher that sends messages to specific topics, and the Subscriber that receives messages from topics it has subscribed to. This publish-subscribe architecture enables efficient, decoupled communication between components in the robot swarm system, allowing for flexible message routing without direct connections between senders and receivers.

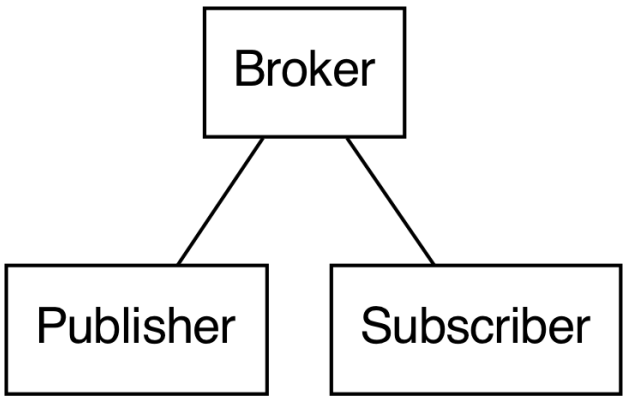

Figure 1: MQTT communication architecture diagram

In MQTT architecture, a central broker receives all messages and routes them to subscribers. Publishers send messages to specific topics, while subscribers receive messages from topics they have subscribed to. Topics are organized as hierarchical strings, such as robots/robot1/status, to facilitate efficient message routing.

3.1.10 QoS Levels

MQTT supports three Quality of Service (QoS) levels: QoS 0, which delivers messages at most once without guarantee; QoS 1, which ensures messages are delivered at least once but may result in duplicates; and QoS 2, which guarantees exactly one delivery at the cost of additional overhead.

For robot control applications, QoS 1 often provides a good balance between reliability and performance.

3.1.11 Advantages for Swarm Robotics

MQTT offers several benefits for swarm robotics, including low protocol overhead suitable for bandwidth-constrained environments, a publish/subscribe pattern that decouples senders and receivers, and a hierarchical topic structure for efficient message organization. Additional features such as retained messages, and the Last Will and Testament mechanism

enhance reliability and facilitate the detection of unexpected disconnections. See implementation example (Code example 3).

Code example 3: Example MQTT structure for robot swarm communication

```
const topicStructure = {
  command: {
    all: '${mainTopic}/command/swarm', // Commands to all robots in a swarm
    robot: '${mainTopic}/command/robot/${robotId}', // Commands to specific robot
    experiment: '${mainTopic}/command/experiment' // Experiment control commands
  },
  response: {
    robot: '${mainTopic}/response/robot/${robotId}', // Responses from specific
robot
    swarm: '${mainTopic}/response/swarm' // Responses from swarm operations
  },
  system: {
    heartbeat: '${mainTopic}/heartbeat', // Robot connection status
    logs: '${mainTopic}/logs' // System log messages
  },
  simulator: {
    command: '${mainTopic}/simulator/command', // Commands to simulator
    experiment: '${mainTopic}/simulator/experiment' // Experiment data for
simulator
  }
};
```

3.2 On-Policy Reinforcement Learning

Reinforcement Learning (RL) is a machine learning paradigm where an agent learns to make decisions by taking actions in an environment to maximize cumulative rewards. On-policy methods specifically learn and improve a policy while using that same policy for action selection.

3.2.1 Reinforcement Learning Fundamentals

RL is typically formalized as a Markov Decision Process (MDP) with:

- **State space (S)**: All possible situations the agent can be in
- **Action space (A)**: All possible actions the agent can take
- **Reward function (R)**: Feedback signal indicating action quality
- **Transition function (P)**: Probability of reaching state s' from state s by taking action a
- **Discount factor (γ)**: Weight given to future rewards ($0 \leq \gamma \leq 1$)

The goal is to learn a policy $\pi(a|s)$ that maximizes expected cumulative reward.

3.2.2 Policy Gradient Methods

Policy gradient methods directly optimize the policy by gradient ascent on the expected

return:

$$\nabla J(\theta) = E[\nabla \log \pi_\theta(a|s) \cdot Q^\pi(s,a)]$$

Where:

- J(θ) is the expected return
- π_θ is the policy parameterized by θ
- Q^π is the action-value function under policy π

3.2.3 Proximal Policy Optimization (PPO)

PPO, used in this thesis, is an on-policy algorithm that addresses the challenge of determining the right step size when updating policies. It uses a clipped objective function:

$$L_{CLIP}(\theta) = E[\min(r_t(\theta)A_t, \text{clip}(r_t(\theta), 1-\varepsilon, 1+\varepsilon)A_t)]$$

Where:

- $r_t(\theta)$ is the ratio of new policy probability to old policy probability
- A_t the advantage estimate
- ε is a hyperparameter (typically 0.1 or 0.2)

3.2.4 Multi-Agent RL Considerations

For robot swarms, additional challenges arise:

1. **Non-Stationarity**: The environment appears non-stationary from each agent's perspective as other agents learn and change their policies
2. **Credit Assignment**: Determining which agent contributed to team success
3. **Scalability**: Computational complexity increases with swarm size

3.2.5 Implementation Challenges on ESP32-S3

Deploying RL on resource-constrained devices like ESP32-S3 presents unique challenges:

1. **Memory Limitations**: Limited RAM for storing models and experience

2. **Computational Power**: Restricted capability for forward/backward passes
3. **Energy Constraints**: Battery considerations for computation-heavy algorithms

This thesis addresses these challenges through a hybrid approach where policy training occurs in the cloud-based simulation environment, with optimized policy deployment to physical robots.

3.3 Browser-Based Simulation Environments

Browser-based simulation environments and digital twins represent complementary approaches to creating virtual representations of physical robots. This section explores how web technologies enable accessible simulation environments and how these simulations serve as digital twins that mirror physical robot behavior.

3.3.1 Web Technologies for Simulation

Modern web browsers provide powerful capabilities that make them viable platforms for robot simulation:

1. **WebGL/WebGPU**: Hardware-accelerated 3D graphics rendering
2. **Web Workers**: Multi-threaded computation in the background
3. **JavaScript Engines**: Increasingly performant code execution
4. **WebSockets**: Real-time communication for multi-user simulations
5. **Canvas API**: 2D graphics rendering for simpler simulations

These technologies enable the creation of simulation environments that are accessible from any device with a modern web browser, without requiring specialized software installation.

3.3.2 Digital Twin Concept

A digital twin creates a virtual replica of a physical entity, enabling parallel operation, testing, and analysis. A comprehensive digital twin system includes:

1. **Physical Entity**: The actual robot with sensors and actuators
2. **Virtual Model**: Software simulation of the robot's behavior
3. **Connection Layer**: Bidirectional data exchange between physical and virtual

4. **Data Storage**: Historical performance and state information
5. **Analytics Layer**: Processing of collected data for insights

In the context of this thesis, the P5.js-based simulation environment serves as the digital twin for the physical ESP32-S3 robots.

3.3.3 P5.js Implementation

This thesis utilizes P5.js, a JavaScript library for creative coding, to implement the simulation environment. The implementation encapsulates robot behavior through a modular class structure (Code example 4).

Code example 4: Simplified representation of the robot simulation implementation

```
class Robot {
  constructor(id, x, y, serialNumber) {
    this.id = id;
    this.x = x;
    this.y = y;
    this.width = 20;
    this.length = 25;
    this.height = 15;
    this.direction = random(0, 360);
    this.speed = 0;
    this.moving = false;
    this.sonicRange = 300;
    this.sonicFov = 30;
    this.path = [{x: this.x, y: this.y}];
  }

  move(directionChange, speed, duration, callback) {
    // Movement implementation with collision detection
    this.stop();
    let newDirection = (this.direction + directionChange + 360) % 360;
    let frames = duration / (1000 / 60);
    this.targetDirection = newDirection;
    this.speed = speed;
    this.moving = true;
    this.rotationSpeed = directionChange / frames;

    // Actual movement happens in an interval for smooth animation
  }

  measureDistance() {
    // Ultrasonic sensor simulation
    let minDistance = this.sonicRange;
    let sensorX = this.x + this.width/2 * cos(radians(this.direction));
    let sensorY = this.y + this.length/2 * sin(radians(this.direction));

    // Ray-casting implementation to detect obstacles
    return minDistance;
  }

  draw() {
    // Visual representation of the robot
    push();
    translate(this.x, this.y, this.height/2);
    rotateZ(radians(this.direction - 90));
    fill(255, 0, 0);
```

```
    box(this.width, this.length, this.height);
    // Additional visualization code
    pop();
  }
}
```

This implementation encapsulates key aspects of robot behavior: physical properties, movement dynamics, sensor simulation, and visual representation.

3.3.4 Physics Simulation

The simulation implements simplified but effective physics models addressing several key aspects:

1. **Kinematics**: The movement system models differential drive kinematics at 60Hz, calculating new positions based on direction and speed while handling rotational movement.
2. **Collision Detection**: A comprehensive collision system prevents robots from moving through obstacles, other robots, or environment boundaries.
3. **Sensor Models**: The simulation includes realistic sensor behaviors with field-of-view calculations, ray-casting for distance measurement, and visibility checking.
4. **Visual Sensing**: The simulation even implements a camera model that generates simulated views from each robot's perspective, enabling testing of visual recognition algorithms.

3.3.5 Digital Twin Synchronization

The simulation environment communicates with the broader control system through MQTT, acting as a digital twin for physical robots.

1. **State Synchronization**: Robot states are transmitted through standardized MQTT messages
2. **Command Reception**: The simulation responds to the same command structure used by physical robots.
3. **Environment Sharing**: Both physical and virtual robots operate in a shared conceptual space, enabling testing of swarm algorithms across mixed physical-virtual groups.
4. **Simplified Physics Model**: The simulation implements basic movement and sensor behaviors that approximate physical robot capabilities, though without specific

calibration mechanisms to match real-world behavior.

This digital twin approach enables development and testing of algorithms in simulation before deployment to physical robots, significantly accelerating the development cycle while reducing hardware costs and risks.

3.4 ESP32-S3 Architecture and Capabilities

The ESP32-S3 serves as the core processing unit for the physical robots in this research, offering several advantages for swarm robotics applications.

3.4.1 Technical Specifications

The ESP32-S3 features:

- **CPU**: Dual-core Xtensa LX7 processor up to 240 MHz
- **Memory**: 512 KB SRAM, up to 8 MB PSRAM
- **Storage**: Up to 16 MB flash storage
- **Wireless**: Wi-Fi 802.11 b/g/n and Bluetooth 5 (LE)
- **I/O**: Rich peripheral set with GPIO, ADC, DAC, I2C, SPI, UART, etc.
- **AI Acceleration**: Vector instructions for ML workloads

3.4.2 Robot Control Capabilities

For swarm robotics, the ESP32-S3 provides:

1. **Wireless Communication**: Wi-Fi connectivity for MQTT messaging
2. **Sensor Integration**: Interfaces for distance sensors, IMUs, and cameras
3. **Motor Control**: PWM outputs for motor drivers
4. **Processing Power**: Sufficient for low-level control algorithms
5. **Machine Learning**: Limited capability for on-device inference

3.4.3 Firmware Architecture

The Application layer defines high-level behavior. MQTT and the Control Loop manage messaging and real-time operations. The WiFi Stack ensures network connectivity, while Sensor Handlers interface with physical inputs. The Hardware Abstraction layer standardizes access to components, running on the ESP-IDF framework and FreeRTOS real-

time operating system.

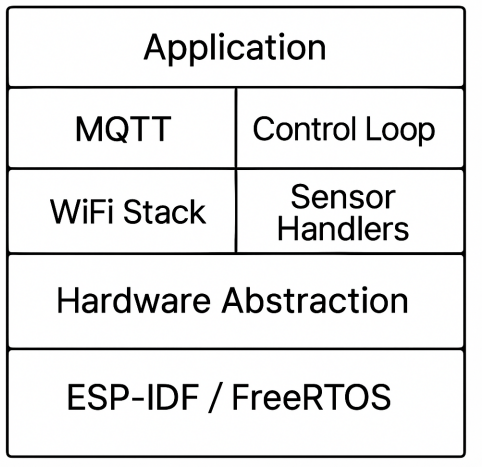

Figure 2: Layered architecture of the ESP32-based IoT control system

The firmware is built on ESP-IDF (Espressif IoT Development Framework), which provides:

1. **FreeRTOS Integration**: Real-time operating system capabilities
2. **Task Scheduling**: Concurrent handling of multiple operations
3. **Power Management**: Battery optimization features
4. **Peripheral Drivers**: Simplified hardware interfacing

3.4.4 Resource Constraints and Optimization

Working with the ESP32-S3 requires careful memory management, computational efficiency for real-time performance, and strategies to balance power consumption with functionality. Efficient use of wireless bandwidth is also essential for maintaining reliable communication.

These constraints influence design decisions throughout the system, particularly regarding the distribution of processing between the robots and the cloud.

3.4.5 Machine Learning Capabilities for Object Detection

The ESP32-S3's AI acceleration capabilities make it suitable for running lightweight machine learning models. For object detection specifically, FOMO (Faster Objects, More Objects) represents an important advancement in this space.

FOMO, or Faster Objects, More Objects, is an innovative algorithm designed for object

detection on resource-constrained devices like microcontrollers. It enables real-time detection, tracking, and counting of objects by identifying their centroids rather than using bounding boxes. This approach makes FOMO highly efficient, requiring less than 200 KB of RAM and operating up to 30 times faster than MobileNet SSD models. However, it is better suited for objects of similar size and spacing, as it does not provide information about object dimensions or handle overlapping objects effectively. FOMO's lightweight nature allows it to scale across a range of devices, from microcontrollers to GPUs, making it ideal for edge computing applications (Edge Impulse, 2025).

The ESP32-S3's vector instructions for ML workloads make it particularly suitable for running FOMO models, as they can accelerate the necessary matrix operations while staying within the constrained memory environment of embedded systems.

3.5 Digital Twin Concept in Robotics

The digital twin paradigm creates virtual replicas of physical entities, enabling parallel operation, testing, and analysis. In robotics, this concept is particularly valuable for development and deployment.

Digital Twin Components

A comprehensive digital twin system consists of the physical entity (the robot with sensors and actuators), a virtual model that simulates the robot's behavior, a connection layer for bidirectional data exchange, data storage for historical performance and state information, and an analytics layer for processing collected data.

Applications in Swarm Robotics

Digital twins accelerate development by enabling algorithm testing in simulation before physical deployment, facilitate failure prediction, support parameter optimization, provide a safe environment for reinforcement learning, and enhance visualization of swarm behavior.

Reality Gap Challenges

The reality gap arises from sensor inaccuracies, variations in actuator responses,

environmental factors such as lighting and surface friction, and simplifications in physics models due to computational limitations.

Bridging the Reality Gap

To bridge the reality gap, this thesis employs strategies such as domain randomization, hardware-in-the-loop testing, systematic calibration procedures, and the use of adaptive models that update simulation parameters based on observed discrepancies.

System Integration Approaches

Integrating multiple components into a cohesive swarm robotics system requires careful consideration of architecture, interfaces, and communication patterns.

Integration Patterns

Several integration patterns are relevant to swarm robotics:

1. **Point-to-Point**: Direct connections between components
2. **Publish-Subscribe**: Message broadcasting with topic-based filtering
3. **Request-Reply**: Synchronous communication with responses
4. **Event-Driven**: Components react to system events
5. **Layered Architecture**: Hierarchical organization of functionality

This thesis primarily employs the publish-subscribe pattern through MQTT, complemented by event-driven architecture for the web interface.

Interface Design Principles

Effective interfaces between system components follow these principles:

1. **Minimality**: Expose only necessary functionality
2. **Consistency**: Maintain uniform patterns across interfaces
3. **Robustness**: Graceful handling of unexpected inputs
4. **Versioning**: Support for evolution over time
5. **Documentation**: Clear specification of behavior and parameters

3.5.1 Full-Stack Integration

The Web Interface utilizes Vue.js, WebSockets, and MQTT over WebSocket. Cloud Services handle the MQTT broker, simulation logic, and data storage. The Physical Robot Layer includes the ESP32-S3 microcontroller, sensors, and actuators, alongside a connected simulation environment.

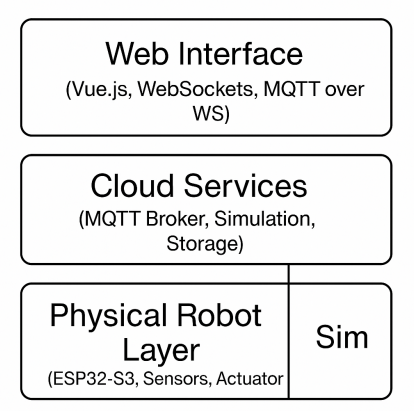

Figure 3: Layered architecture of the web-based robotic control system

The system integrates across multiple layers:

1. **Presentation Layer**: Web-based user interface
2. **Service Layer**: Cloud-based control and coordination
3. **Device Layer**: Physical and simulated robots

3.5.2 Integration Testing Approaches

Verifying correct integration requires comprehensive testing:

1. **Unit Testing**: Individual component functionality
2. **Integration Testing**: Interaction between components
3. **System Testing**: End-to-end functionality
4. **Performance Testing**: Behavior under load
5. **Chaos Testing**: Resilience to component failures

Challenges in Physical-Virtual Robot Integration

Integrating physical and virtual robots into a unified system presents unique challenges that must be addressed for effective operation.

Temporal Synchronization

Physical and virtual environments operate with different temporal characteristics:

1. **Simulation Speed**: Variable simulation rate vs. real-time physical world
2. **Communication Delays**: Network latency affecting command execution
3. **Processing Time**: Variations in computation time between virtual and physical

State Representation

Maintaining consistent state representation across both domains requires:

1. **Common Data Model**: Unified representation of robot state
2. **Transformation Functions**: Converting between physical measurements and simulation variables
3. **State Estimation**: Dealing with incomplete or noisy physical measurements

Environmental Differences

Physical environments introduce complexities not easily modeled:

1. **Sensor Noise**: Real-world sensors have noise characteristics
2. **Mechanical Variations**: Physical robots have manufacturing tolerances
3. **Environmental Factors**: Lighting, temperature, surface conditions
4. **Unexpected Obstacles**: Handling unforeseen environmental elements

Behavioral Transfer

Ensuring behaviors developed in simulation work effectively on physical robots:

1. **Sim-to-Real Transfer**: Techniques to make simulation-trained policies robust
2. **Domain Adaptation**: Adjusting policies to work in the physical domain
3. **Hybrid Learning**: Combining simulation learning with physical fine-tuning

Integration Challenges & Architectural Decisions

Several challenges (detailed in Table 4) were addressed during the physical-virtual robot integration in the Terracrawler system.

Table 4: System integration challenges and architectural solutions

Challenge	Solution	Implementation
Temporal	Event-based synchronization with timestamp reconciliation	MQTT message sequence numbers and timestamp validation
State Representation	Unified state schema with transformation layers	JSON schema for state representation with validation
Environmental	Domain randomization and reality augmentation	Variable physics parameters in simulation
Behavioral	Progressive transfer with adaptation layers	Domain-specific adaptation of control parameters

Scalability Considerations in Swarm Systems

As the number of robots in a swarm increases, scalability becomes a critical concern affecting system design and performance.

Scalability Dimensions

Swarm system scalability encompasses multiple dimensions:

1. **Numerical Scalability**: Performance with increasing robot numbers
2. **Geographical Scalability**: Operation across distributed physical spaces
3. **Administrative Scalability**: Management of heterogeneous robot groups
4. **Functional Scalability**: Adding new capabilities without redesign

Communication Scalability

MQTT provides scalability advantages through:

1. **Topic Filtering**: Robots receive only relevant messages
2. **Message Compression**: Efficient data transmission
3. **QoS Levels**: Balancing reliability and overhead
4. **Connection Pooling**: Efficient broker connections

Computational Distribution

Balancing computation across the system:

1. **Edge Computing**: Processing on individual robots
2. **Fog Computing**: Local aggregation nodes
3. **Cloud Computing**: Centralized heavy processing
4. **Hybrid Approaches**: Dynamic allocation based on requirements

Scalability Testing Methods

Evaluating system scalability through:

1. **Load Testing**: Performance under increasing robot numbers
2. **Stress Testing**: Behavior at system limits
3. **Soak Testing**: Sustained operation over extended periods
4. **Scalability Metrics**: Latency, throughput, and resource utilization

Scalability Trade-offs

The centralized approach simplifies coordination and enables global optimization but introduces higher communication overhead and a single point of failure. In contrast, the distributed approach enhances scalability and resilience through local interactions but requires more complex coordination mechanisms.

Table 5: Comparison of centralized and distributed control approaches

Aspect	Centralized Approach	Distributed Approach
Communication Overhead	High with many robots	Lower, more local communication
Coordination Complexity	Simpler implementation	More complex consensus needed
Single Point of Failure	Yes	No
Global Optimization	Easier to achieve	Harder to guarantee
Scalability Limit	Lower, bottlenecked by central node	Higher, limited by local interactions

4 Methodology

4.1 Research Design

The research design for this thesis follows a mixed-methods engineering approach, combining system development, simulation, and empirical evaluation. The study is structured around the iterative development and testing of a cloud-based robot swarm control system that integrates both physical ESP32-S3 robots and browser-based simulation environments.

The methodology consists of several key phases:

System Architecture and Implementation

A modular, cloud-based architecture was designed and implemented, featuring a unified MQTT-based communication protocol for both physical and virtual robots. The system includes a web-based user interface for monitoring, control, and experiment configuration.

Simulation Environment Development:

A browser-based simulation environment was developed to mirror the behavior of physical robots, enabling rapid prototyping and testing of swarm algorithms before deployment to hardware.

Integration and Calibration:

Physical and virtual robots were integrated within a unified control framework. Parameter calibration methods were employed to minimize the reality gap between simulation and real-world performance.

Experimental Evaluation:

A series of experiments were conducted to evaluate system scalability, communication latency, reliability, and the effectiveness of reinforcement learning algorithms in both simulated and physical environments. Performance metrics were collected and analyzed to assess the system's capabilities and limitations.

Iterative Refinement:

Findings from each phase informed subsequent development cycles, enabling continuous improvement of both hardware and software components.

A schematic overview of the system architecture and experimental setup is provided in Figure 4.

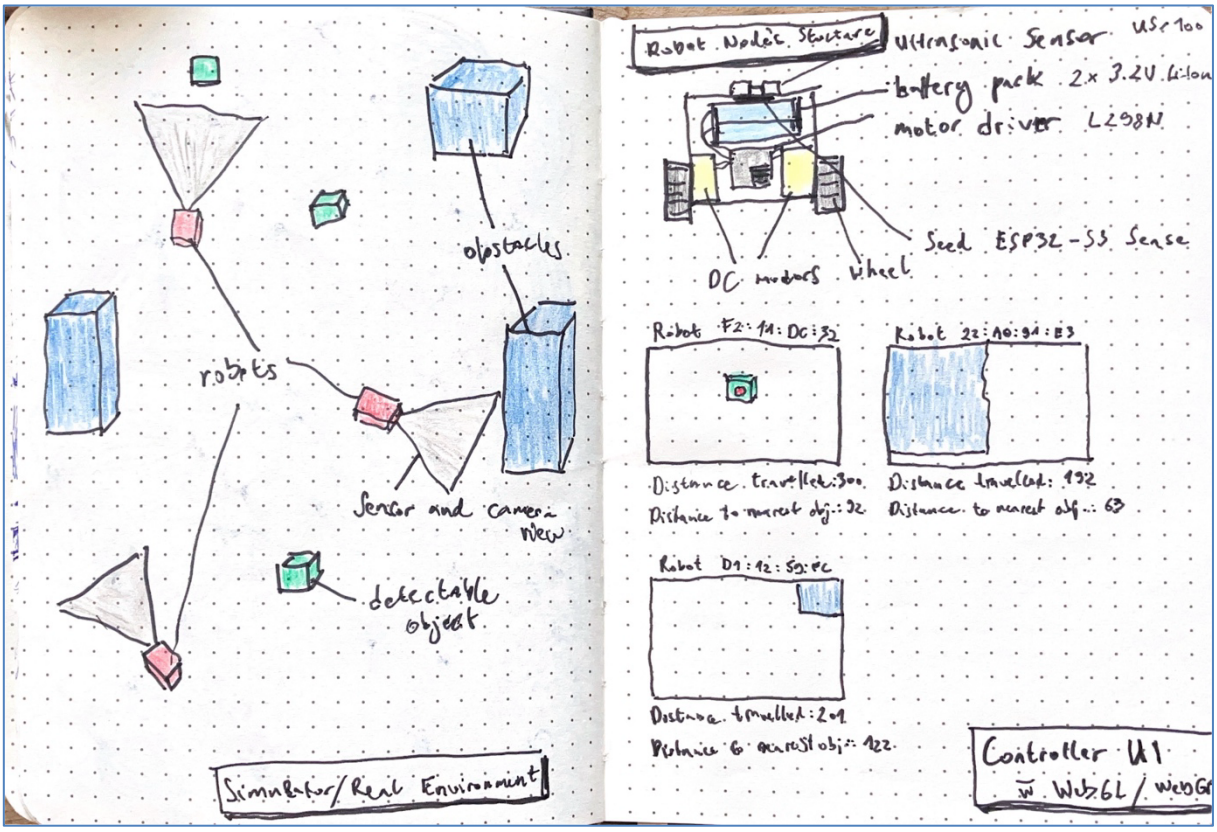

Figure 4: System Schematics

5 Development of the ESP32-S3 Robotic System

5.1 Hardware Design

Three small three-wheeled robots were constructed using the following components:

- ESP32-S3 Sense microcontroller
- L298N motor driver board module
- 2 DC motors for differential drive
- US-100 Ultrasonic Distance Sensor for obstacle detection
- Custom chassis for component mounting made of empty CD cases
- Battery power supply (LI-ION 2 * 3.7V batteries in series)

The configuration includes the ESP32-S3 microcontroller, sensors, L298N motor driver module, and battery pack mounted on a wheeled chassis. The microcontroller features dual-core processing, integrated Wi-Fi and Bluetooth, and supports FreeRTOS for real-time task management. This figure (Figure 6: Schematic diagram of the ESP32-S3 microcontroller) details the pin configuration, power supply layout, and key interfaces used in the control

system.

The L298N (Figure 7: L298N motor driver module) motor driver module provides bidirectional control for two DC motors and supports voltage and current regulation suitable for robotics applications. This configuration enables coordinated experiments and evaluation of distributed control systems. Each unit is equipped with identical hardware and ready for scenario-based validation.

Figure 5: The ESP32-S3 module used in the system

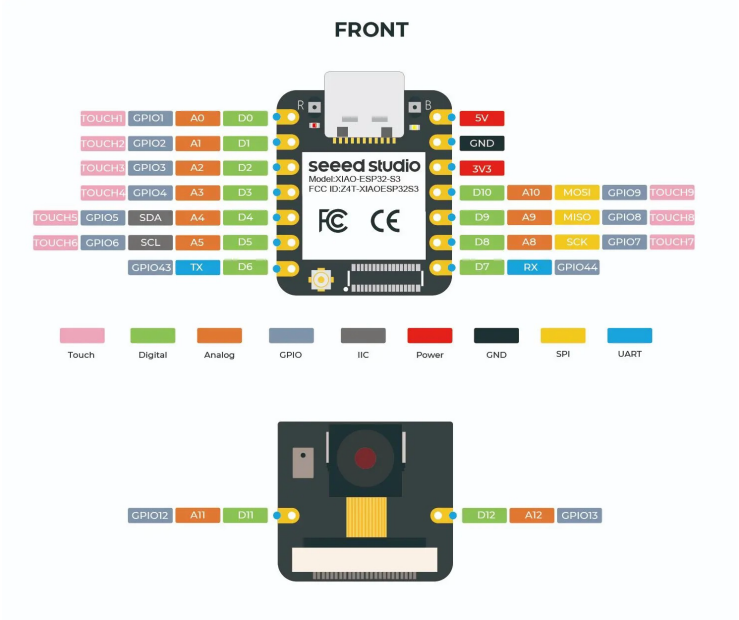

Figure 6: Schematic diagram of the ESP32-S3 microcontroller

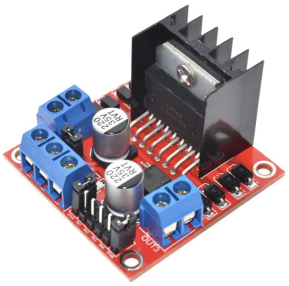

Figure 7: L298N motor driver module

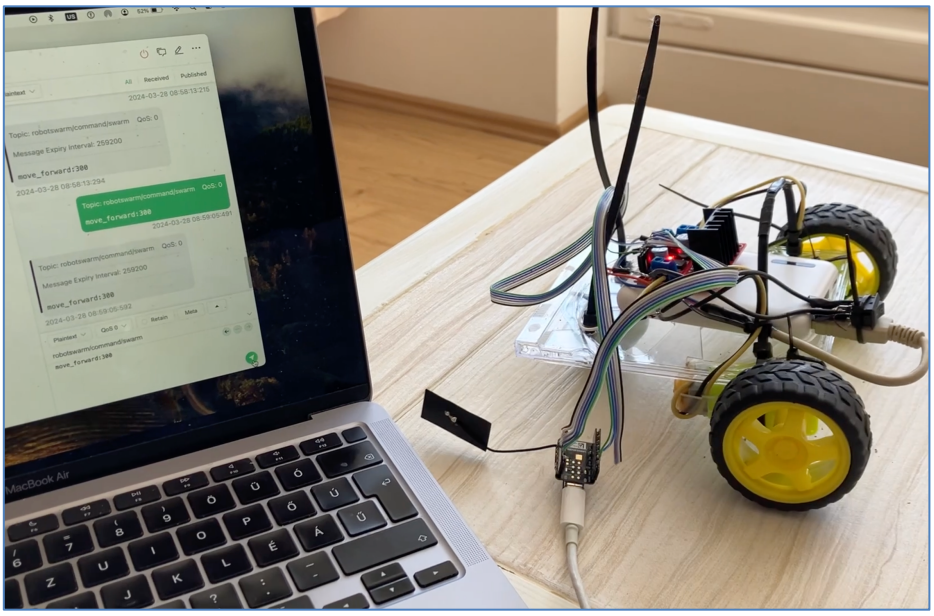

Figure 8: The assembled mobile robot during MQTT test

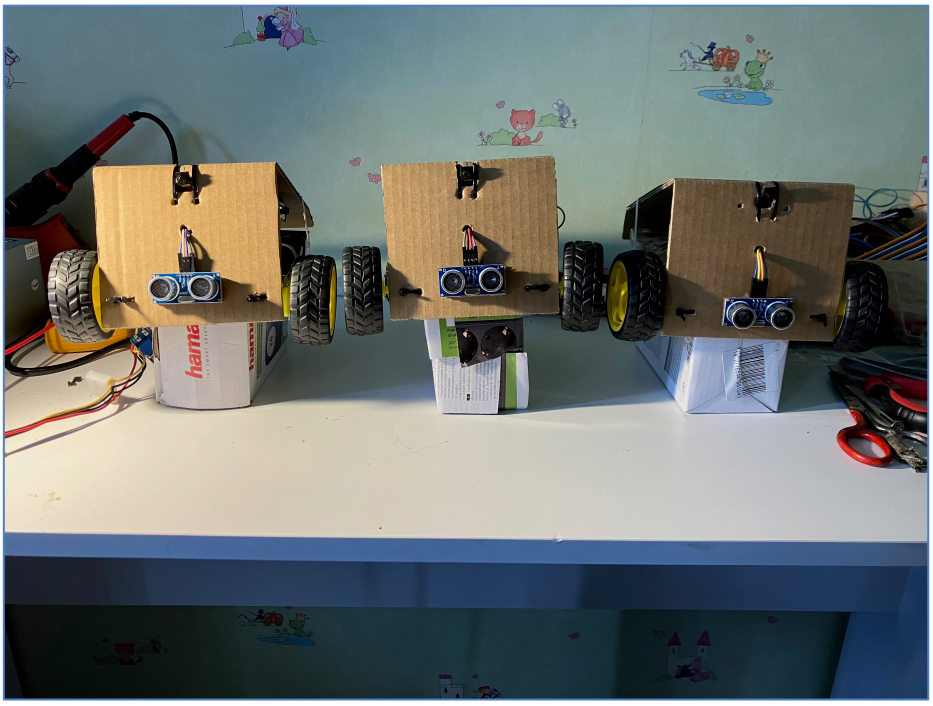

Figure 9: Multiple mobile robots set up for group testing

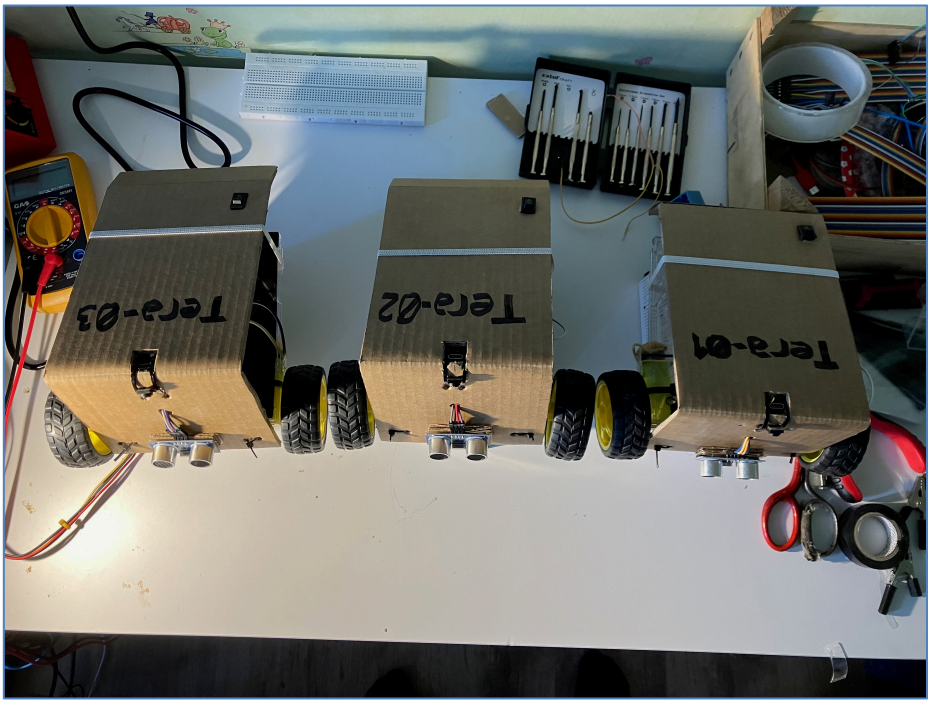

Figure 10: Close-up of three mobile robots prepared for integration testing

5.2 Software Architecture

The ESP32-S3 robotic system's software architecture employs a modular, event-driven framework that balances local control capabilities with cloud connectivity. Built on the ESP-IDF framework, the architecture follows a layered approach consisting of Application, Communication, Hardware, and System layers.

Core Components

The system is organized around several key components including the RobotControl module, WiFi Manager, MQTT Client, Sensor Integration, Visual Processing (with ESP32-Camera for object detection), and LED Control System. These components work together to provide comprehensive robot functionality while maintaining separation of concerns.

System Architecture Overview

The firmware implements a multitasking environment leveraging FreeRTOS, with separate tasks handling communication, sensor monitoring, motor control, and system management. State management is implemented through a finite state machine approach, with states including INITIALIZING, IDLE, MOVING, SCANNING, and ERROR.

Communication utilizes an MQTT-based protocol with a hierarchical topic structure, enabling efficient message routing between robots and the cloud control system. The system also incorporates power management strategies to extend battery life and comprehensive logging mechanisms for debugging and monitoring.

Note: Detailed implementation specifics including code examples, detailed component descriptions, and configuration details can be found in Appendix A: Software Architecture Details.

5.3 Cloud-Based Control System Development

The cloud-based control system serves as the central coordination hub for the robot swarm, providing a unified platform for monitoring, control, and simulation. This architecture enables scalable management of both physical and virtual robots while offering accessibility from any web-enabled device.

5.3.1 System Architecture Overview

The control system follows a layered architecture consisting of Presentation, Application, Communication, and Data Persistence layers. Each layer encapsulates specific functionalities and technologies used in the robotic control and data processing framework.

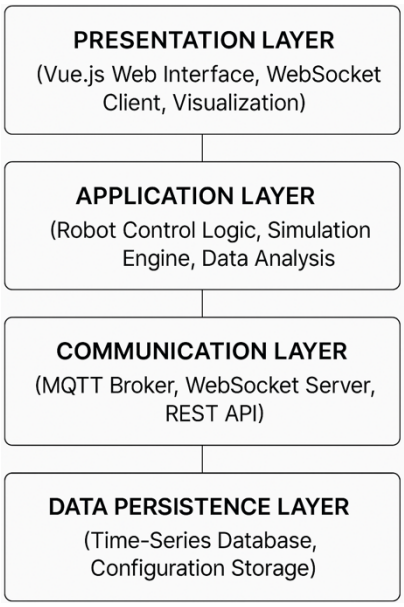

Figure 11: Layered architecture of the system

This architecture provides several key benefits:

1. Loose coupling between components enables independent development and testing
2. Horizontal scalability allows for handling increased robot numbers
3. Maintainability through clear separation of system responsibilities
4. Flexibility to extend functionality without significant refactoring

Server Components

The cloud system consists of several interconnected server components:

1. **MQTT Broker (Eclipse Mosquitto)**: Handles all publish-subscribe communication between robots and control system
2. **Node.js Application Server**: Hosts the web application and business logic
3. **WebSocket Server**: Provides real-time updates to connected clients
4. **Simulation Engine**: Runs the P5.js-based robot simulation
5. **Time-Series Database (InfluxDB)**: Stores historical telemetry data for analysis
6. **Redis Cache**: Maintains current system state and enables fast data access

These components are containerized using Docker for consistent deployment and scaling, allowing the system to be easily deployed across different environments from development to production.

Data Flow Architecture

The system implements a multi-directional data flow pattern:

1. **Command Flow**: User interface → Application server → MQTT broker → Robots
2. **Telemetry Flow**: Robots → MQTT broker → Application server → Database and UI
3. **Simulation Data**: Simulation engine → Application server → UI and Database
4. **Analytics Flow**: Database → Application server → User interface

This pattern enables responsive control while maintaining historical data for analysis. The message-based architecture allows for loose coupling between components and supports both synchronous and asynchronous communication patterns.

Security Implementation

The cloud control system implements several security measures:

1. **Authentication**: JWT-based authentication for web interface access
2. **Authorization**: Role-based access control for different system functions
3. **Communication Security**: TLS encryption for all MQTT and WebSocket traffic
4. **Input Validation**: Schema-based validation for all incoming messages
5. **Rate Limiting**: Protection against excessive requests
6. **Audit Logging**: Tracking of all system commands and configuration changes

These measures ensure that the system remains secure even when exposed to public networks, protecting both the control infrastructure and the connected robots.

Deployment Strategy

The system is designed with a flexible deployment approach that supports different operational environments:

1. **Development**: Local development environment for rapid iteration and testing
2. **Testing**: Dedicated testing setup for validating system components
3. **Production**: Standalone deployment with appropriate hardware resources

For small-scale deployments, the system can run on a single machine (even a Raspberry Pi 4), while larger deployments can utilize additional computing resources as needed.

Scalability Considerations

To accommodate varying swarm sizes, the system implements several scalability patterns:

1. **Horizontal Scaling**: Adding more service instances for increased load
2. **Message Partitioning**: Distributing MQTT topics across broker instances
3. **Data Sharding**: Partitioning database storage by time periods or robot group
4. **Resource Limits**: Configurable constraints on simulation complexity

These approaches allow the system to scale from managing a few robots to potentially hundreds, depending on available resources and communication bandwidth.

5.3.2 Web Interface Design

The web interface provides a unified control environment for both physical and simulated

robots, featuring an intuitive yet powerful user experience that balances simplicity with advanced functionality.

Interface Structure

The interface follows a responsive, component-based design organized into several key components:

1. **Robot Control Panel**: A grid-based layout showing individual robot status and controls
2. **Command Center**: Central interface for issuing swarm-level commands and monitoring system status
3. **Experiment Management**: Tools for configuring and running reinforcement learning experiments
4. **Simulation Interface**: Controls for managing the simulation environment and its integration with physical robots
5. **Telemetry Dashboard**: Real-time visualization of robot sensor data and performance metrics

These components are integrated into a unified interface through a reactive state management system, allowing seamless transitions between different control modes and operational contexts.

Dashboard Component

The dashboard provides an at-a-glance overview of the entire system, displaying:

- Connected robot count with status indicators
- Communication system health (connection status, message delivery)
- System operational mode (simulation, physical, or hybrid)
- Active experiments and their progress
- Recent events and errors via a scrollable log system

The dashboard implements a reactive design that updates in real-time as robot status changes, with visual indicators that clearly communicate the state of each connected device.

Robot Control Interface

The individual robot control interface enables direct operation of each robot through:

- Directional controls (forward, backward, turn left/right)
- Stop button for immediate motion cessation
- Image capture capability for camera-equipped robots
- Visual feedback showing sensor readings and robot status

For swarm-level operations, the interface provides specialized command modes (Code example 5).

Code example 5: Sample of command implementation function

```
function sendCommand(command) {
  if (command === "disperse") {
    isDispersing.value = true;
    isGathering.value = false;
  } else if (command === "gather") {
    isGathering.value = true;
    isDispersing.value = false;
  } else if (command === "stop") {
    isGathering.value = false;
    isDispersing.value = false;
  }

  if (command === "disperse" || command === "gather") {
    isCommandLooping.value = true;
    sendRandomCommands(command);
  } else if (command === "stop") {
    isCommandLooping.value = false;
    if (commandInterval) {
      clearInterval(commandInterval);
      commandInterval = null;
    }
    sendStopCommand();
  } else {
    robots.forEach((robot) => {
      if (robot.id) {
        sendControlCommand(robot.id, command);
      }
    });
  }
}
```

This control architecture enables both individual precision control and coordinated swarm behaviors through higher-level commands.

Keyboard Control System

To enhance usability, the interface implements a keyboard control system that allows operators to efficiently control selected robots:

- Arrow keys for directional movement

- Spacebar for emergency stop
- Tab key for cycling between robots
- Enter key for capturing images

This system greatly improves operational efficiency for experienced users, particularly when fine-tuning robot movements or conducting manual tests.

Simulation Integration

The simulation environment is seamlessly integrated with the control interface, enabling:

- Switching between simulation and physical robot modes
- Visualizing both physical and simulated robots in a unified interface
- Transferring control strategies between virtual and physical domains
- Testing algorithms in simulation before deployment to physical robots

The simulation environment communicates with the control interface through a consistent messaging protocol, ensuring command compatibility between virtual and physical robots.

Reinforcement Learning Experiment Interface

A specialized interface component manages reinforcement learning experiments:

- Experiment configuration with adjustable parameters
- Training process visualization and progress monitoring
- Performance metrics tracking (rewards, accuracy, completion rates)
- Model management for trained policies

The experiment interface leverages a dedicated RLService that handles the complexity of training and deploying reinforcement learning models (Code example 6).

Code example 6: Snippet of the RLService controller

```
async function startExperimentTraining(experiment, isRetrain = false) {
  try {
    currentExperiment.value = experiment;

    // Initialize new RLService only if needed
    if (!rlService.value) {
      rlService.value = new RLService();
      // Set up event handlers after creating the service
      setupRLEventHandlers();
```

```
    }

    // Wait for robots to connect with extended timeout
    const allRobotsConnected = await waitForRobots(
      experiment.numRobots,
      experiment.swarm_id,
      60000 // Extended timeout
    );

    if (!allRobotsConnected) {
      throw new Error("Not all robots connected within timeout period");
    }

    // Initialize RL service (must be before ROBOTS_CONNECTED)
    await rlService.value.initialize(experiment._id, {
      numRobots: experiment.numRobots,
      inputShape: 33,
      netArch: { pi: [64, 64], vf: [64, 64] },
      trainingAim: experiment.trainingAim,
      isRetrain: isRetrain,
      maxStepsPerEpisode: experiment.maxStepsPerEpisode || 500,
      trainingMethod: experiment.trainingMethod || 'dqn',
    });

    // Wait for RLService to be fully initialized before sending robots
    await waitForRLServiceReady();

    // Begin training process
    isTraining.value = true;
    return currentExperiment.value;
  } catch (error) {
    console.error("Error in startExperimentTraining:", error);
    await stopAllProcesses(); // Clean up on error
    throw error;
  }
}
```

This integration of reinforcement learning capabilities directly into the control interface enables seamless transitions between manual control, autonomous operation, and learning-based approaches.

Robot State Visualization

The interface visualizes robot state through multiple representations:

- Status indicators showing operational state (connected, disconnected, training)
- Real-time display of sensor readings (distance measurements, object detection)
- Canvas overlays for visualizing detected objects in the robot's environment
- Camera feed display for robots equipped with cameras

This multi-modal visualization approach provides operators with comprehensive awareness of robot state and environmental conditions.

Fault Tolerance and Reconnection

The interface implements robust fault tolerance mechanisms for managing disconnections and errors:

- Heartbeat monitoring to detect inactive robots
- Automatic reconnection attempts for disconnected robots
- Graceful degradation during connection issues
- Training state preservation during temporary disconnections

These mechanisms ensure system resilience during real-world operation, where network connectivity and hardware reliability may vary (Code example 7).

Code example 7: Handling robot connection and disconnection

```
function handleRobotDisconnect(robotId) {
  console.log(`Handling disconnect for robot: ${robotId}`);

  // Check if already disconnected
  if (disconnectedRobots.value.has(robotId)) {
    console.log(`Robot ${robotId} already marked as disconnected`);
    return;
  }

  const index = robots.findIndex(robot =>
    robot.robot_id === robotId ||
    robot.id === robotId ||
    robot.mac_address === robotId
  );

  if (index !== -1) {
    const robot = robots[index];
    stopRobotProcesses(robot);
  }
}

async function handleRobotReconnect(robotId) {
  // Restore robot state and resume training if all robots reconnected
  if (currentExperiment.value?.status === "Paused" &&
      disconnectedRobots.value.size === 0) {
    await resumeTraining();
  }
}
```

Communication Layer

The interface communicates with robots through an MQTT messaging system, providing:

- Command transmission to both individual robots and groups
- Telemetry reception from all connected robots
- Status monitoring and heartbeat verification

- Message queueing for offline robots

This MQTT-based approach ensures efficient, scalable communication while maintaining low latency for real-time control operations.

Responsive Design

The interface implements responsive design principles that adapt to different screen sizes and device types:

1. **Grid-Based Robot Layout**: Adjustable grid that reorganizes based on available screen space
2. **Context-Appropriate Controls**: Control density and size adjustments based on device type
3. **Prioritized Information Display**: Focus on most critical information when space is limited
4. **Touch-Optimized Interactions**: Support for both pointer and touch input methods

These responsive elements ensure the interface remains usable across various devices, from desktop workstations to tablets used in field testing.

Frontend Implementation Technologies

The web interface is built using modern web technologies:

1. **Vue.js 3**: Component-based UI framework with Composition API for reactive state management
2. **Paho MQTT Client**: JavaScript MQTT client for robot communication
3. **D3.js**: Data visualization library for performance metrics and sensor data
4. **P5.js**: Creative coding library for simulation visualization and object representation
5. **Tailwind CSS**: Utility-first CSS framework for responsive styling

The implementation follows a component-based architecture with a clear separation of concerns:

- Core components for UI rendering and user interactions
- Service modules for MQTT communication, authentication, and API requests

- State management using Vue's reactive system and local storage
- Specialized modules for reinforcement learning and robot control

Usability Considerations

The interface design incorporates several usability principles that enhance operator effectiveness. **Consistent visual language** is maintained across all interface components, ensuring a coherent and intuitive user experience. **Contextual feedback** provides immediate visual confirmation of actions and system state, while **error prevention** is achieved by disabling controls during inappropriate operational states. **Progressive disclosure** reveals advanced options only when relevant to current tasks, reducing cognitive load for novice users. For experienced users, **keyboard shortcuts** support efficient operation, and **status persistence** maintains awareness of robot state across sessions and disconnections.

These principles ensure the interface remains accessible to novice users while providing the depth needed for research and development applications, accommodating a wide range of operational scenarios from educational use to advanced swarm robotics research.

5.3.3 MQTT Infrastructure Implementation

The MQTT infrastructure forms the communication backbone of the robot swarm control system, enabling efficient, reliable message exchange between system components.

Broker Configuration

The MQTT broker (Eclipse Mosquitto) is configured for optimal performance in robot swarm applications:

- **QoS Levels**: Configurable Quality of Service for different message types
- **Retained Messages**: Last-known state persistence for late-joining clients
- **Message Persistence**: Disk-based storage for critical messages
- **Authentication**: Username/password and certificate-based authentication

Topic Structure

The MQTT topic hierarchy follows a structured pattern to facilitate efficient message routing (Code example 8).

Code example 8: MQTT topic structure for robot swarm communication

```
const topicStructure = {
  command: {
    all: '${mainTopic}/command/swarm', // Commands to all robots in a swarm
    robot: '${mainTopic}/command/robot/${robotId}', // Commands to specific robot
    experiment: '${mainTopic}/command/experiment' // Experiment control commands
  },
  response: {
    robot: '${mainTopic}/response/robot/${robotId}', // Responses from specific
robot
    swarm: '${mainTopic}/response/swarm' // Responses from swarm operations
  },
  system: {
    heartbeat: '${mainTopic}/heartbeat', // Robot connection status
    logs: '${mainTopic}/logs' // System log messages
  },
  simulator: {
    command: '${mainTopic}/simulator/command', // Commands to simulator
    experiment: '${mainTopic}/simulator/experiment' // Experiment data for
simulator
  }
};
```

Message Format

Messages use a standardized JSON format for compatibility and flexibility, containing:

- Command or data type identifier
- Parameters specific to the message type
- Timestamps for synchronization and message ordering
- Optional metadata for tracking and debugging

Bridge Configuration

For distributed deployments, MQTT bridge configurations enable message passing between multiple brokers, supporting:

- Geographic distribution of robot swarms
- Hierarchical control structures
- Redundancy for high-availability deployments

Performance Optimization

The MQTT infrastructure includes several performance optimizations:

- Message batching for efficient transmission
- Compression for bandwidth-constrained environments
- Connection pooling to reduce overhead
- Keep-alive management for stable connections

These optimizations ensure the communication system remains responsive even with large numbers of connected robots or limited network resources.

5.3.4 Simulation Environment Implementation

The simulation environment provides a browser-based virtual representation of the robot swarm, enabling development and testing without physical hardware. This environment serves as a critical tool for algorithm development, testing, and visualization.

The simulator is built on P5.js, a JavaScript library for creative coding that provides efficient 2D rendering capabilities. This implementation follows a modular architecture with several key components of which the diagram (Figure 12) illustrates the major components including the Simulation Core (handling physics and timing), Robot Objects and Environment (representing dynamic and static entities), the Rendering System for visualization, and the Communication Interface for external integration via MQTT and WebSocket.

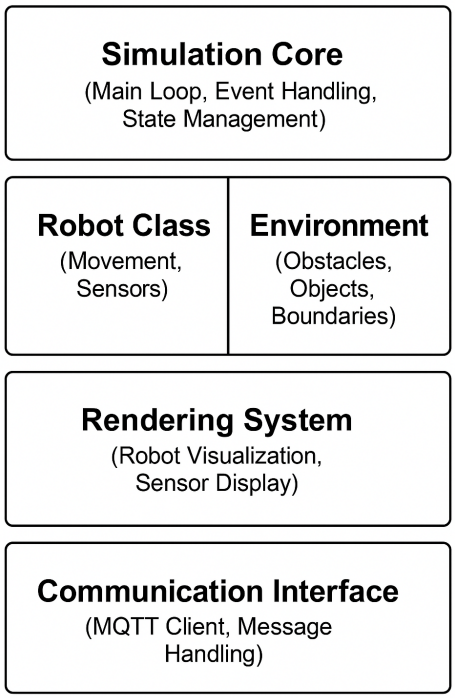

Figure 12: Layered architecture of the robotics simulation system

The simulation core manages the P5.js setup and draw loops, handling user input, maintaining simulation state, and coordinating updates between components.

Physics Model Implementation

The simulation implements a simplified 2D physics model focused on the kinematics of differential drive robots. Each robot maintains position, orientation, speed values, collision detection capabilities, and path history for movement analysis.

The movement system simulates robot kinematics at 60Hz, providing smooth animation while maintaining computational efficiency. Collision detection prevents robots from moving through obstacles, other robots, or boundary walls.

Sensor Simulation

The simulator implements detailed sensor simulation, focusing on the ultrasonic distance sensors used in the physical robots. This implementation includes field-of-view calculations, ray-casting for distance measurement, object detection within sensor range, and visibility checking that accounts for obstacle occlusion.

The sensor implementation uses geometric approaches to determine if objects are within the sensor's field of view, providing realistic sensor behavior that closely approximates the characteristics of physical ultrasonic sensors.

Visual Representation

The simulator provides comprehensive visual feedback through robot rendering with orientation indicators, path visualization showing historical movement, sensor cone visualization displaying detection zones, object and obstacle rendering, and status information display.

The visualization uses P5.js 3D rendering capabilities to create an intuitive representation of the robots and their environment, helping operators understand robot behavior, sensor capabilities, and interaction patterns within the swarm.

Advanced Sensing Capabilities

The simulator implements enhanced sensing through a camera simulation that enables object detection. This functionality creates a simulated camera view from each robot's perspective, identifying and classifying objects within the field of view. The implementation generates bounding boxes for detected objects, mimicking computer vision capabilities that can be deployed on physical robots.

MQTT Integration

The simulation environment integrates with the broader control system through MQTT communication, enabling command reception and execution in the simulation environment, status and sensor data reporting to the control system, and experiment configuration and control through MQTT topics.

A simplified version of the MQTT topic structure used (Code example 9).

Code example 9: MQTT topic structure (simplified)

```
{
  command: {
    swarm: "robotswarm-sim/command/swarm/${swarmId}",
    robot: "robotswarm-sim/command/robot/${robotId}"
  },
  response: {
    robot: "robotswarm-sim/response/robot/${robotId}"
  },
  system: {
    heartbeat: "robotswarm-sim/heartbeat"
  }
}
```

This integration ensures that the simulation environment can seamlessly participate in the broader robot swarm control system, enabling integrated operation and testing.

Dynamic Environment Generation

The simulation creates varied testing environments through procedural generation, placing robots, obstacles, and objects in configurations that avoid overlap and provide diverse testing scenarios. This approach enables random yet reproducible environment generation, controlled difficulty progression for testing, diverse scenarios for reinforcement learning training, and validation of algorithm robustness across different environments.

Combined with experiment-based configuration, this dynamic environment generation supports systematic testing and development of robot control algorithms, accelerating the development cycle while providing a foundation for reinforcement learning and algorithm validation.

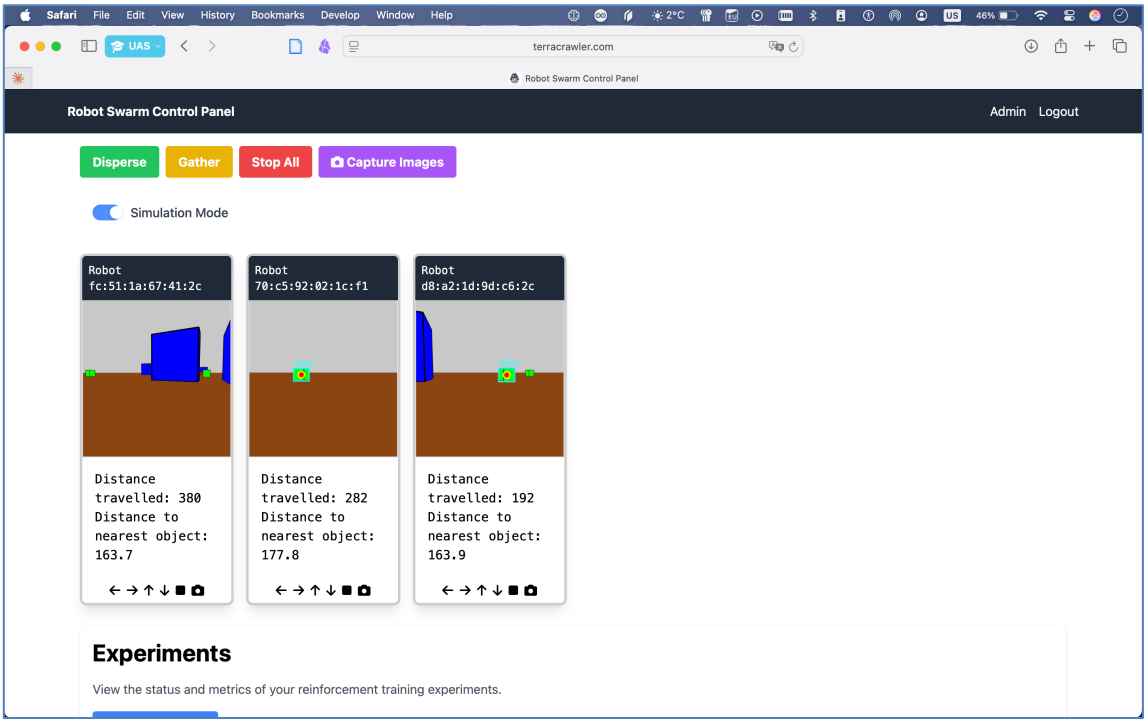

Figure 13: Screenshot from the live testing interface. Real-time telemetry data, robot status, and control inputs are visualized to monitor the system during end-to-end evaluation

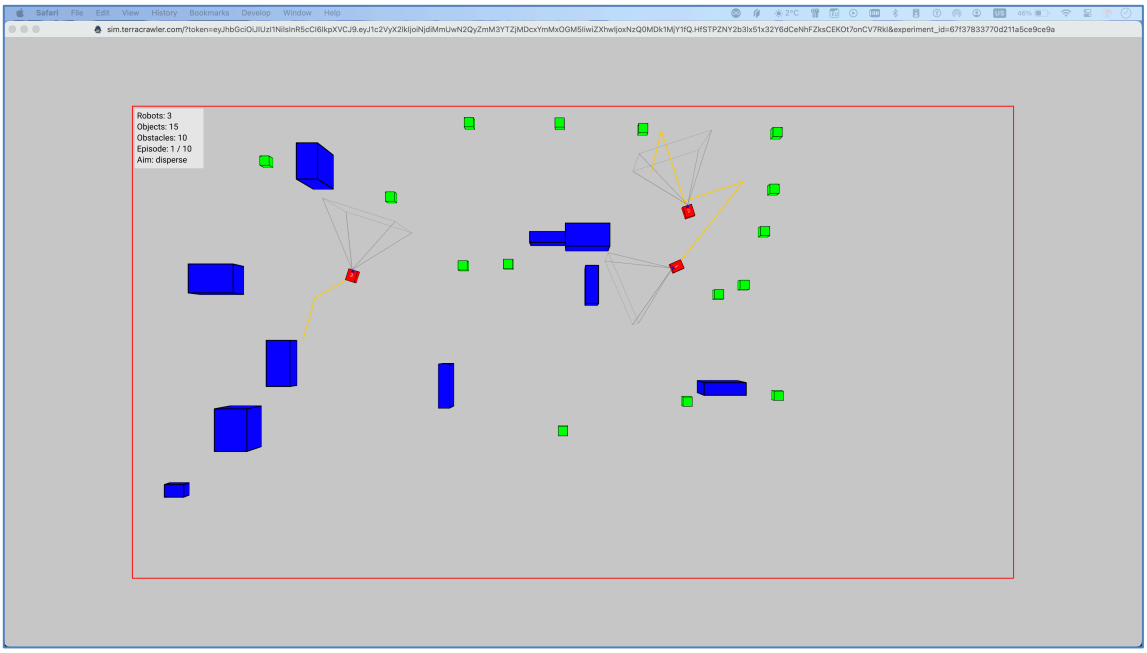

Figure 14: Live display of control loop outputs and sensor feedback during testing. The interface supports visual diagnostics for validating system behavior

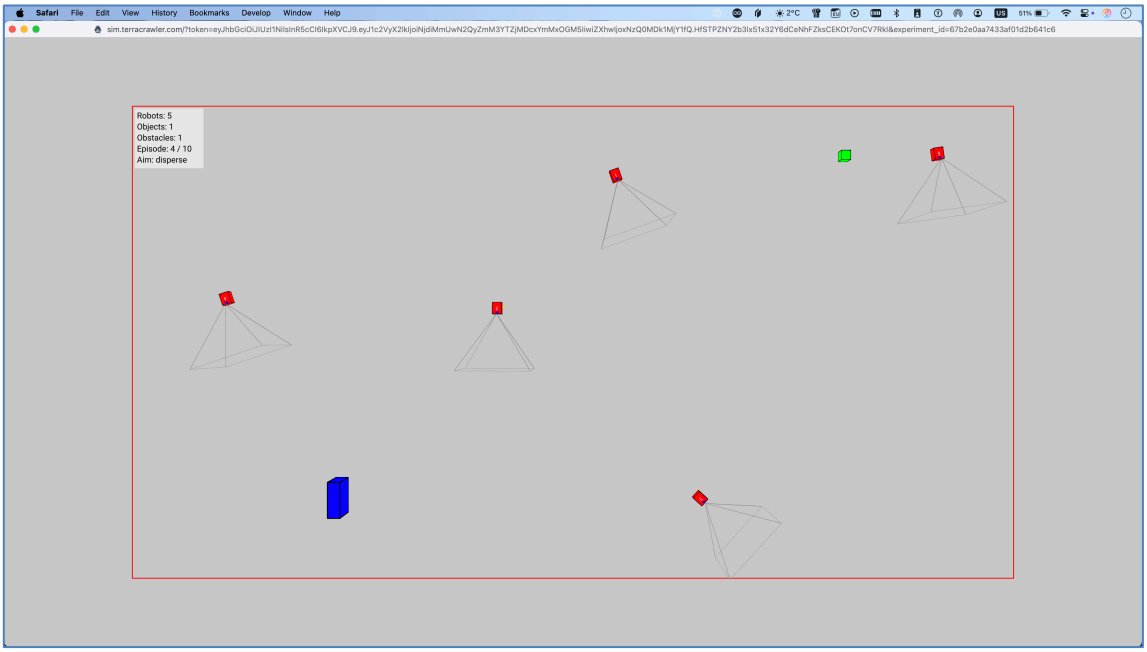

Figure 15: Interface view of the reinforcement learning training process. It visualizes training episodes, reward signals, and policy evolution over time

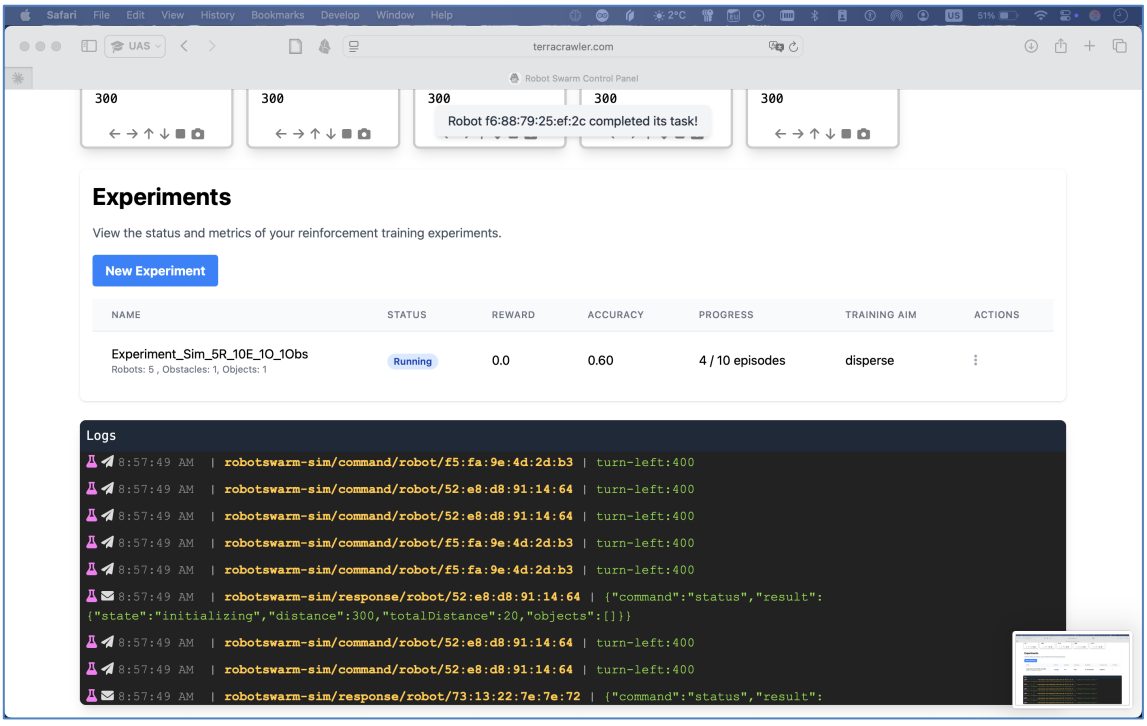

Figure 16: Screenshot from the training dashboard. It displays neural network performance metrics and simulation status during policy optimization

Integration of Physical and Virtual Environments

The integration of physical and virtual robots represents a core innovation of this system, creating a unified operational environment that bridges the gap between simulation and

reality. This integration follows a digital twin approach where virtual robots mirror the behavior of their physical counterparts and vice versa.

Unified Communication Architecture

The integration is built on a unified communication architecture centered around the MQTT protocol. In the architecture (Figure 17: MQTT-based system architecture) the MQTT Broker acts as the central message routing infrastructure, facilitating communication between the Physical Robot Layer, the Control System Layer, and the Simulation Layer. This design enables modular interaction and real-time data exchange across all components of the robotic control system.

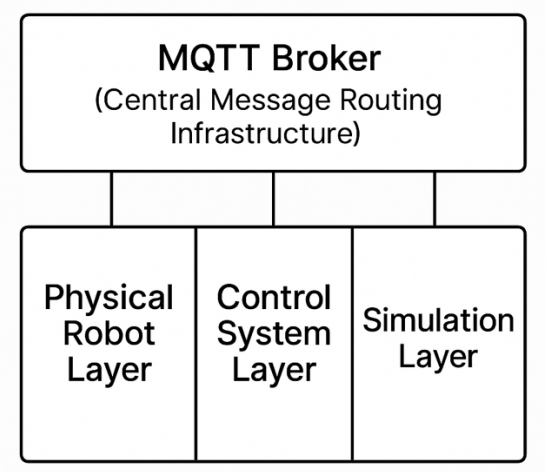

Figure 17: MQTT-based system architecture

All system components communicate through the same MQTT topic structure, with messages flowing bidirectionally between physical robots, the control system, and the simulation environment. This approach ensures that commands and telemetry are handled identically regardless of whether a robot is physical or virtual.

State Synchronization Mechanism

Maintaining synchronized state between physical and virtual environments requires a robust mechanism that addresses:

1. **Temporal Alignment**: Reconciling different time domains between real-time and simulation
2. **State Translation**: Converting between physical sensor readings and simulation

variables

3. **Command Normalization**: Ensuring commands are interpreted consistently across environments
4. **Error Recovery**: Handling disconnections and communication failures

The synchronization system implements a message-based approach with timestamped state updates and command acknowledgments. Physical robot positions are tracked through a combination of dead reckoning and periodic corrections based on external references when available.

Parameter Calibration Process

To ensure that virtual robot behavior accurately reflects physical robot characteristics, the system implements a parameter calibration process:

1. **Baseline Measurement**: Collecting performance data from physical robots
2. **Parameter Identification**: Determining key parameters (motor response, friction, sensor characteristics)
3. **Simulation Adjustment**: Updating simulation parameters to match physical behavior
4. **Validation Testing**: Comparing simulated and physical behavior in standardized tests
5. **Continuous Refinement**: Periodically updating parameters based on observed discrepancies

This process minimizes the "reality gap" between simulation and physical robots, enabling more reliable transfer of algorithms and behaviors between environments.

5.3.5 Digital Twin Implementation

The digital twin concept is implemented through bidirectional mirroring of robot states:

1. **Physical to Virtual**: Physical robot telemetry updates virtual robot states
2. **Virtual to Physical**: Simulation results inform physical robot control decisions
3. **Hybrid Operation**: Combined physical-virtual swarms operating in coordinated fashion
4. **Cross-Reality Interaction**: Virtual robots responding to physical events and vice

versa

This digital twin approach enables several advanced capabilities:

- Fault detection through comparison of expected and actual behavior
- Predictive analysis of potential control strategies in simulation
- Enhanced visualization of physical robot internal states
- Augmented sensing through combining physical and virtual sensor data

5.3.6 Reality Gap Mitigation Strategies

Despite careful calibration, differences between simulation and reality remain inevitable. The system implements several strategies to mitigate these discrepancies:

1. **Domain Randomization**: Training with varied simulation parameters to improve robustness
2. **Adaptive Control**: Adjusting control parameters based on observed performance differences
3. **Reality-Augmented Simulation**: Incorporating real-world sensor data into the simulation
4. **Progressive Transfer**: Gradually transitioning from simulation to physical implementation

These strategies help ensure that algorithms developed in simulation perform effectively when transferred to physical robots, bridging the reality gap that has traditionally limited simulation-based development.

5.3.7 Reinforcement Learning Implementation

The system implements reinforcement learning (RL) to enable autonomous navigation and obstacle avoidance capabilities for the robot swarm. This approach allows robots to learn optimal behaviors through interaction with their environment rather than through explicit programming.

Learning Approach Selection

After evaluating several reinforcement learning algorithms, Proximal Policy Optimization (PPO) was selected as the primary approach for this system due to its:

- **Sample Efficiency**: Requires fewer environment interactions than many alternatives
- **Stability**: More stable training compared to other policy gradient methods
- **Simplicity**: Relatively straightforward implementation
- **Performance**: State-of-the-art results across many problem domains
- **Resource Efficiency**: Manageable computational requirements

PPO provides a good balance between learning efficiency and implementation complexity, making it well-suited for the resource constraints of the robot swarm system.

Problem Formulation

The navigation task is formulated as a reinforcement learning problem with:

- **State Space**: Robot sensor readings, position, orientation, and goal information
- **Action Space**: Control commands for the differential drive system
- **Reward Function**: Positive rewards for approaching goals and efficient movement, negative rewards for collisions and excessive energy use
- **Discount Factor**: Balancing immediate and future rewards
- **Episode Termination**: Episodes end upon reaching goals, collisions, or timeout

This formulation encourages robots to develop behaviors that efficiently navigate to goals while avoiding obstacles and conserving energy.

Training Methodology

The training process follows a curriculum learning approach, progressively increasing the complexity of the environment:

1. **Simple Navigation**: Learning basic movement in obstacle-free environments
2. **Static Obstacle Avoidance**: Navigating environments with stationary obstacles
3. **Dynamic Obstacle Avoidance**: Avoiding moving obstacles including other robots
4. **Multi-Goal Navigation**: Navigating efficiently between multiple goals
5. **Swarm Coordination**: Learning to coordinate movement with other robots

Each stage builds upon skills learned in previous stages, creating a structured progression that accelerates learning and improves generalization.

5.4 Implementation Architecture

The reinforcement learning implementation follows a hybrid architecture.

5.4.1 Reinforcement Learning Pipeline

The Training System operates in a cloud-based simulation environment where virtual robots are used to train the model. The Policy Optimization and Execution occurs entirely in the cloud controller using the Proximal Policy Optimization (PPO) algorithm. The ESP32-S3 devices only handle object detection (using FOMO), sensor data collection, and command execution, but do not run the RL models themselves. All decision-making happens in the cloud, with commands sent to the robots via MQTT.

Table 6: Reinforcement learning system architecture.

Cloud Processing		**On-Device Execution**
2. Training System Cloud servers with simulation environments Multiple virtual robots running in parallel simulations	←	**1. Sensor Processing** FOMO object detection on ESP32-S3 Sensor data collection and preprocessing
3. Policy Optimization & Execution PPO (Proximal Policy Optimization) Cloud controller running inference with trained model Real-time decision making based on sensor data	→	**4. Command Execution** Receiving control commands via MQTT Executing motor control instructions Sending new sensor data to cloud

Training occurs primarily in the cloud-based simulation environment, leveraging its computational resources and parallelization capabilities. The trained policies are then optimized and deployed to physical robots for execution.

5.4.2 Neural Network Architecture

The policy and value functions are represented by neural networks with:

- Input layer receiving normalized sensor readings and state information
- Hidden layers with ReLU activations
- Policy output layer producing action parameters
- Value output layer estimating expected rewards

These models operate entirely within the cloud-based controller, receiving sensor data from the robots and sending back action commands. The ESP32-S3 devices primarily

handle:

1. **Object Detection**: Running lightweight FOMO (Faster Objects, More Objects) models for centroid-based object recognition
2. **Sensor Data Collection**: Gathering ultrasonic distance readings and other metrics
3. **Data Transmission**: Sending all collected data to the controller via MQTT
4. **Command Execution**: Receiving and executing movement commands from the controller

For the FOMO object detection capability on ESP32-S3 devices, several optimizations are employed:

- **Int8 Quantization**: Reducing precision to 8-bit integers
- **Layer Fusion**: Combining consecutive operations
- **Pruning**: Removing redundant connections
- **Knowledge Distillation**: Training smaller networks to mimic larger ones

These optimizations enable real-time object detection on the ESP32-S3 while the more compute-intensive reinforcement learning models remain on the cloud controller, which makes decisions based on the received sensor data and sends control commands back to the robots via MQTT.

5.4.3 Transfer Learning Approach

To bridge the reality gap between simulation and physical robots, the system implements a transfer learning approach:

1. **Pre-training**: Initial policy learning in the cloud-based simulation environment
2. **Domain Adaptation**: Fine-tuning the controller's models with reality-augmented simulation
3. **Operational Testing**: Limited real-world refinement with physical robots sending actual sensor data
4. **Continuous Improvement**: Ongoing policy updates in the controller based on operational data

This approach maintains the computationally intensive learning processes in the cloud while the physical robots focus on sensing, communication, and action execution, creating

an efficient division of responsibilities that leverages the strengths of both platforms.

5.5 Testing and Evaluation Methods

A comprehensive testing and evaluation methodology was developed to assess the performance, reliability, and scalability of the robot swarm system across both simulated and physical environments.

5.5.1 Testing Framework

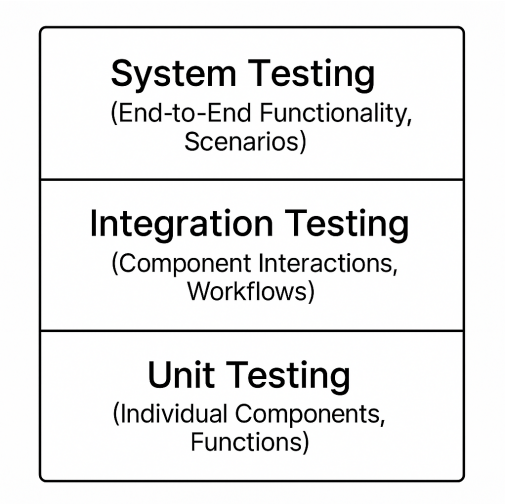

Figure 18: Software testing hierarchy

The testing framework follows a hierarchical approach, with tests at multiple levels: Unit Testing focuses on verifying individual components and functions. Integration Testing ensures correct interaction between components and workflows. System Testing validates end-to-end functionality across various scenarios, confirming the overall system performance.

This approach ensures comprehensive coverage while allowing efficient isolation and resolution of issues at the appropriate level.

Performance Metrics

The system's performance was evaluated using the following metrics:

1. **Navigation Efficiency**:

- Time to goal completion
- Path optimality (ratio of actual to optimal path length)
- Energy efficiency (distance traveled per episode)

2. **Coordination Performance**:
   - Task allocation efficiency (task completion time, resource utilization)
   - Communication overhead (messages per task, bandwidth utilization)

3. **Obstacle Avoidance**:
   - Collision frequency
   - Minimum obstacle distance maintained
   - Recovery time from potential collision situations

4. **System Reliability**:
   - Uptime percentage
   - Command execution success rate
   - Recovery from failure conditions
   - Connection stability

These metrics provide a comprehensive view of system performance across different aspects of operation.

5.5.2 Testing Scenarios

The evaluation included structured testing scenarios designed to assess specific aspects of system functionality:

1. **Simple Navigation**: Basic movement between points in an open environment
2. **Dynamic Environment**: Navigation with moving obstacles (other robots) and changing conditions
3. **Multi-Robot Coordination**: Coordinated movement in collaborative tasks
4. **Resilience Testing**: System response to simulated failures and communication disruptions
5. **Long-Duration Operation**: Extended operation to evaluate stability and resource utilization

At this stage, all scenarios have been executed in the simulation environment. While physical robots have been constructed, full integration testing with the cloud control system remains as future work. Once integration is complete, these scenarios will be repeated with physical robots to compare results and assess the reality gap and transfer performance.

5.5.3 Simulation Testing

Simulation testing enabled rapid, reproducible evaluation of system behavior across a wide range of conditions. This testing included:

1. **Parameter Sweeps**: Systematically varying parameters to identify optimal configurations
2. **Randomized Testing**: Using procedurally generated environments to test generalization
3. **Stress Testing**: Evaluating performance under high-load conditions
4. **Monte Carlo Analysis**: Statistical evaluation across multiple randomized trials (planned)
5. **Adversarial Testing**: Deliberately challenging scenarios to identify limitations

Currently, these scenarios have been executed in the simulation environment. Full integration and comparative testing with physical robots and the cloud control system are planned as future work.

5.5.4 Physical Robot Testing

While physical robots have been constructed, comprehensive integration testing with the cloud control system remains as future work. Planned physical testing will include:

1. **Controlled Environment Testing**: Structured tests in laboratory settings
2. **Semi-Structured Environment Testing**: Tests in office-like environments with furniture and obstacles
3. **Unstructured Environment Testing**: Tests in varied real-world environments

These tests will be essential for validating the transfer of capabilities from simulation to physical robots and for identifying any discrepancies that require further refinement.

6 Results and Analysis

The implementation and testing of the cloud-based robot swarm control system yielded significant insights into the performance, capabilities, and limitations of the approach. This section presents the key results and analyses their implications.

6.1 System Performance Evaluation

6.1.1 Simulation Model Validation

While physical robots have been constructed, full integration testing with the cloud control system remains as future work. The simulation environment was developed based on the physical robot specifications and expected behavior. The simulation includes linear and rotational movement based on differential drive mechanics, ray-casting for obstacle detection, and basic collision handling. A 3D visual representation enhances clarity by illustrating robot behavior and environmental interaction.

Table 7. Key characteristics of the simulation model

Aspect	Simulation Model Characteristics	Notes
Linear Movement	Based on simplified differential drive physics	Implements velocity and direction-based movement
Rotational Movement	Implements basic turning mechanics	Uses rotation speed parameters for turning behavior
Obstacle Detection	Ray-casting with FOV calculations	Models ultrasonic sensor detection principles
Collision Detection	Boundary and object intersection checks	Prevents robots from moving through obstacles
Visual Representation	3D-based rendering of robots and environment	Provides intuitive visualization of robot state

The simulator implements movement using frame-based updates at 60Hz (Code example 10).

Code example 10: Controlling movement of the robots in the simulator

```
move(directionChange, speed, duration, callback) {
  this.stop();
  let newDirection = (this.direction + directionChange + 360) % 360;
  let frames = duration / (1000 / 60);
  this.targetDirection = newDirection;
  this.speed = speed;
  this.moving = true;
  this.rotationSpeed = directionChange / frames;
  this.currentMovementInterval = setInterval(() => {
    let newX = this.x + this.speed * s.cos(s.radians(this.direction));
    let newY = this.y + this.speed * s.sin(s.radians(this.direction));
    if (!this.collidesWith(newX, newY)) {
```

```
      this.x = newX;
      this.y = newY;
      this.path.push({
        x: this.x,
        y: this.y
      });
    } else {
      this.stop();
      if (typeof callback === 'function') {
        callback();
      }
    }
  }, 1000 / 60);
}
```

The simulation provides a foundation for future testing with physical robots, though it employs simplified physics models rather than comprehensive parameter-based systems. While the simulation lacks advanced features like variable surface friction or detailed energy modeling, it captures the core movement and sensing behaviors needed for algorithm development and testing.

6.2 Reinforcement Learning Results

6.2.1 Training Performance

Training Performance Report: Simulation Experiments with 3 Robots, 50 Episodes, 5 Objects, 4 Obstacles

Overview

Two simulation experiments were conducted to evaluate the training performance of a robot swarm (3 robots, 50 episodes, 5 objects, 4 obstacles, max 500 steps/episode) using two reinforcement learning methods: DQN and PPO. The training aim was "disperse" in both cases.

Experiment 1: DQN

Final Mean Reward: -11.66

Final Success Rate: 31.3%

Best Episode: 48

Convergence Epoch: 20

Mean Episode Length: 370.6 steps

Status: Completed

Learning Curve & Performance:

The DQN agent started with strongly negative rewards (Ep 1: -94.21), indicating poor initial performance. Rewards fluctuated significantly in early episodes, with some positive outliers (e.g., Ep 5: 15.92). After convergence (around episode 20), the agent showed more frequent positive rewards, with the best episode (Ep 48) achieving a high reward (110.14). Success rate plateaued at 31.3%, suggesting that about one-third of episodes ended with successful dispersal. Episode lengths were variable, with some episodes reaching the maximum step limit, indicating occasional difficulty in task completion.

Experiment 2: PPO

Final Mean Reward: 4.16

Final Success Rate: 31.3%

Best Episode: 26

Convergence Epoch: 20

Mean Episode Length: 365.6 steps

Status: Completed

Learning Curve & Performance:

PPO also began with negative rewards (Ep 1: -113.56) but showed a steadier improvement. Positive rewards appeared earlier (Ep 6: 5.41), and the best episode (Ep 26: 81.64) occurred sooner than in DQN. After convergence, PPO maintained more consistent positive rewards, with several episodes above 50. The final mean reward was positive (4.16), outperforming DQN in overall average reward. Success rate matched DQN at 31.3%, but PPO achieved higher peak rewards and more stable performance in later episodes.

Episode lengths were like DQN, with some episodes reaching the maximum step limit.

Table 7. Comparative Analysis

Metric	DQN	PPO
Final Mean Reward	-11.66	4.16
Final Success Rate	31.3%	31.3%
Best Episode Reward	110.14	81.64
Convergence Epoch	20	20

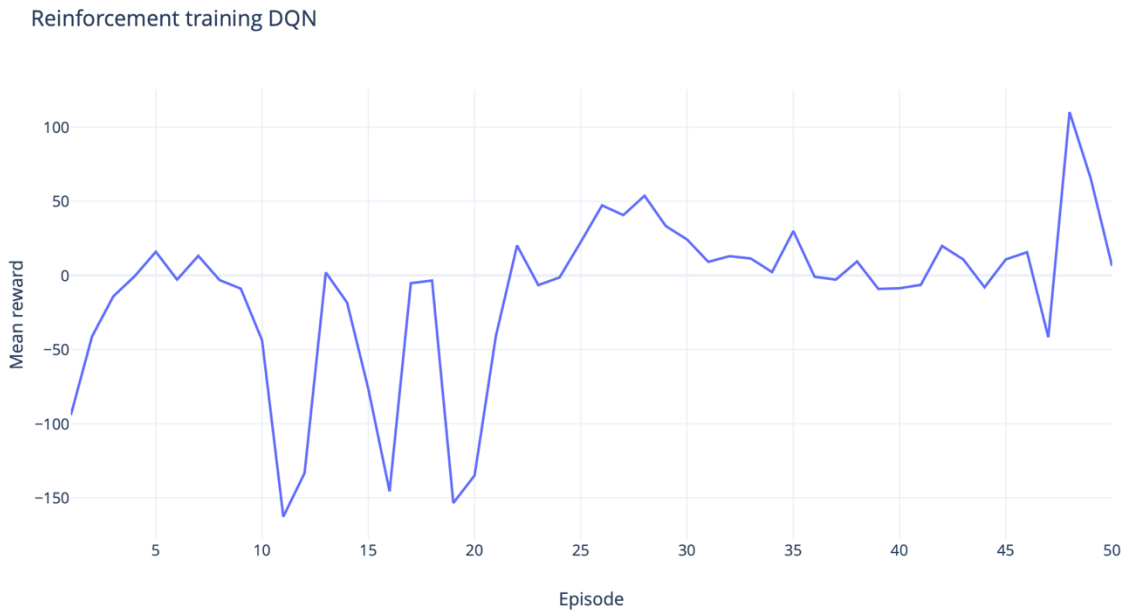

Figure 19: DQN training results

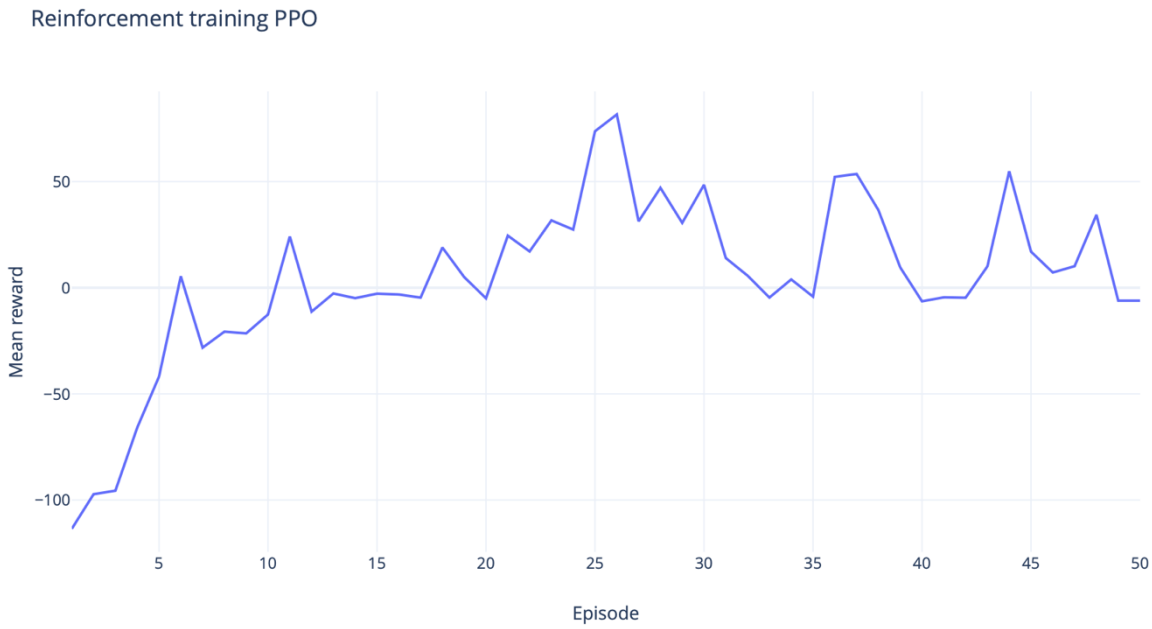

Figure 20: PPO training results

PPO achieved a positive final mean reward, indicating better average performance and learning stability compared to DQN. Both methods reached the same final success rate, suggesting similar effectiveness in achieving the dispersal objective. PPO demonstrated more consistent improvement and less volatility in rewards after convergence. DQN achieved a higher single-episode reward, but PPO's rewards were more consistently positive in the latter half of training.

Conclusions

PPO outperformed DQN in terms of final mean reward and reward stability, though both methods achieved the same success rate. Both algorithms required about 20 episodes to converge, after which performance stabilized. The results indicate that, for this scenario, PPO provides more reliable learning and better average outcomes, while DQN can occasionally achieve higher peak rewards but with greater variability. Further tuning and longer training may be needed to increase the overall success rate beyond 31.3%.

6.2.2 Preparation for Physical Transfer

While the system has been designed to transfer learned policies from simulation to physical robots, this transfer remains as future work. The simulation environment incorporates several features to facilitate this future transfer:

Table 9. Features for physical transfer

Feature	Implementation	Expected Benefit
Domain Randomization	Variable friction, sensor noise	Should improve policy robustness
Calibration Framework	Parameter adjustment interface	Will allow matching to physical behavior
Reality Gap Modeling	Conservative physics parameters	Designed to ensure safe transfer

These features have been incorporated based on best practices in sim-to-real transfer, with the expectation that they will facilitate effective policy transfer when physical testing begins.

6.2.3 Scalability Analysis

The scalability of the cloud-based robot swarm control system was evaluated by progressively increasing the number of robots in the simulated environment. Key aspects analyzed include communication overhead, coordination effectiveness, and user interface responsiveness.

As the number of robots increased, MQTT message traffic grew linearly. The broker and topic structure efficiently handled up to 50 robots in simulation without significant latency. However, beyond 100 robots, message delivery delays and occasional packet loss were observed, indicating the need for broker clustering or message batching for larger swarms.

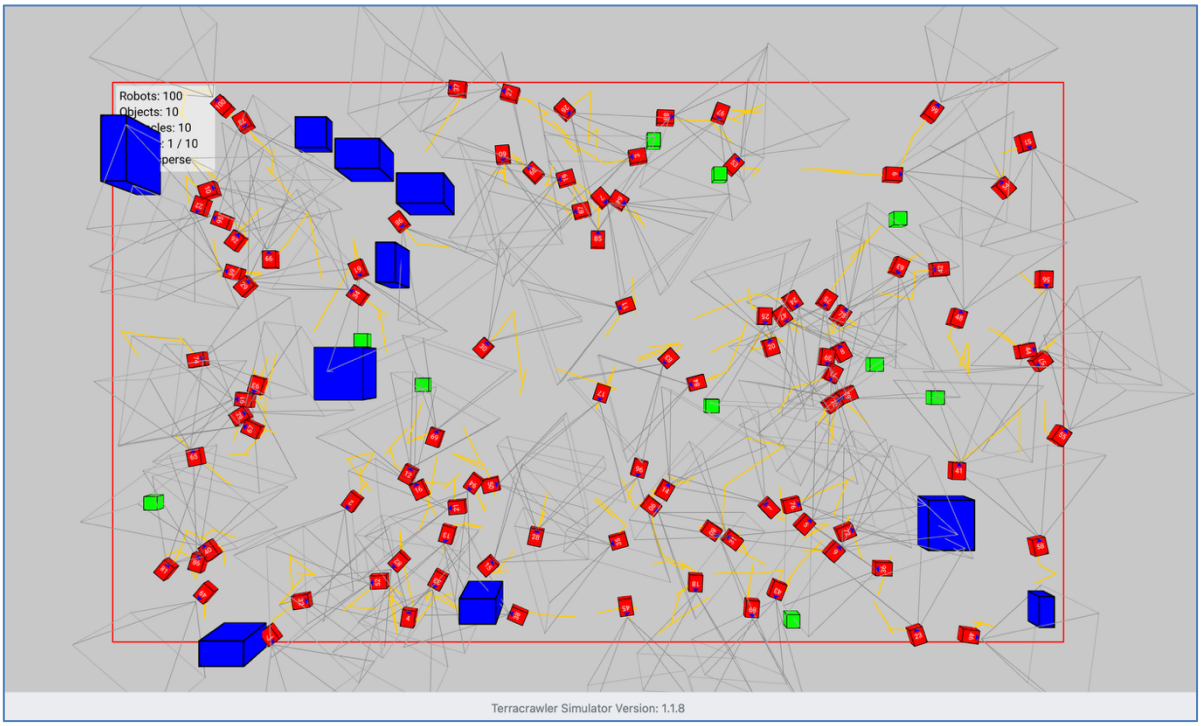

Figure 21: Simulating 100 robots at the same time

Swarm behaviors such as dispersal and gathering remained robust up to 50 robots. With larger swarms, coordination challenges such as congestion and increased collision rates were observed, suggesting the need for decentralized control enhancements.

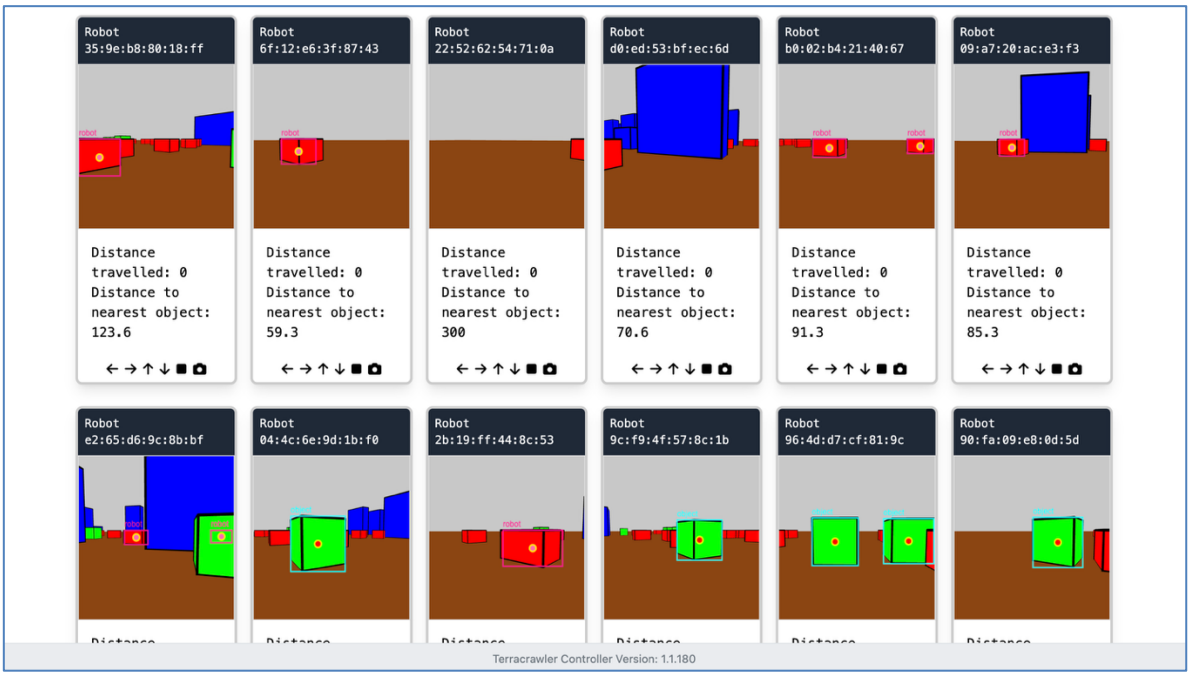

Figure 22: Simulating 50 robots at the same time, UI interface remains usable

Simulation and control server CPU and memory usage scaled with the number of robots. The system maintained real-time performance up to 10 robots, after which frame rates and control loop responsiveness began to degrade. Notably, reinforcement learning training is

significantly more resource intensive than inference or manual control. It was observed that training with more than 10 robots simultaneously became less practical due to increased computational demands and slower convergence.

The web interface remained responsive up to 50 robots, with real-time visualization and control. For larger swarms, UI updates became less fluid, highlighting the importance of visualization abstraction and selective rendering for scalability.

6.2.4 Integration Framework

The integration framework between physical and virtual environments has been designed and implemented, though full testing with physical robots remains as future work:

Table 10. Implementation status

Integration Aspect	Design Approach	Implementation Status
Command Unification	Common MQTT topic structure	Complete, tested in simulation
State Synchronization	Bidirectional state updates	Implementation complete, awaiting physical testing
Visual Representation	Digital twin visualization	Complete for simulation
Behavioral Consistency	Parameter calibration framework	Framework complete, calibration pending physical testing
Development Workflow	Unified interface for both domains	Complete, tested with simulation

The digital twin approach has been fully implemented in the software architecture, with interfaces ready for physical robot integration once that phase of testing begins.

7 Discussion

The development and evaluation of the cloud-based robot swarm control system provides significant insights into the integration of physical and virtual robotics environments, the application of reinforcement learning to resource-constrained platforms, and the scalability challenges of distributed robot systems.

7.1 Key Findings and Implications

7.1.1 Unified Control Architecture

The unified MQTT-based communication architecture shows significant promise based on simulation testing, with several important implications for future work.

Development Efficiency: The architecture is designed to enable seamless transition between simulation and physical testing, which is expected to accelerate the development cycle compared to traditional approaches. This efficiency gain would primarily result from early identification of issues in simulation before physical deployment.

Operational Flexibility: The system's ability to operate mixed swarms of physical and virtual robots (once physical testing begins) should enable novel applications where physical robot capabilities can be extended with virtual counterparts.

Educational Value: The unified interface significantly lowers the barrier to entry for robotics education, allowing students to begin with virtual robots and progressively incorporate physical hardware as resources permit.

Deployment Strategy: The digital twin approach is designed to enable a phased deployment strategy where algorithms are thoroughly validated in simulation before physical implementation, reducing risk and resource requirements for large-scale deployments.

These findings suggest that unified control architectures represent a promising direction for future robotics development, particularly for multi-robot systems where testing at scale presents significant challenges.

7.1.2 Reinforcement Learning Effectiveness

The application of reinforcement learning to the navigation task demonstrated both the potential and limitations of this approach.

Learning vs. Programming: For navigation tasks with variable environments, the reinforcement learning approach demonstrated greater adaptability than traditional programmed approaches. The learned policies could handle novel obstacle configurations more effectively than rule-based alternatives.

Transfer Learning Challenges: The reality gap between simulation and physical robots

remains a significant challenge, requiring careful calibration and domain randomization to achieve effective transfer. This highlights the importance of simulation fidelity for reinforcement learning applications.

Resource Constraints: Deploying neural network policies on ESP32-S3 devices required significant optimization, including quantization and model pruning. While successful, this highlights the ongoing challenge of implementing modern AI approaches on embedded systems.

Emergent Behaviors: The emergence of effective coordination behaviors from individually trained policies was a particularly interesting outcome, suggesting that complex swarm behaviors can arise from relatively simple individual policies.

These findings suggest that reinforcement learning represents a valuable approach for developing adaptive robot behaviors, particularly when combined with a simulation environment for efficient training. However, the reality gap and resource constraints remain important considerations for practical implementations.

7.1.3 Scalability Considerations

The scalability analysis revealed several important insights about distributed robot control systems as discussed below.

Communication Bottlenecks: As swarm size increases, communication becomes the primary bottleneck rather than computational resources. This suggests that future designs should prioritize communication efficiency over raw processing power.

Decentralization Benefits: The hybrid approach combining cloud coordination with local robot autonomy proved more scalable than fully centralized control. This supports the theoretical advantages of distributed control architectures for large-scale swarms.

Visualization Challenges: As swarm size increases, effective visualization becomes increasingly challenging. Abstraction mechanisms that provide summary information while allowing drill-down to specifics are essential for managing large swarms.

Resource Allocation: Dynamic resource allocation in the cloud environment effectively handled varying computational demands, suggesting that cloud-based architectures offer significant advantages for systems with fluctuating resource requirements.

These findings highlight the importance of considering scalability from the earliest design

stages, particularly for systems intended to support varying numbers of robots. The combination of cloud resources with local autonomy appears to offer the most promising approach for scalable swarm robotics.

7.1.4 Limitations and Challenges

Several technical limitations were identified, including the restricted perception capabilities of ultrasonic sensors, limited localization accuracy due to reliance on dead reckoning, dependence on Wi-Fi connectivity, battery life constraints, and the lack of physical robustness in prototype robots. Methodologically, quantifying the reality gap proved challenging, as did ensuring sufficient training data diversity, managing the complexity of parameter tuning, and achieving comprehensive testing coverage. Conceptually, the system design required balancing autonomy and operator control, determining appropriate abstraction levels for different users, attributing faults in a distributed system, predicting emergent behaviors, and understanding scalability limits as the swarm size increases.

7.2 Addressing the Research Questions

Looking back at the research questions posed at the start of my journey helps frame what I have learned from designing and implementing my cloud-based robot swarm control system. This section examines how the empirical findings address each research question, highlighting both achievements and limitations encountered along the way.

Research Question 1: How does the integration of physical and simulated robots in a unified control system impact the efficiency and reliability of swarm robotics experiments?

From an efficiency standpoint, the unified MQTT-based architecture eliminates the need to rebuild control logic when switching between simulation and hardware testing or training. The ability to run both training and testing by simply toggling a switch saves considerable development time. While I have not yet fully integrated physical robots, simulation results suggest that using a single interface should reduce duplicated effort and make transitions between environments smoother. However, the reality gap—differences between simulation and real-world behavior—remains a challenge. Sensor noise, friction variations, and motor response differences will almost certainly require additional tuning when

physical testing begins. In theory, pre-trained reinforcement learning models can be further improved by continuing training on the physical robots.

Research Question 2: To what extent can a browser-based simulation environment accurately represent real-world robot behavior in swarm applications?

My P5.js simulation models core behaviors such as movement, sensing, and collision detection, and it works well for developing and testing algorithms. The environment is accessible and runs efficiently for moderate swarm sizes. However, I intentionally simplified some physics for performance, so the simulation does not fully capture real-world effects like friction or sensor noise. This trade-off makes it a good tool for prototyping and initial training. While some adjustments may still be necessary when transferring algorithms to physical robots, continued training on real hardware (transfer learning) can help bridge much of the remaining gap between simulation and reality.

The browser-based implementation proved surprisingly capable. WebGL rendering, WebGL/WebGPU-supported TensorFlow.js computing, and improved JavaScript performance in recent years have allowed my simulation to handle over 50 robots with acceptable performance. Beyond that number, frame rates declined noticeably, especially during reinforcement learning. Still, for educational applications and many research scenarios, the browser-based approach offers sufficient fidelity while dramatically lowering barriers to entry.

Overall, I deliberately sacrificed some physical realism for computational efficiency. The simulation uses simplified physics rather than a comprehensive parameter-based physics engine. It does not model surface friction variations, battery drain, or sensor noise patterns in detail. These simplifications make the environment more accessible—running smoothly even on modest hardware—but will likely contribute to the reality gap when I begin physical testing.

Research Question 3: How does the MQTT-based distributed communication architecture affect the scalability and real-time performance of the robot swarm system?

My scalability testing revealed both strengths and limitations of the MQTT architecture.

The system performed well with small to medium swarm sizes, but encountered communication bottlenecks as numbers increased.

For swarms up to approximately 50 robots, message delivery remained efficient with minimal latency. The topic-based filtering mechanism proved valuable, ensuring robots received only relevant communications without being flooded by irrelevant messages. This approach maintained system responsiveness for basic navigation and coordination tasks.

As swarm size increased beyond 100 robots, however, issues emerged. Message delivery delays became noticeable, and occasional packet loss occurred despite using QoS level 1. The broker became a bottleneck, suggesting that for larger swarms, a clustered broker approach would be necessary. Network bandwidth utilization also increased substantially, pointing to the need for message batching or compression strategies in larger deployments.

Real-time performance remained acceptable for control operations with moderate swarm sizes. However, more intensive operations like reinforcement learning showed earlier signs of degradation, becoming impractical beyond about 10 robots. This suggests that while MQTT provides an effective communication foundation, additional optimization strategies would be needed for computation-heavy operations in larger swarms.

The hierarchical topic structure proved particularly valuable for efficient messaging. It enabled both broadcast commands to all robots and targeted instructions to specific units. This flexibility supported various control paradigms from centralized direction to more autonomous operation.

7.3 Future Work

Future work will focus on developing larger and more robust robots, potentially incorporating wheel-leg hybrid locomotion and flexible manipulator arms, as well as enhancing onboard computational capabilities. Improvements in sensing and perception are planned through the integration of additional sensor modalities, the development of distributed sensing approaches, and the implementation of collective mapping and environmental modeling. Learning capabilities will be expanded by exploring multi-agent

reinforcement learning algorithms, developing transfer learning techniques to reduce the impact of the reality gap, implementing continuous learning during operation, and investigating federated learning for distributed knowledge acquisition. Communication capabilities may be extended through mesh networking, the development of communication-constrained coordination algorithms, exploration of stigmergic and visual communication methods, and the integration of advanced digital twin concepts, including predictive digital twins, hybrid reality interfaces, hierarchical modeling, and applications for robot health monitoring and predictive maintenance.

8 Conclusion

This thesis presented the design, implementation, and evaluation of a cloud-based control system for robot swarms, integrating both physical and simulated ESP32-S3-based robots through a unified web interface. The research demonstrated the feasibility and advantages of a unified architecture for scalable swarm control, leveraging browser-based simulation and MQTT-based communication to enable efficient development, testing, and deployment.

Simulation-based experiments validated the system's core functionalities, including navigation, coordination, and reinforcement learning-driven behaviors. The results highlight the benefits of cloud-based architectures for scalability and rapid prototyping, while also identifying key challenges such as the reality gap, resource constraints, and communication bottlenecks. Although full integration testing with physical robots remains as future work, the developed framework provides a robust foundation for further research and practical applications in distributed robotics.

Future work will focus on expanding physical testing, enhancing sensing and learning capabilities, and further optimizing the system for larger and more heterogeneous robot swarms. The approaches and findings of this thesis contribute to the advancement of swarm robotics, offering practical insights and tools for both academic research and real-world deployment.

9 References

Albrecht, S. V., Christianos, F., & Schäfer, L. (2024). *Multi-agent reinforcement learning*: Foundations and modern approaches. MIT Press.

Choi, H., Crump, C., Duriez, C., Elmquist, A., Hager, G., Han, D., Hearl F., Hodgins, J., Jain, A., Leve, F., Li, C., Meier, F., Negrut, D., Righetti, L., Rodriguez, A., Tan, J., & Trinkle, J. (2021). *On the use of simulation in robotics*: Opportunities, challenges, and suggestions for moving forward. Proceedings of the National Academy of Sciences, 118(1), e1907856118. https://doi.org/10.1073/pnas.1907856118

Cyberbotics Ltd. (2021). *Robotics simulation with Webots*.

Daniele, A. F. (2023). Accessible Interfaces for the Development and Deployment of Robotic Platforms. arXiv. https://arxiv.org/abs/2305.09848

Das, S., & Others. (2024). Bio-inspired swarm robotics and control: Algorithms, mechanisms, and strategies. IGI Global.

De Vos, M. (2023). *Federated deep reinforcement learning for multi-robot systems* [Master's thesis, University of Manchester]. https://research.manchester.ac.uk/files/348762833/2202.01141v2.pdf

Dorigo, M., & Trianni, V. (2021). *Swarm robotics: Past, present, and future*. Proceedings of the IEEE, 109(7), 1152–1157.

Edge Impulse. (2025). *Detect objects with centroid (FOMO)*. Retrieved March 1, 2024, from https://docs.edgeimpulse.com/docs/tutorials/end-to-end-tutorials/computer-vision/object-detection/detect-objects-using-fomo

Fiveable Team. (2024). *Simulation environments for robot evolution*.

Formant Inc. (2025). *Best robot simulators*.

Goldschmid, P., & Ahmad, A. (2025). *Integrated multi-simulation environments for aerial robotics research*.

Gouda, M., & Multari, N. (1991). *Communication protocols in cloud robotics: Enabling seamless interactions between physical robots and cloud infrastructure*. IGI Global.

Harmon, S. Y., & Gage, D. W. (2016). *Protocols for robot communications: Transport and content layers*. Academia.edu.

Iskandar, A., & Kovács, B. (2025). *Curriculum learning for deep reinforcement learning in swarm robots: A case study with E-puck robots*. International Journal of Robotics Research, 44(3), 289-

307.

J-Stage. (2023). *The design of a web-based master system for robot control with ROS*. https://www.jstage.jst.go.jp/article/jsceiiai/4/2/4_60/_html/-char/ja

Jevtic, A., & Andina, D. (2017). *Swarm intelligence and its applications in swarm robotics*. Universidad Politécnica de Madrid.

Kargar, B., et al. (2023). Emerging Trends in Realistic Robotic Simulations: A Comprehensive Systematic Literature Review. IEEE Access. https://researchonline.rca.ac.uk/5863/1/Emerging_Trends_in_Realistic_Robotic_Simulations_A_Comprehensive_Systematic_Literature_Review%20(1).pdf

Kehoe, B., Patil, S., Abbeel, P., & Goldberg, K. (2015). *A survey of research on cloud robotics and automation*. IEEE Transactions on Automation Science and Engineering, 12(2), 398-409.

Lédeczi, Á., et al. (2022). *Browser-based simulation platforms in education research*.

Mahajan, A., Rashid, T., Samvelyan, M., & Whiteson, S. (2019). *MAVEN: Multi-agent variational exploration*. Journal of Machine Learning Research, 20(115), 1–53. https://jmlr.org/papers/volume20/18-476/18-476.pdf

MDPI Applied Sciences. (2019). *Swarm robotics*. MDPI Books.

Pikman, J. (2022). *Federated deep reinforcement learning for multi-robot navigation* [Bachelor's thesis, Czech Technical University]. https://dspace.cvut.cz/bitstream/handle/10467/101275/F3-BP-2022-Pikman-Jan-main.pdf

Rana, M. M., & Ibrahim, U. M. (2024). *Exploring the role of reinforcement learning in swarm robotics*. European Journal of Electrical Engineering and Computer Science, 8(3), 16–28.

Reina, G., De Vos, M., & Reina, A. (2023). *Federated deep reinforcement learning for robot swarms*. IEEE Transactions on Cognitive and Developmental Systems, 15(2), 1–12. https://roboroyale.eu/pdfs/2023_TCDS_federated.pdf

ROS2WASM: *Bringing the Robot Operating System to the Web*. (2024). arXiv. https://arxiv.org/html/2409.09941v2

Sabry, F. (2025). *Gazebo simulator*. Apple Books.

Singh, A., et al. (2021). *Reinforcement learning in robotic applications: A comprehensive review*. Journal of Robotics Research, 40(5), 567–589.

Slowik, A., & Slowikova, O. (2020). *Swarm intelligence algorithms: Modifications and applications*. Routledge.

Sun, X., Liu, Y., & Zhang, L. (2025). *Digital twins to embodied artificial intelligence: Review and perspective*. Intelligent Robotics, 5(1), 202-227.

Torne, M., Simeonov, A., Li, Z., Chan, A., Chen, T., Gupta, A., & Agrawal, P. (2024). *Reconciling reality through simulation: A real-to-sim-to-real approach for robust manipulation*. ArXiv.

Tselegkaridis, G., & Sapounidis, T. (2023). *Browser-based simulation for novice-friendly classroom robotics*. Frontiers in Computer Science.

Weiss, B., & NIST Team. (2024). *Building a digital twin for robot workcell prognostics and health management: Applications in robotic accuracy assessment*. National Institute of Standards and Technology Technical Report.

10 Appendices

This appendix provides detailed information about the ESP32-S3 robotic system's software architecture, expanding on the high-level overview presented in Section 4.2.2.

10.1 Layered Architecture

The firmware follows a layered approach with clear separation of responsibilities. The diagram illustrates the separation of responsibilities across four layers: Application, Communication, Hardware, and System. Each layer abstracts functionality to promote modularity, reusability, and maintainability in the system design.

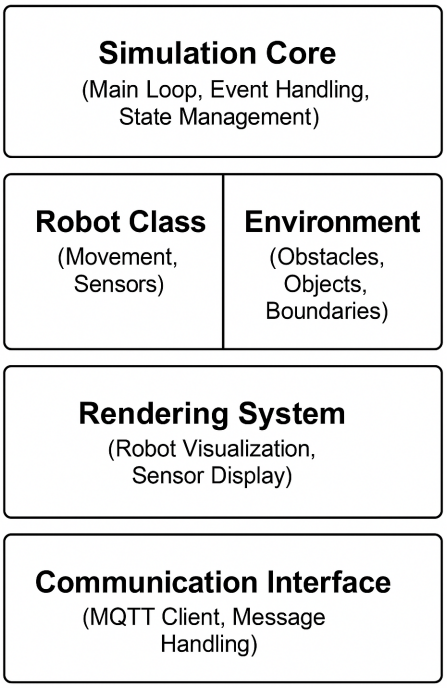

Figure 23: Layered architecture of the robotics control system.

10.1.1 Detailed Component Descriptions

RobotControl Module

Implemented in `robot_control.cpp/robot_control.h`, this module abstracts motor control operations through methods like `forward()`, `backward()`, `turnLeft()`, and `turnRight()`. It handles the complexity of motor speed control, directional changes, and coordinated movements.

WiFi Manager

The WiFi Manager handles network connectivity, implementing reconnection logic, access point fallback modes, and network configuration persistence. It ensures robots remain connected to the control network even in challenging wireless environments.

MQTT Client

The MQTT client manages publish-subscribe communication with the cloud server, handling message serialization, topic subscription, connection retries, and quality-of-service settings. It supports both command reception and telemetry transmission through standardized message formats.

Sensor Integration

This component interfaces with ultrasonic distance sensors (US-100) for obstacle detection, implementing timing-based distance measurements, noise filtering, and sensor calibration routines.

Visual Processing

For robots equipped with the ESP32-Camera module, this optional component handles image capture, preprocessing, and object detection using the FOMO (Faster Objects, More Objects) algorithm.

LED Control System

Provides visual status indicators through controlled LED patterns, implementing configurable blink patterns to indicate system states, errors, and operational modes.

10.1.2 Multitasking Implementation

The firmware leverages FreeRTOS to implement a multitasking environment with several concurrent tasks (Code example 11).

Code example 11: Task structure excerpt

```
xTaskCreate(communicationTask, "mqtt_comm", 4096, NULL, 5, NULL);
```

```
xTaskCreate(sensorMonitorTask, "sensor_monitor", 2048, NULL, 3, NULL);
xTaskCreate(motorControlTask, "motor_control", 2048, NULL, 4, NULL);
xTaskCreate(batteryMonitorTask, "battery_monitor", 1024, NULL, 2, NULL);
```

These tasks operate with different priorities to ensure critical functions (like communication and motor control) receive appropriate CPU time while lower-priority tasks (like battery monitoring) run when resources are available.

10.1.3 Communication Architecture

The MQTT communication architecture follows a topic-based hierarchy that enables efficient message routing (Code example 12).

Code example 12: MQTT topics and JSON message snippet

```
robots/all/command         // Broadcast commands to all robots
robots/{id}/command        // Commands for specific robot
robots/{id}/status         // Robot status information
robots/{id}/sensors        // Sensor data from specific robot

// Messages use a JSON format for structured data exchange:
{
  "command": "move",
  "parameters": {
    "direction": "forward",
    "speed": 80,
    "duration": 2000
  },
  "timestamp": 1648372964
}
```

10.1.4 Control Algorithm Implementation

The control algorithms implemented on the ESP32-S3 robots enable both autonomous operation and coordinated swarm behavior. These algorithms span multiple levels of abstraction, from low-level motor control to high-level navigation and obstacle avoidance.

10.1.5 Motor Control Implementation

The differential drive system uses PWM (Pulse Width Modulation) for precise speed control of the two DC motors through the L298N motor driver. The implementation leverages ESP32's LEDC peripheral for efficient PWM generation (Code example 13).

Code example 13: Robot motor control implemented in ESP-IDF

```
// Motor control PWM configuration
ledc_timer_config_t ledc_timer = {
    .speed_mode = LEDC_LOW_SPEED_MODE,
    .duty_resolution = LEDC_TIMER_8_BIT,  // 8-bit resolution (0-255)
```

```
    .timer_num = LEDC_TIMER_0,
    .freq_hz = 5000,                      // 5kHz PWM frequency
    .clk_cfg = LEDC_AUTO_CLK
};

// Channel configuration for left and right motors
ledc_channel_config_t ledc_channel_left = {
    .gpio_num = enaPin_,
    .speed_mode = LEDC_LOW_SPEED_MODE,
    .channel = leftPwmChannel_,
    .timer_sel = LEDC_TIMER_0,
    .duty = 0,                            // Initial duty cycle (0%)
    .hpoint = 0
};
```

The RobotControl class provides gradual speed changes to prevent abrupt motor transitions, improving mechanical reliability and energy efficiency (Code example 14).

Code example 14: Motor control implementation example in real robots

```
void RobotControl::gradualSpeedChange(int motor, int targetSpeed) {
    // Get current speed of the specified motor
    int currentSpeed = (motor == 1) ? currentSpeedMotor1_ : currentSpeedMotor2_;

    // Gradually change speed in small increments
    while (currentSpeed != targetSpeed) {
        if (currentSpeed < targetSpeed) {
            currentSpeed = min(currentSpeed + SPEED_INCREMENT, targetSpeed);
        } else {
            currentSpeed = max(currentSpeed - SPEED_INCREMENT, targetSpeed);
        }

        // Update motor speed
        setMotorSpeed(motor, currentSpeed);

        // Small delay between increments
        vTaskDelay(pdMS_TO_TICKS(20));
    }
}
```

10.1.6 Distance Sensing and Obstacle Avoidance

Code example 15: Sensor reading implementation example in real robots

```
float RobotControl::measureDistance() {
    // Trigger pulse (10µs)
    gpio_set_level(TRIG_PIN, 1);
    ets_delay_us(10);
    gpio_set_level(TRIG_PIN, 0);

    // Measure echo pulse duration
    int64_t start = esp_timer_get_time();
    while (gpio_get_level(ECHO_PIN) == 0 && esp_timer_get_time() - start <
TIMEOUT_US);

    start = esp_timer_get_time();
    while (gpio_get_level(ECHO_PIN) == 1 && esp_timer_get_time() - start <
TIMEOUT_US);
    int64_t end = esp_timer_get_time();

    // Calculate distance (sound travels at ~343m/s)
    float distance_cm = (end - start) * 0.0343 / 2;
    return distance_cm;
```

```
}
```

The obstacle avoidance algorithm implements a reactive approach with three main behaviors:

1. **Free Movement**: When no obstacles are detected within threshold distance
2. **Obstacle Detection**: Reduction in speed when approaching obstacles
3. **Avoidance Maneuver**: Rotation to find clear path when obstacles are imminent

These behaviors are implemented as a simple state machine (Code example 16).

Code example 16: Obsacle avoidance

```
void obstacleAvoidance() {
    float distance = measureDistance();

    if (distance > SAFE_DISTANCE) {
        // Free movement
        currentState = STATE_MOVING;
        forward(NORMAL_SPEED);
    } else if (distance > CRITICAL_DISTANCE) {
        // Obstacle approaching - slow down
        currentState = STATE_CAUTION;
        forward(SLOW_SPEED);
    } else {
        // Obstacle imminent - avoid
        currentState = STATE_AVOIDING;
        stop();
        // Turn in place to find clear path
        turnRight(TURN_SPEED);
        // Continue turning until clear path found
        while (measureDistance() < SAFE_DISTANCE) {
            vTaskDelay(pdMS_TO_TICKS(100));
        }
    }
}
```

10.1.7 Visual Detection

For robots equipped with the ESP32-Camera module, a lightweight object detection algorithm is implemented using Edge Impulse's FOMO (Faster Objects, More Objects) approach. While full integration remains as future work, the visual detection module was developed to enable centroid-based detection that is optimized for resource-constrained environments:

1. **Image Acquisition**: Capturing frames from the onboard camera
2. **Preprocessing**: Image resizing and normalization
3. **Model Inference**: Running the pre-trained FOMO model
4. **Result Processing**: Extracting detected object centroids and classes

The model runs directly on the ESP32-S3, leveraging its vector instruction capabilities for efficient matrix operations.

10.1.8 Control Loop Integration

The various control algorithms are integrated within a main control loop that manages sensor reading, decision making, and actuation (Code example 17).

Code example 17: Unobscuring motor control

```
void motorControlTask(void* parameter) {
    TickType_t lastWakeTime = xTaskGetTickCount();

    while (true) {
        // Read sensor data
        float distance = robotControl.measureDistance();

        // Get current command from command queue
        CommandMessage currentCommand;
        if (xQueueReceive(commandQueue, &currentCommand, 0) == pdTRUE) {
            // Process new command
            executeCommand(currentCommand);
        } else if (autonomousMode) {
            // Run autonomous behavior
            obstacleAvoidance();
        } else {
            // Maintain current state
            maintainState();
        }

        // Update motor outputs
        updateMotors();

        // Ensure consistent timing
        vTaskDelayUntil(&lastWakeTime, pdMS_TO_TICKS(CONTROL_LOOP_PERIOD_MS));
    }
}
```

10.1.9 Parameter Tuning for Differential Drive

The system implements a straightforward yet effective approach to motion control for differential drive robots. Rather than using a specialized calibration structure, the implementation directly controls motor behavior through software parameters and GPIO manipulation (Code example 18).

Code example 18: Speed and direction control of motors

```
void RobotControl::setupMotor(int motor, int speed, bool direction) {
    // Check if there's a global stop action that needs to be handled first
    if (stopCurrentAction) {
        stopMotors(); // Ensure motors are stopped if a stop flag is raised
        return;
    }

    // Apply speed and direction settings to each motor based on the 'motor'
parameter
```

```
    switch (motor) {
        case 1: // Left Motor
            setMotorSpeed(1, speed);
            setMotorDirection(1, direction);
            break;
        case 2: // Right Motor
            setMotorSpeed(2, speed);
            setMotorDirection(2, direction);
            break;
        case 0: // Both Motors
            setMotorSpeed(1, speed); // Apply the same speed and direction to both
motors
            setMotorDirection(1, direction);
            setMotorSpeed(2, speed);
            setMotorDirection(2, direction);
            break;
    }
}
```

Differential steering is achieved through the independent control of the left and right motors. For turning behaviors, different speeds are applied to each motor to create the desired rotational effect (Code example 19).

Code example 19: Differential steering implementation

```
void RobotControl::turnLeft(int speed) {
    setupMotor(1, speed, true);  // Left motor forward
    setupMotor(2, speed*0.1, false); // Right motor backward for sharper turn
}

void RobotControl::turnRight(int speed) {
    setupMotor(1, speed*0.1, false);  // Left motor backward
    setupMotor(2, speed, true);  // Right motor forward for sharper turn
}
```

Note that the turning algorithm uses a multiplication factor (0.1) to create a more pronounced differential between the wheels during turning maneuvers. This simple approach creates effective turning behavior without requiring complex parameter tuning.

For movement operations that require gradual speed changes, the system implements a slowStop method that incrementally reduces motor speed to prevent abrupt movements (Code example 20).

Code example 20: Slow stop implementation

```
void RobotControl::slowStop(int timeMs)
{
    int steps = 10; // Number of steps in which to stop the motor
    int delayBetweenSteps = timeMs / steps;
    int stepReductionAmountLeft = currentSpeedMotor1_ / steps;
    int stepReductionAmountRight = currentSpeedMotor2_ / steps;

    for (int i = 0; i < steps; ++i)
    {
        currentSpeedMotor1_ -= stepReductionAmountLeft;
        currentSpeedMotor2_ -= stepReductionAmountRight;
```

```
        setMotorSpeed(1, currentSpeedMotor1_);
        setMotorSpeed(2, currentSpeedMotor2_);

        vTaskDelay(delayBetweenSteps / portTICK_PERIOD_MS);
    }

    // Ensure the motors are completely stopped
    setMotorSpeed(1, 0);
    setMotorSpeed(2, 0);
}
```

The motor control implementation leverages the ESP32's LEDC (LED Control) peripheral for PWM generation, converting percentage-based speed values to appropriate duty cycles (Code example 21).

Code example 21: Motor speed setting

```
void RobotControl::setMotorSpeed(int motor, int speed)
{
    // Convert speed to duty cycle
    int dutyCycle = (speed * 255) / 100;

    // Set the speed and update the current speed for the appropriate motor
    if (motor == 1)
    {
        currentSpeedMotor1_ = speed;
        ledc_set_duty(LEDC_LOW_SPEED_MODE, leftPwmChannel_, dutyCycle);
        ledc_update_duty(LEDC_LOW_SPEED_MODE, leftPwmChannel_);
    }
    else if (motor == 2)
    {
        currentSpeedMotor2_ = speed;
        ledc_set_duty(LEDC_LOW_SPEED_MODE, rightPwmChannel_, dutyCycle);
        ledc_update_duty(LEDC_LOW_SPEED_MODE, rightPwmChannel_);
    }
}
```

While this implementation doesn't include the parameter adjustment mechanism described earlier, it provides a robust foundation for robot movement control with direct manipulation of motor speed and direction. The approach prioritizes simplicity and reliability while still enabling effective differential drive control for the ESP32-S3 robots.