POLITECNICO DI TORINO

Master's Degree in Computing Engineering

Master's Degree Thesis

Leveraging Edge Computing Resources on Computing-Intensive Robotic Tasks

Supervisors
Prof. Fulvio RISSO
Ing. Jacopo MARINO
MSc. Daniele CACCIABUE

Candidate
Sebastian MONTOYA

December 2025

Acknowledgements

This thesis would not have been possible without the support of many wonderful people.

First and foremost, I want to thank my parents, Juan and Angie, for their unwavering love and encouragement throughout this journey. Your belief in me has been my constant source of strength. To my brother Juanca, thank you for always being there, for the laughs when I needed them most, and for reminding me that life exists beyond my responsabilities.

To my grandmother Oti, your wisdom and warmth have guided me through the toughest moments. Thank you for always having faith in me.

I am deeply grateful to my supervisor, Professor Fulvio, for his invaluable guidance, patience, and mentorship. Your insights have shaped not only this work but also my growth as a Professional. To Jacopo and Daniele, thank you for your tireless assistance, thoughtful feedback, and for making the research process both productive and enjoyable.

Finally, to everyone who has been part of this adventure—thank you for making it memorable.

With gratitude,
Sebas

Summary

This thesis presents a distributed robotic system for real-time stop sign detection using ROS 2, computer vision, and edge computing technologies. The system is composed of a robot with an Intel RealSense D435 camera for depth perception and RGB image capture, movement capabilities and considerable networking and computational resources.

These resources will be leveraged to to perform gpu intensive tasks, specifically stop sign detection using YOLO. Said tasks will be offloaded to edge computing resources when available, allowing us to study the performance of such a system.

System Architecture

The system implements a ROS 2-based stop sign detection node using YOLO for object detection. The node subscribes to RGB and depth camera streams from the RealSense D435, enabling both object detection and distance estimation. The detection algorithm is optimized specifically for stop sign recognition to improve efficiency and reduce computational overhead.

A key element of the architecture is the use of ROS 2 Lifecycle Nodes that can dynamically activate or deactivate based on system conditions. This allows the robot to intelligently manage resource consumption by stopping local processing when offloading to remote servers.

To enable seamless use of remote resources, the system integrates Liqo to establish peer-to-peer connections between the robot's Kubernetes cluster and edge computing resources. A custom monitoring node continuously evaluates Liqo's health status and peer availability to make real-time decisions about task distribution.

When remote processing is available, the system uses WebSocket connections to stream camera images to edge servers. The edge servers perform YOLO detection and return coordinates and classification results, which are then integrated back into the robot's perception pipeline. This communication path is designed to minimise bandwidth use and latency while preserving detection accuracy.

Operational Modes

The system operates in two distinct modes:

- **Local Processing Mode:** When Liqo peers are unavailable or network conditions are poor, all detection processing occurs locally on the robot using onboard computational resources.
- **Edge Offloading Mode:** When Liqo successfully establishes connections to edge computing resources, camera images are transmitted via WebSocket to remote servers for processing, allowing the robot to conserve local computational resources.

This work demonstrates a practical implementation of adaptive edge computing in robotics by combining ROS 2 lifecycle management, efficient computer vision for stop sign detection, peer-to-peer Kubernetes networking via Liqo, and WebSocket-based offloading. The combined design allows the robot to dynamically switch between local and remote processing based on real-time network and peer availability, conserving local compute when possible and maintaining robust perception when operating in degraded network conditions. The integration of Lifecycle Nodes for dynamic resource management and Liqo for peer discovery and networking are central contributions, showing how modern containerization and networking technologies can be applied to create flexible, resource-efficient robotic systems.

Table of Contents

List of Figures

Chapter 1

Introduction

Modern autonomous robotics faces an increasingly complex computational challenge: they need to process great amounts of data on real time while looking to achieve a balance between weight reduction, computational power and thermal constraints. With the ongoing interest on AI applications these complications become compounded due to the increasing demand for "intelligent" decision making in realms with a lot of variables, hence input and at the same time with the beforehand mentioned computational constraints. This thesis would like to investigate how edge computing can be leveraged to address these challenges, particularly in the context of mobile robotics in the area of computer vision, working with Liqo, a computational distribution tool designed for dynamic resource allocation and task offloading in edge computing environments.

1.1 Problem Statement and Resource Constraints

Mobile robotic platforms, particularly autonomous vehicles and service robots, face fundamental trade-offs between computational capability and operational constraints. These systems must process high-resolution camera feeds, LiDAR point clouds, accelerometers, gyroscopes, arms, and other sensor data streams while maintaining real-time responsiveness for critical applications. However, several key limitations constrain onboard computational resources:

Power Consumption: High-performance CPUs and GPUs consume significant power, directly impacting battery life and operational range. For mobile robots, this creates a critical constraint where computational intensity must be balanced against mission duration and energy efficiency.

Thermal Management: Intensive computational workloads generate substantial heat, requiring active cooling systems that add weight, complexity, and additional power consumption. In compact robotic platforms, thermal throttling can significantly reduce available processing power.

Size and Weight Constraints: Mobile robots have strict limitations; high-performance computing hardware often conflicts with both requirements, usually leading to a vicious cycle of under-performance and increased resource demands.

Real-time Requirements: Many robotic applications, particularly those involving navigation and obstacle avoidance, require deterministic response times. In most of these applications the onboard capabilities may fall short.

Cost Considerations: Equipping every robotic platform with high-end computational hardware significantly increases system cost, limiting scalability and commercial viability.

These constraints create a fundamental challenge: how can robotic systems access the computational resources needed for advanced Artificial Intelligence (AI) and computer vision algorithms while maintaining the mobility, efficiency, and reliability required for real-world deployment?

1.2 Edge Computing as a Solution

Edge computing presents a promising solution to the computational limitations of mobile robotics by enabling dynamic distribution of processing tasks from local to Edge "remote" devices which can offer greater computational resources without the latency penalties of traditional cloud computing.

Latency Reduction: By processing data at relatively close proximity to the source, edge computing can achieve sub-millisecond to low-millisecond latencies suitable for real-time robotic applications. This enables offloading of computationally intensive tasks without compromising safety-critical response times.

Adaptive Resource Allocation: Edge computing enables dynamic task allocation based on current network conditions, computational load, and power constraints. Robots can intelligently decide which tasks to process locally versus remotely, optimizing for performance, energy efficiency, or reliability as conditions change.

Scalability: Edge infrastructure can serve multiple robotic platforms simultaneously, providing a wide range of computational resources to individual robots which allows them to remain lightweight and cost-effective.

Reliability and Graceful Degradation: Well-designed edge computing systems can provide fallback capabilities, allowing robots to continue operation with either reduced functionality or higher energy consumption when network connectivity is lost or edge resources are unavailable.

However, realizing these benefits requires sophisticated coordination mechanisms that can dynamically manage the distribution of computational tasks based on changing conditions. This coordination must account for network quality, computational load, power constraints, and application requirements to make optimal decisions in real-time.

1.3 Prior Work and State of the Art

A comprehensive review of the state of the art is essential to position this research within the broader landscape of edge computing for robotics. Current research in this domain focuses on several key areas:

Edge Computing Architectures for Robotics: Recent studies have demonstrated that edge-only configurations can achieve average decision latencies as low as 45 ms, compared to 210 ms in cloud-only configurations, representing a critical advancement for real-time robotic applications[1]. Edge computing architectures have been shown to improve energy efficiency by up to 4.7x compared to purely local processing solutions.

Middleware and Orchestration: ROS 2 has emerged as a production-ready middleware for distributed robotic systems [2], with its Lifecycle Node capabilities providing sophisticated mechanisms for managing computational workloads across distributed resources. Kubernetes and specialized federation tools like Liqo have demonstrated effectiveness for container orchestration in heterogeneous edge environments.

Dynamic Workload Allocation: Recent benchmarking studies have established that ROS 2 Lifecycle Node architectures can achieve switching times reduced by at least 85% compared to alternative approaches[3], making them particularly suitable for dynamic task allocation in edge computing scenarios.

AI Workload Optimization: Computer vision tasks like object detection with YOLO have been successfully optimized for edge deployment through techniques including model quantization and hardware acceleration, with empirical measurements showing inference times as low as 24 milliseconds on edge nodes.

Chapter 2 provides a detailed analysis of these advances and their implications for the present research.

1.4 Use Case: YOLO-based Stop Sign Detection

To investigate the practical implications of edge computing for mobile robotics, this thesis focuses on a representative computer vision workload: real-time stop sign detection using YOLO (You Only Look Once) object detection algorithms [4]. This use case was selected for several compelling reasons:

Safety-Critical Nature: Object detection is fundamental to autonomous navigation and represents a class of safety-critical computer vision tasks where both accuracy and response time are paramount. This domain is important to explore due to its real-time requirements and hardware constraints.

Computational Intensity: YOLO-based object detection requires significant computational resources, involving complex convolutional neural network operations that can strain mobile hardware. This workload effectively demonstrates the computational challenges faced by modern robotic systems.

Real-time Requirements: Object detection must operate with minimal latency to be effective for navigation and safety systems. This requirement makes it an ideal testbed for evaluating the latency and quality characteristics of edge computing solutions.

Variable Computational Demand: The processing requirements for object detection can vary based on image resolution, scene complexity, and detection confidence requirements, providing opportunities to study adaptive resource allocation strategies.

The experimental implementation utilizes ROS 2 (Robot Operating System 2) as the middleware framework, providing a standardized platform for distributed robotic applications. The system architecture includes:

- A mobile robotic platform equipped with camera sensors and onboard computing capability
- Local processing node capable of running YOLO inference locally
- Remote processing service deployed on edge computing infrastructure
- A dynamic Lifecycle manager that coordinates task functionality to optimize resource usage
- A multiplexer system to manage data streams between local and remote processing nodes
- Network monitoring and performance measurement systems

The experimental design evaluates system performance across two network environments: high-reliability wired Ethernet connections and wireless networks with the robot stationary. This comprehensive evaluation captures the range of network conditions that mobile robots encounter in real-world deployments.

1.5 Thesis Contributions

This thesis makes several key contributions to the field of edge computing for mobile robotics:

Dynamic Task Allocation Framework: Development and evaluation of a ROS 2-based Lifecycle management system that can dynamically switch between local and remote processing based on network conditions, computational load, and power constraints. This framework provides a practical foundation for implementing adaptive edge computing in robotic systems.

Real-world Validation: Experimental validation using actual robotic hardware and representative computer vision workloads, demonstrating the practical feasibility of dynamic edge computing for safety-critical applications.

Design Guidelines: Development of evidence-based recommendations for implementing edge computing solutions in mobile robotics, including guidelines for task allocation policies, network requirements, and system architecture considerations.

These contributions advance the understanding and documentation of how edge computing can be effectively applied to mobile robotics while providing practical tools and insights for system designers and researchers in the field.

1.6 Thesis Organization

The remainder of this thesis is organized as follows:

Chapter 2: State of the Art examines the current landscape of edge computing for robotic applications, reviewing architectural approaches, performance characteristics, and optimization techniques that enable efficient deployment of computationally intensive workloads in resource-constrained environments.

Chapter 3: Background provides theoretical foundations on edge computing, mobile robotics, and distributed processing architectures, establishing the technical context for this research.

Chapter 4: System Design presents the detailed design of the ROS 2-based edge computing framework, including the Lifecycle management system, network monitoring components, and task allocation algorithms. This chapter provides the technical foundation for the experimental implementation.

Chapter 5: Implementation describes the experimental setup, robotic platform configuration, and evaluation metrics used to assess system performance. This chapter details the controlled experiments designed to evaluate the edge computing framework under various conditions.

Chapter 6: Evaluation presents comprehensive experimental results, including performance comparisons across different network environments, energy efficiency analysis, and latency characterization. This chapter provides quantitative evidence for the effectiveness of the proposed approach.

Chapter 7: Conclusion summarizes the key findings, reiterates the contributions of this work, and outlines promising directions for future research in edge computing for mobile robotics.

Throughout this thesis, the focus remains on practical, implementable solutions that can be deployed in real-world robotic systems while providing the performance, reliability, and efficiency required for successful autonomous operation.

Chapter 2

State of the Art

2.1 Introduction

Edge computing represents a paradigm shift in how computational resources are distributed and utilized in robotic systems. This chapter examines the current state of the art in edge computing for robotic applications, focusing on architectural approaches, performance characteristics, and optimization techniques that enable efficient deployment of computationally intensive workloads in resource-constrained environments [5, 1].

2.2 Edge Computing and Robotics Constraints

2.2.1 Limitations of Traditional Computing Approaches

Traditional approaches to robotic computing fall into two primary categories: onboard computing and cloud-based computing. Each presents significant limitations for modern robotics applications [6].

Cloud-based computing systems, while offering virtually unlimited computational resources, suffer from unacceptable latency, bandwidth constraints, and security concerns that make them unsuitable for mission-critical robotic applications [6, 7]. The dependency on stable network connectivity also introduces a critical point of failure.

Conversely, onboard computing is severely constrained by power consumption, thermal management limitations, size, and weight considerations [1]. These constraints fundamentally limit the scalability and commercial viability of complex robotic systems that require significant computational resources.

2.2.2 Latency and Real-Time Requirements

Robotics applications, particularly autonomous systems, demand real-time responsiveness and autonomous decision-making capabilities [1, 8]. Conventional cloud architectures cannot meet these requirements due to network-induced latency.

Edge computing addresses this challenge by enabling data processing closer to the physical devices, significantly reducing response times and enhancing reliability [7, 5]. The proximity advantage of edge systems translates directly to very low latency, which is critical for time-sensitive operations.

2.2.3 Quantitative Latency Benefits

Empirical studies demonstrate the quantitative benefits of edge computing for robotics. Edge-only configurations have achieved an average decision latency of 45 milliseconds, compared to 210 milliseconds in cloud-only configurations [1]. This nearly 5x improvement in latency enables robots to react more fluidly to dynamic changes in their environment, which is essential for safety-critical applications [8].

2.2.4 Energy and Operational Efficiency

Edge computing architectures offer significant energy efficiency improvements by optimizing the distribution of computational workloads. By delegating high-complexity computational tasks to edge nodes, robots can significantly reduce their onboard processing load, extending battery life and operational time [1, 7].

Quantitative measurements demonstrate that remote processing frameworks can achieve 4.7x better energy efficiency compared to local processing, consuming 81% less energy while maintaining comparable performance levels (see Chapter 6). This energy efficiency translates directly to extended mission durations and reduced operational costs.

2.2.5 Resilience and Graceful Degradation

Edge architectures support enhanced autonomy and local decision-making, allowing robots to continue functioning even when disconnected from the internet [8]. This capability is crucial for operations in environments with unreliable connectivity.

Well-designed edge systems incorporate fallback capabilities and maintain reliable operation with graceful degradation when network connectivity is compromised [8]. This resilience ensures that critical functionalities remain available even under suboptimal conditions.

2.3 Architectural Components: ROS 2, Kubernetes, and Liqo

2.3.1 ROS 2 as Production Middleware

ROS 2 represents a complete redesign of the original ROS framework, addressing fundamental limitations regarding security, reliability in non-traditional environments, and support for large embedded systems [2].

A key architectural advantage of ROS 2 is its foundation on the Data Distribution Service (DDS), an open middleware standard widely used in critical infrastructure applications [2]. This foundation provides robust communication primitives that are essential for distributed robotic systems.

2.3.2 ROS 2 Lifecycle Nodes

ROS 2 introduces a sophisticated pattern for managing the Lifecycle of nodes, allowing them to transition through well-defined states including Unconfigured, Inactive, Active, and Finalized [2]. This Lifecycle Node (LCN) feature provides essential capabilities for coordinating various components of a distributed asynchronous system.

LCNs facilitate dynamic task switching, allowing computational workloads to be seamlessly transferred between different processing locations without service interruption [3]. This capability is fundamental to the implementation of flexible edge computing architectures.

2.3.3 Container Orchestration with Kubernetes

Kubernetes has emerged as the de facto standard for automating the deployment, scaling, and management of containerized applications [3]. In edge computing contexts, Kubernetes provides critical capabilities for decoupling hardware from software, allowing applications to run as self-contained units across heterogeneous physical platforms.

For edge computing applications, lightweight Kubernetes distributions such as K3s[1] offer particular advantages, providing core orchestration capabilities with reduced resource requirements [7, 3].

2.3.4 Computing Continuum

The Computing Continuum concept unifies diverse computing resources—spanning cloud, fog, edge, and IoT layers—into a seamless computational fabric [9]. This holistic approach enables workloads to be dynamically allocated to the most appropriate computational resources.

Orchestration systems play a vital role in this continuum, automating the seamless delivery of applications across diverse environments and ensuring that Quality of Service (QoS) objectives are consistently met [9].

2.3.5 Federation with Liqo

Liqo represents an important advancement in Kubernetes technology, enabling dynamic multi-cluster configurations across varied infrastructures [3]. As an open-source tool, Liqo enhances Kubernetes with federation capabilities that are particularly valuable for edge computing architectures.

The key innovation of Liqo is its ability to create virtual nodes that represent shared resources from remote clusters, seamlessly expanding the local cluster's resource pool [3]. This federation is achieved through secure network tunnels, enabling transparent resource sharing across cluster boundaries.

2.4 Empirical Validation: Dynamic Switching Performance

2.4.1 The Switching Challenge

A central challenge in edge computing for robotics is the need to guarantee service continuity and transparency when offloading ROS 2 modules to edge clouds [3]. Switching the process of transitioning active logic from one location to another must be accomplished with minimal service disruption.

The ideal switching process is characterized by speed and undetectability, ensuring that the transition between computational resources does not impact the robot's operational capabilities [3].

[1]`https://k3s.io`

2.4.2 Lifecycle Node Performance

Benchmarking studies have established that the ROS 2 Lifecycle Node (LCN) architecture provides superior performance for quick transitions between clusters [3]. The LCN approach enables switching times that are reduced by at least 85% compared to alternative non-redeployment solutions such as SwR, SwRNP, and SwNP.

This performance advantage is particularly significant for hot redundancy architectures, where both nodes are simultaneously running [3, 8]. In such configurations, the time required to switch control using LCN approaches becomes negligibly small.

2.4.3 Energy Efficiency of Switching Approaches

The LCN approach offers additional advantages in terms of energy efficiency. Deactivated Lifecycle Nodes execute significantly less code than their active counterparts, resulting in lower power consumption compared to solutions based on Network Policies (NetPol) [3, 8].

2.4.4 Switching Intervals and Metrics

Comprehensive evaluation of switching performance requires measuring specific intervals, including Time t_1 (the period from request initiation to when the new node begins transmitting data) and Time t_2 (the duration of message overlap or silence) [3].

Approaches requiring redeployment (SwR, SwRNP) consistently show inferior performance for t_1 due to the time required to deploy new nodes [3]. This deployment overhead represents a fundamental limitation for certain switching architectures.

2.5 Workload Context: YOLO and Edge AI Optimization

2.5.1 YOLO as a Representative Workload

Real-time object detection using YOLO (You Only Look Once) algorithms represents an ideal test case for edge computing in robotics [10, 4]. This workload is representative of broader challenges in edge AI due to its safety-critical nature, real-time requirements, and substantial computational demands.

2.5.2 Edge AI Hardware Acceleration

The deployment of AI workloads at the edge has been significantly simplified by specialized hardware such as NVIDIA's Jetson family[2]. These Edge AI devices are specifically designed for AI applications—particularly computer vision—and deliver high performance with low power consumption [4, 10].

2.5.3 Model Optimization Techniques

Modern deep learning tasks such as object detection present significant challenges for edge deployment due to their resource requirements [10, 4]. Model optimization techniques have emerged as essential tools for addressing these constraints.

[2]`https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/`

Quantization—particularly to FP16 precision—can provide a 2x to 6x increase in inference execution speed without significant accuracy degradation [10, 4]. This approach represents an important trade-off between computational efficiency and detection accuracy.

2.5.4 TensorRT Acceleration

NVIDIA TensorRT has emerged as a critical tool for optimizing deep learning models on NVIDIA GPUs. By converting models into highly efficient GPU-executable files, TensorRT enables substantial performance improvements for edge inference [4].

Benchmark studies indicate that model optimization through TensorRT can yield up to 36 times increase in inference speed compared to unoptimized models, dramatically expanding the range of AI workloads that can be effectively deployed on edge hardware [4].

2.5.5 Empirical Performance of Edge AI

Empirical measurements demonstrate the viability of edge-based object detection. YOLOv5s implementations have achieved inference times of 24 milliseconds on edge nodes [1], enabling real-time processing of video streams.

More detailed optimization studies show that YOLOv3-320 inference on a Jetson Nano can be reduced from 323 ms to 154 ms (a 2.10x speedup) through the application of techniques including loop fusion, kernel tuning, and FP16 quantization [10]. These optimizations are essential for enabling real-time perception capabilities on resource-constrained edge devices.

2.6 Conclusion

Edge computing offers a transformative approach to robotics by addressing the fundamental limitations of traditional computing architectures [5, 1]. Through reduced latency, improved energy efficiency, and enhanced resilience, edge computing enables new classes of robotic applications. The combination of ROS 2, Kubernetes, and Liqo provides a robust architectural foundation [2, 3], while sophisticated switching techniques ensure seamless workload migration. As demonstrated through the deployment of computationally intensive AI workloads such as YOLO, edge computing represents a viable and effective approach for next-generation robotic systems [10, 4].

Chapter 3

Background

This chapter provides the theoretical foundation for understanding edge computing in mobile robotics. We begin with an overview of the ROS framework that underlies our implementation, followed by fundamental concepts in edge computing. We then examine the role of computer vision in robotics, with particular focus on YOLO-based object detection. Finally, we review the current state of the art and provide a comparative analysis of different computing paradigms for mobile robotics.

3.1 ROS

ROS is a flexible framework for writing robot software that has become the de facto standard for robotics research and development [2]. Despite its name, ROS is not an operating system in the traditional sense, but rather a middleware layer that provides services typically expected from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management.

3.1.1 ROS 2 Architecture and Design Principles

ROS 2 [2], the second generation of ROS, was designed to address the limitations of the original ROS when applied to production robotics systems. Key improvements include:

Real-time Capabilities: ROS 2 incorporates real-time computing principles, enabling deterministic behavior crucial for safety-critical applications. This is achieved through the use of DDS (Data Distribution Service)[1] as the underlying communication middleware, which provides quality-of-service (QoS) guarantees.

Security: Unlike the original ROS, ROS 2 includes built-in security features, including authentication, authorization, and encryption capabilities[2]. This is essential for deployment in production environments where security vulnerabilities could have serious consequences.

[1]DDS is an Object Management Group (OMG) standard for real-time communication in distributed systems, providing middleware services for high-performance and scalable data exchanges.

[2]ROS 2 security features are implemented through the SROS2 (Secure ROS 2) framework, which leverages DDS Security plugins for end-to-end security.

Multi-platform Support: ROS 2 supports multiple operating systems, including Linux, Windows, and macOS, as well as real-time operating systems and embedded platforms[3]. This flexibility is crucial for edge computing deployments where diverse hardware platforms may be involved.

Distributed Computing: ROS 2 was designed from the ground up to support distributed computing scenarios[4], making it well-suited for edge computing applications where processing may be distributed across multiple physical locations.

3.1.2 Communication Patterns in ROS 2

ROS 2 supports several communication patterns that are relevant to edge computing implementations:

Publish-Subscribe: The primary communication mechanism in ROS 2, where nodes can publish messages to topics and subscribe to receive messages from topics of interest. This decoupled communication pattern is ideal for distributed systems where producers and consumers of data may be located on different machines.

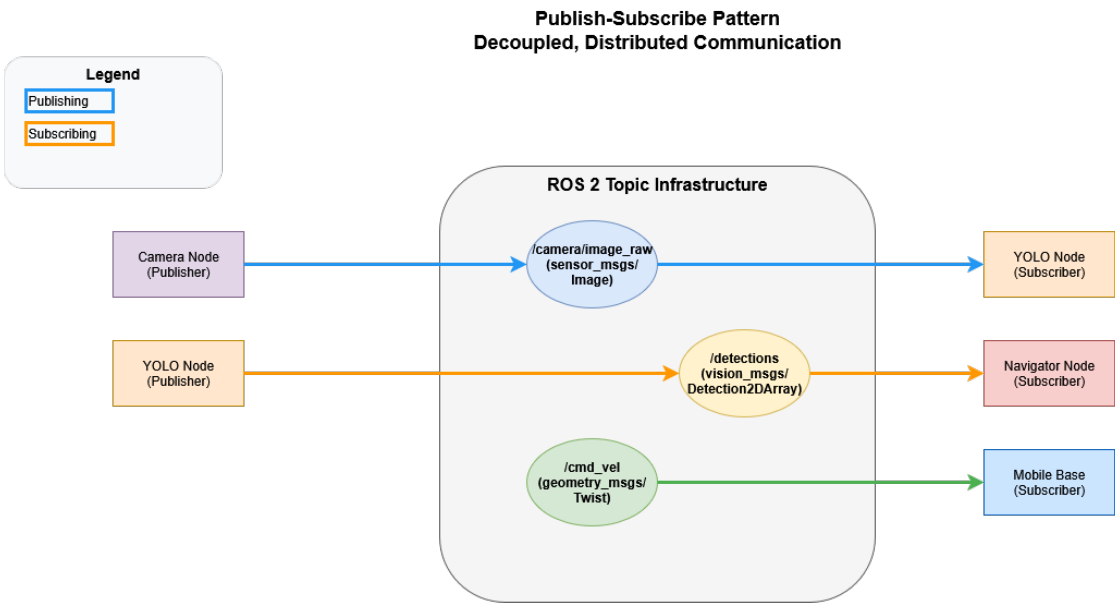

Publish-Subscribe Pattern
Decoupled, Distributed Communication
Legend
Publishing
Subscribing
ROS 2 Topic Infrastructure
Camera Node (Publisher)
/camera/image_raw (sensor_msgs/ Image)
YOLO Node (Subscriber)
YOLO Node (Publisher)
/detections (vision_msgs/ Detection2DArray)
Navigator Node (Subscriber)
/cmd_vel (geometry_msgs/ Twist)
Mobile Base (Subscriber)

Figure 3.1: Publish-Subscribe Communication Pattern in ROS 2

Service Calls: Synchronous request-response communication pattern suitable for operations that require immediate responses, such as configuration changes or status queries.

[3]Cross-platform support in ROS 2 is achieved through careful abstraction of platform-specific functionality and extensive use of portable libraries.

[4]The distributed architecture of ROS 2 eliminates the single point of failure inherent in ROS 1's master node design, enabling more robust multi-machine deployments.

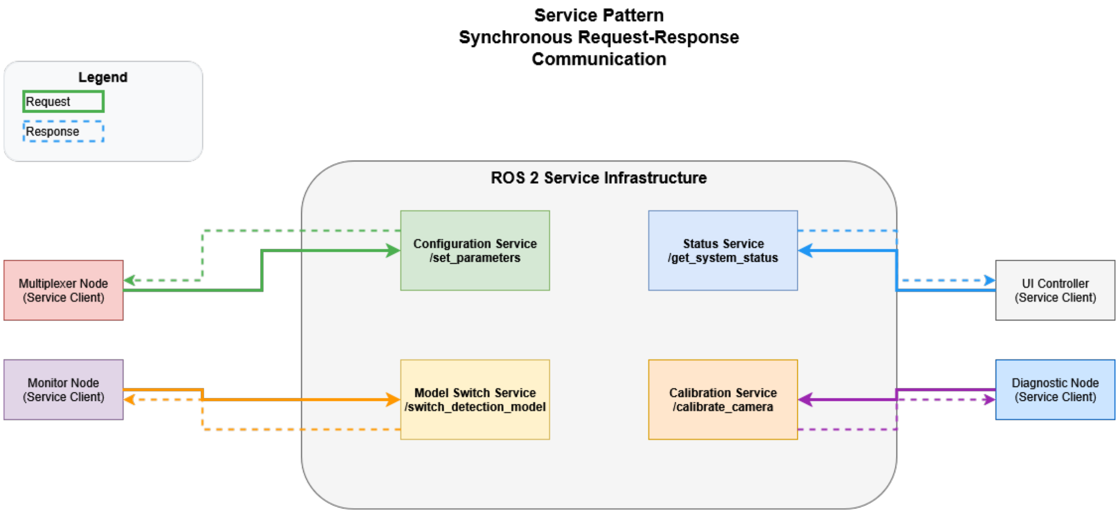
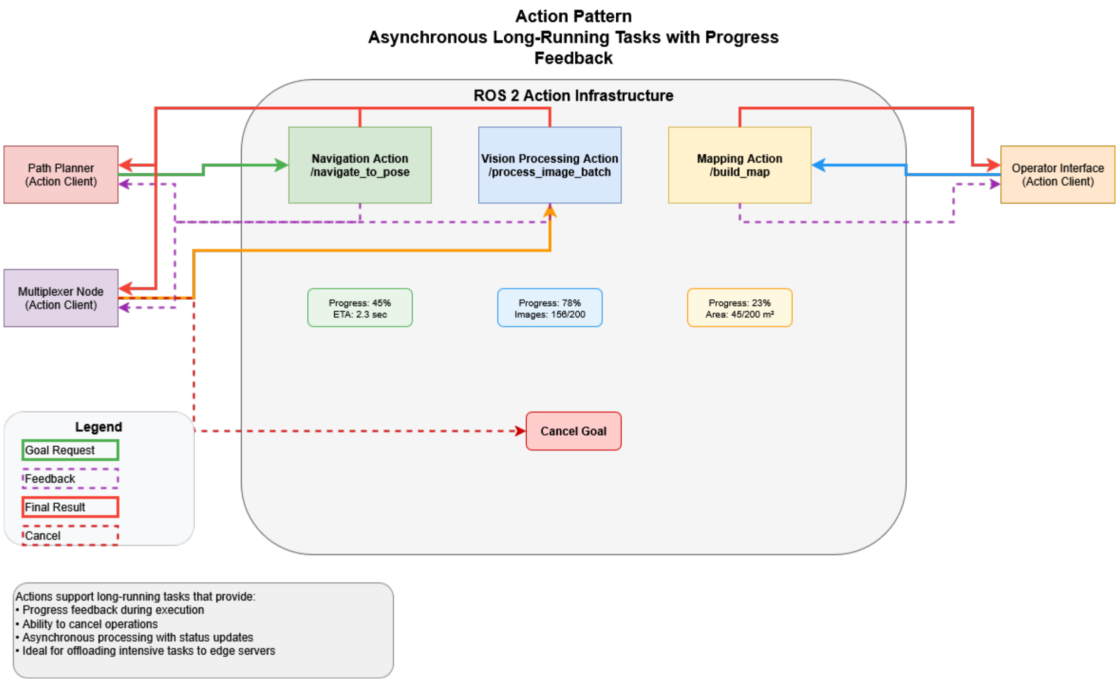


Figure 3.2: Service Call Communication Pattern in ROS 2

Actions: Asynchronous communication pattern for long-running tasks that provides progress feedback and the ability to cancel operations. This is particularly useful for offloading computationally intensive tasks to edge servers.



Actions support long-running tasks that provide:
- Progress feedback during execution
- Ability to cancel operations
- Asynchronous processing with status updates
- Ideal for offloading intensive tasks to edge servers

Figure 3.3: Action Communication Pattern in ROS 2

3.1.3 Lifecycle Management

ROS 2 introduces a standardized Lifecycle management system that allows nodes to transition through well-defined states (inactive, active, finalized) in a controlled manner. This capability is crucial for implementing dynamic task allocation in edge computing scenarios, as it enables controlled switching between local and remote processing modes without disrupting system operation.

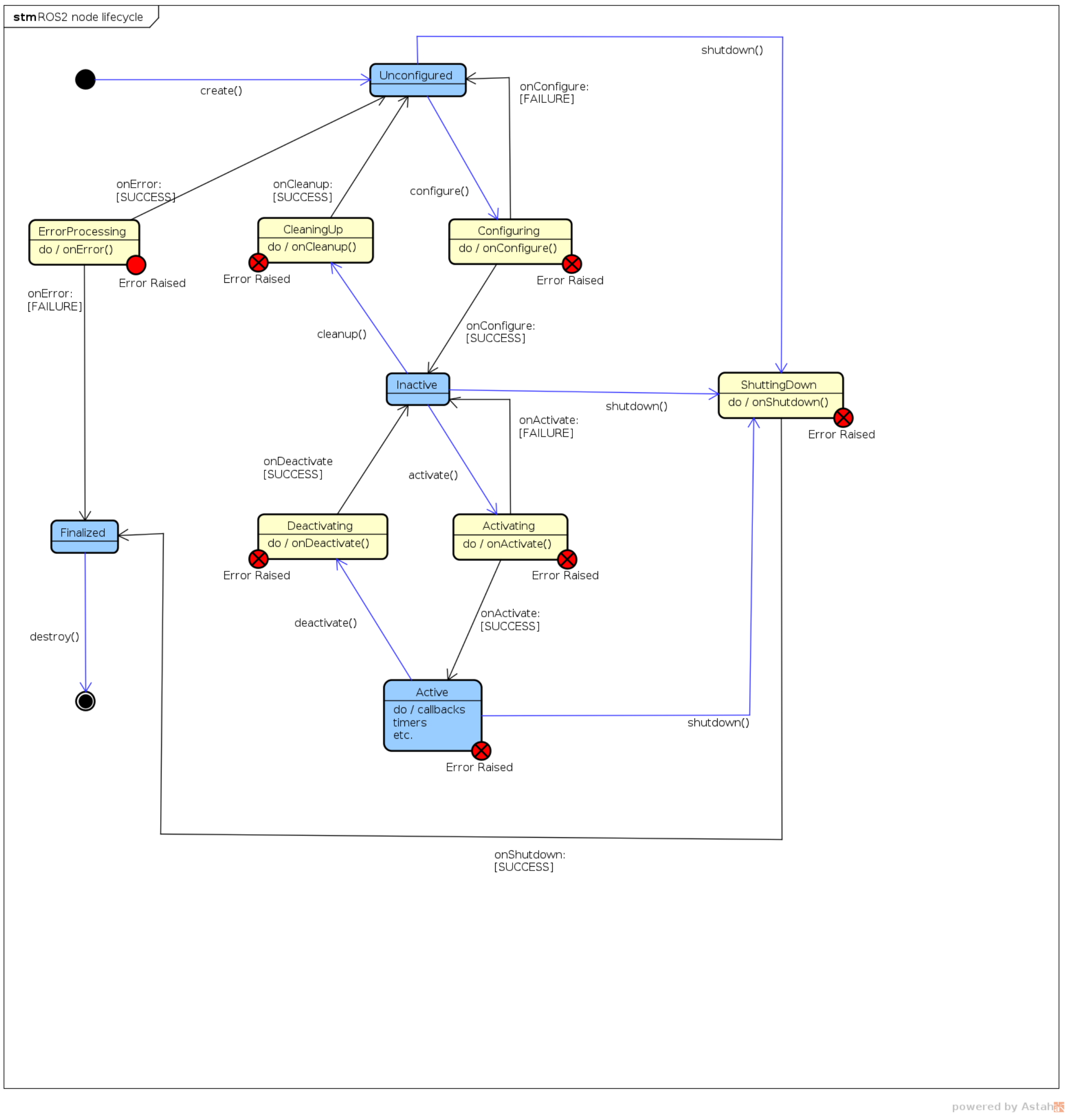


Figure 3.4: Lifecycle Management in ROS 2

The Lifecycle management system provides:

- Controlled startup and shutdown sequences

- State transition callbacks for implementing custom behavior
- External control of node states through service calls
- Monitoring and health checking capabilities

This infrastructure forms the foundation for the dynamic task allocation framework developed in this thesis.

3.2 Edge Computing Fundamentals

Edge computing represents a paradigm shift from centralized cloud computing toward distributed processing at the network edge. This section examines the fundamental concepts, motivations, and characteristics that make edge computing particularly relevant for mobile robotics applications.

3.2.1 Defining Edge Computing

Edge computing brings computation and data storage closer to the location where it is needed, to improve response times and save bandwidth. In the context of mobile robotics, edge computing typically involves deploying computational resources at network access points, base stations, or dedicated edge servers located in close proximity to robotic deployments.

Key characteristics of edge computing include:

Proximity: Edge resources are geographically distributed and located close to end devices, reducing network latency compared to centralized cloud services.

Low Latency: By processing data locally, edge computing can achieve sub-millisecond to low-millisecond response times suitable for real-time applications.

Bandwidth Efficiency: Edge computing reduces the need to transmit large amounts of raw data over wide-area networks by performing initial processing locally.

Autonomy: Edge systems can operate independently of central cloud services, providing resilience against network failures and enabling offline operation.

3.2.2 Edge Computing vs. Cloud Computing

Cloud computing has been the dominant paradigm for delivering computational resources over the internet, providing on-demand access to a shared pool of configurable computing resources. It enables users to provision and scale resources dynamically, without the need for physical hardware management. However, cloud computing often involves significant latency due to the physical distance between end devices and centralized data centers. The key differences include:

Latency: Cloud computing typically involves round-trip times of 50-100ms or more, while edge computing can achieve single-digit millisecond latencies.

Bandwidth Requirements: Cloud computing requires high-bandwidth connections to transmit data to distant servers, while edge computing can process data locally and transmit only results.

Scalability: Cloud computing provides virtually unlimited scalability, while edge resources are constrained by local computational capacity.

Cost Model: Cloud computing typically uses pay-per-use models, while edge computing may involve higher upfront infrastructure costs but lower operational costs for bandwidth-intensive applications.

3.2.3 Edge Computing Architectures

Several architectural patterns have emerged for edge computing deployments:

Device Edge: Processing occurs directly on end devices, such as smartphones or IoT devices. This provides the lowest latency but is limited by device computational capabilities.

Network Edge: Processing occurs at network infrastructure points, such as cellular base stations or WiFi access points. This balances latency with computational capability.

Regional Edge: Processing occurs at regional data centers that are geographically distributed but still aggregated to achieve economies of scale.

Hierarchical Edge: Multi-tier architectures that combine multiple edge layers, allowing for dynamic task allocation based on computational requirements and latency constraints.

For mobile robotics applications, network edge and hierarchical edge architectures are most relevant, as they provide the computational resources needed for complex AI workloads while maintaining low latency.

3.2.4 Quality of Service (QoS) Considerations

Edge computing systems must provide QoS guarantees to support real-time applications. Key QoS parameters include:

Latency: The time required to process a request and return results.

Throughput: The number of requests that can be processed per unit time.

Reliability: The probability that a request will be processed successfully within specified time constraints.

Availability: The fraction of time that the service is operational and accessible.

These QoS parameters are particularly critical for safety-critical robotics applications where failures can have serious consequences.

3.3 Computer Vision in Robotics

Computer vision has become increasingly important in robotics as sensors have improved and computational capabilities have expanded. This section examines the role of computer vision in robotics and focuses on object detection algorithms, particularly YOLO, which serves as the primary workload in our experimental evaluation.

3.3.1 Role of Computer Vision in Robotics

Computer vision enables robots to perceive and understand their environment through visual sensors. Key applications include:

Navigation and Obstacle Avoidance: Visual odometry, SLAM (Simultaneous Localization and Mapping), and real-time obstacle detection enable autonomous navigation in complex environments.

Object Recognition and Manipulation: Identifying and localizing objects enables robots to interact with their environment through manipulation tasks.

Human-Robot Interaction: Face detection, gesture recognition, and activity recognition enable natural interaction between humans and robots.

Quality Control and Inspection: Automated visual inspection enables robots to perform quality control tasks in manufacturing and other domains.

Safety and Security: Anomaly detection and surveillance capabilities enable robots to monitor environments for safety and security purposes.

3.3.2 Computational Challenges in Robotic Computer Vision

Computer vision algorithms, particularly those based on deep learning, present significant computational challenges for mobile robots:

Computational Intensity: Modern computer vision algorithms, especially convolutional neural networks (CNNs), require substantial computational resources, including high memory bandwidth and parallel processing capabilities.

Real-time Constraints: Many robotics applications require real-time processing of visual data, with frame rates of 30 FPS or higher. This constrains the computational complexity of algorithms that can be deployed.

Power Consumption: The computational requirements of computer vision algorithms directly impact power consumption, which is a critical constraint for battery-powered mobile robots.

Model Size and Memory Requirements: Large neural network models may exceed the memory capacity of mobile hardware, requiring model compression or distributed processing approaches.

3.3.3 YOLO: You Only Look Once

YOLO (You Only Look Once) is a state-of-the-art object detection algorithm that frames object detection as a single regression problem, directly predicting bounding boxes and class probabilities from full images in one evaluation. This approach offers several advantages for robotics applications:

Speed: YOLO can process images in real-time, making it suitable for applications requiring immediate responses to visual stimuli.

Global Context: Unlike sliding window approaches, YOLO sees the entire image during training and test time, enabling it to encode contextual information about classes and their appearance.

Generalization: YOLO generalizes well to new domains and is less likely to fail on unexpected inputs compared to other detection systems.

Unified Architecture: The single network architecture simplifies deployment and reduces the complexity of the overall system.

YOLO Architecture and Variants

The YOLO family has evolved through several versions, each improving accuracy and efficiency:

YOLOv1: The original YOLO architecture divided images into a grid and predicted bounding boxes and class probabilities for each grid cell.

YOLOv2/YOLO9000: Introduced batch normalization, anchor boxes, and multi-scale training to improve accuracy while maintaining speed.

YOLOv3: Used a more sophisticated feature extractor (Darknet-53) and multi-scale predictions to improve detection of small objects.

YOLOv4/YOLOv5: Incorporated various optimization techniques, including CSPNet backbones, PANet path aggregation, and improved training strategies.

YOLOv8: The latest version offering improved accuracy and efficiency through architectural improvements and training optimizations.

3.3.4 Stop Sign Detection as a Representative Workload

Stop sign detection was chosen as the primary workload for this thesis due to several characteristics that make it representative of robotics computer vision tasks:

Safety Criticality: Stop sign detection is fundamental to autonomous navigation and safety systems, making reliability and performance critical requirements.

Real-time Requirements: Effective stop sign detection must operate with minimal latency to be useful for navigation and collision avoidance.

Computational Complexity: YOLO-based stop sign detection requires substantial computational resources, making it an appropriate test case for evaluating edge computing trade-offs.

Variable Complexity: Detection complexity varies with scene conditions, lighting, and image resolution, providing opportunities to study adaptive resource allocation.

Well-defined Ground Truth: Stop signs have standardized visual characteristics, making it possible to establish clear performance metrics and evaluation criteria.

3.4 State of the Art Review

This section reviews existing research at the intersection of edge computing and mobile robotics, identifying current approaches, limitations, and opportunities for advancement.

3.4.1 Edge Computing for Mobile Robotics

Several research efforts have explored the application of edge computing to mobile robotics:

Offloading Strategies: Researchers have proposed various strategies for deciding when and how to offload computational tasks from robots to edge servers. These include threshold-based approaches that consider network latency and computational load, machine learning-based approaches that predict optimal offloading decisions, and hybrid approaches that combine multiple factors.

Task Partitioning: Some work has focused on partitioning complex computational tasks between local and remote processing. This includes approaches that partition neural network inference across multiple devices and methods that dynamically adjust the granularity of task partitioning based on network conditions.

Multi-Robot Coordination: Research has examined how edge computing can facilitate coordination among multiple robots by providing shared computational resources and communication infrastructure.

3.4.2 Distributed Computing in ROS

The ROS community has developed several approaches for distributed computing:

Multi-machine ROS: Traditional approaches for running ROS across multiple machines, typically involving manual configuration of network connections and node distributions.

ROS Industrial: Extensions to ROS for industrial applications that include distributed processing capabilities and integration with cloud services.

Kubernetes Integration: Recent work has explored running ROS applications in Kubernetes clusters, enabling automatic scaling and load balancing of robotic workloads.

3.4.3 Computer Vision Offloading

Specific research on offloading computer vision tasks includes:

Adaptive Model Selection: Approaches that dynamically select between different neural network models based on available computational resources and accuracy requirements.

Progressive Inference: Methods that perform initial processing locally and refine results remotely, reducing latency for time-critical decisions while improving accuracy when time permits.

Collaborative Inference: Approaches where multiple robots collaborate to improve computer vision performance through shared processing and data fusion.

3.4.4 Energy-Aware Computing

Research on energy efficiency in mobile robotics includes:

Dynamic Voltage and Frequency Scaling: Techniques for adjusting processor performance based on computational requirements to minimize energy consumption.

Task Scheduling: Algorithms for scheduling computational tasks to minimize energy consumption while meeting real-time constraints.

Energy-Aware Offloading: Approaches that consider energy consumption when making offloading decisions, balancing local processing costs against communication and remote processing costs.

3.4.5 Gaps in Current Research

Despite significant progress, several gaps remain in current research:

Comprehensive Evaluation: Most existing work evaluates systems under limited conditions or with simplified workloads. There is a need for comprehensive evaluation across realistic deployment scenarios[7, 4].

Dynamic Adaptation: While many systems propose adaptive behavior, few provide detailed evaluation of how well these systems adapt to changing conditions in practice[3].

Real-world Validation: Much of the existing research relies on simulation or simplified experimental setups. There is a need for validation using actual robotic hardware and realistic workloads[8, 10].

Energy Analysis: Few studies provide detailed analysis of energy consumption patterns and their implications for battery-powered mobile robots[1, 7].

Integration Challenges: Limited attention has been paid to the practical challenges of integrating edge computing capabilities into existing robotic systems[3, 2].

3.5 Comparative Analysis

This section provides a systematic comparison of different computing paradigms for mobile robotics, establishing the context for our edge computing approach.

3.5.1 Onboard Computing

Onboard computing involves performing all computational tasks using resources available on the robot itself.

Advantages:

- Zero network latency for computational tasks
- Complete autonomy and independence from external infrastructure
- Predictable performance characteristics

- No communication security concerns for computation
- Offline operation capability

Disadvantages:

- Limited computational resources constrained by power, weight, and thermal considerations
- High hardware costs for capable computing platforms
- Limited scalability as computational requirements increase
- Difficulty upgrading computational capabilities
- Energy consumption directly impacts operational range

Suitable Applications: Simple navigation tasks, basic sensor processing, safety-critical functions requiring guaranteed response times.

3.5.2 Cloud Computing

Cloud computing involves offloading computational tasks to remote data centers accessed over the internet.

Advantages:

- Virtually unlimited computational resources
- Cost-effective pay-per-use pricing models
- Access to specialized hardware (GPUs, TPUs) for AI workloads
- Automatic scaling based on demand
- Regular updates and maintenance handled by providers

Disadvantages:

- High network latency (typically 50-100ms + round-trip)
- Dependence on reliable internet connectivity
- High bandwidth requirements for data-intensive tasks
- Security and privacy concerns for sensitive data
- Variable performance due to shared infrastructure

Suitable Applications: Non-time-critical data analysis, long-term learning and adaptation, complex optimization problems.

3.5.3 Edge Computing

Edge computing provides a middle ground between onboard and cloud computing by placing computational resources at the network edge.

Advantages:

- Low latency (single-digit milliseconds possible)
- Reduced bandwidth requirements compared to cloud computing
- Better reliability than cloud computing for local operations
- Scalable computational resources beyond onboard limitations
- Potential for local data processing to address privacy concerns

Disadvantages:

- Limited computational resources compared to cloud computing
- Requires edge infrastructure deployment and maintenance
- Still dependent on network connectivity
- More complex system architecture than purely onboard solutions
- Variable resource availability based on location

Suitable Applications: Real-time computer vision, sensor fusion, adaptive behavior requiring moderate computational resources.

3.5.4 Hybrid Approaches

Hybrid approaches combine multiple computing paradigms to optimize for different requirements and conditions.

Dynamic Task Allocation: Systems that can dynamically choose between onboard, edge, and cloud processing based on current conditions, computational requirements, and performance objectives.

Hierarchical Processing: Architectures that use multiple processing tiers, with initial processing onboard, refinement at the edge, and complex analysis in the cloud.

Graceful Degradation: Systems designed to maintain functionality across different levels of computational availability, from onboard-only operation to full cloud integration.

3.5.5 Performance Comparison Framework

To systematically compare these approaches, we consider several key performance dimensions:

Latency: The time required to complete computational tasks, including network communication overhead.

Throughput: The number of computational tasks that can be completed per unit time.

Energy Efficiency: The energy required to complete computational tasks, including both processing and communication energy.

Reliability: The probability that computational tasks will be completed successfully within specified time constraints.

Cost: The financial cost associated with computational resources, including hardware, infrastructure, and operational costs.

Scalability: The ability to handle increasing computational loads without proportional increases in cost or decreases in performance.

The choice between these approaches depends on the specific requirements of the robotics application, including real-time constraints, computational complexity, energy budgets, and deployment environments. The edge computing approach investigated in this thesis aims to provide an optimal balance for applications requiring real-time computer vision processing with computational demands that exceed onboard capabilities.

This background establishes the foundation for understanding the design decisions and trade-offs involved in implementing edge computing solutions for mobile robotics, which will be detailed in subsequent chapters.

3.6 Container Orchestration and Network Federation

Modern robotics deployments increasingly leverage container orchestration and network federation technologies to manage distributed computational workloads. This section examines Kubernetes as a foundation for robotics applications and Liqo as an enabling technology for seamless resource federation across edge computing environments.

3.6.1 Kubernetes for Robotics Applications

Kubernetes is an open-source container orchestration platform that automates deployment, scaling, and management of containerized applications. For robotics applications, Kubernetes provides several key benefits:

Application Isolation: Containers provide process isolation and dependency management, enabling multiple ROS 2 nodes to run independently without conflicts over shared libraries or system resources.

Resource Management: Kubernetes enables fine-grained control over CPU, memory, and other resource allocations for individual components, crucial for managing computational resources on resource-constrained robotic platforms.

Service Discovery: Kubernetes networking abstractions simplify inter-node communication in distributed ROS 2 systems by providing consistent service endpoints regardless of underlying infrastructure.

Declarative Configuration: Kubernetes manifests enable reproducible deployments and version-controlled system configurations, essential for managing complex robotic software stacks.

Health Monitoring: Automatic health checking and restart capabilities ensure system resilience, particularly important for autonomous robotic operations.

ROS 2 Node Containerization

Running ROS 2 nodes as containerized applications presents both opportunities and challenges:

Benefits:

- Dependency isolation prevents conflicts between different ROS 2 packages
- Reproducible deployments ensure consistent behavior across different environments
- Version control enables controlled rollbacks and incremental updates
- Resource limits prevent individual nodes from consuming excessive system resources

Challenges:

- DDS discovery across container networks requires careful network configuration
- Shared memory communication patterns may require special container privileges
- Hardware access (cameras, sensors) needs explicit device mounting
- Real-time performance may be affected by container overhead

3.6.2 K3s: Lightweight Kubernetes for Edge Deployments

K3s is a lightweight Kubernetes distribution designed for edge computing and resource-constrained environments. Key characteristics that make it suitable for robotics applications include:

Reduced Resource Footprint: K3s requires significantly less memory and storage than full Kubernetes distributions, making it viable for deployment on robotic hardware with limited computational resources.

Simplified Installation: Single binary distribution and automated installation scripts reduce deployment complexity in field environments.

Built-in Components: Includes ingress controller, service load balancer, and other essential components out-of-the-box, reducing configuration overhead.

ARM64 Support: Native support for ARM64 processors commonly used in robotic platforms enables consistent deployment across different hardware architectures.

Edge-Optimized Networking: Designed for environments with intermittent connectivity and variable network conditions typical in mobile robotics deployments.

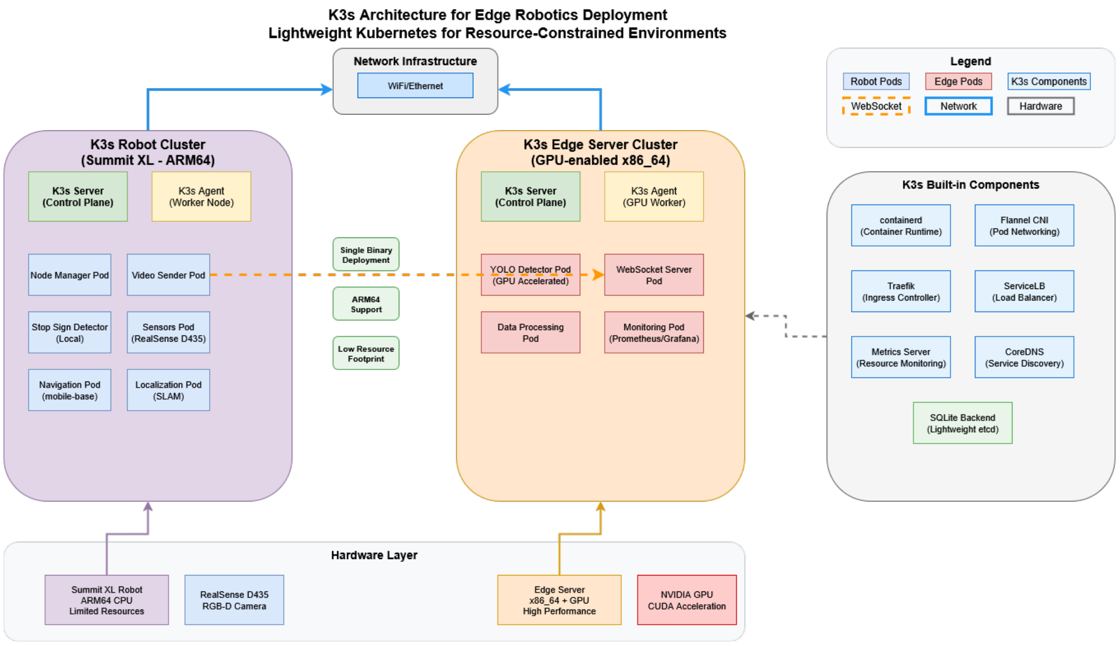

K3s Architecture for Edge Robotics Deployment
Lightweight Kubernetes for Resource-Constrained Environments
Network Infrastructure
WiFi/Ethernet
Legend
Robot Pods
Edge Pods
K3s Components
WebSocket
Network
Hardware
K3s Robot Cluster (Summit XL - ARM64)
K3s Server (Control Plane)
K3s Agent (Worker Node)
Node Manager Pod
Video Sender Pod
Stop Sign Detector (Local)
Sensors Pod (RealSense D435)
Navigation Pod (mobile-base)
Localization Pod (SLAM)
Single Binary Deployment
ARM64 Support
Low Resource Footprint
K3s Edge Server Cluster (GPU-enabled x86_64)
K3s Server (Control Plane)
K3s Agent (GPU Worker)
YOLO Detector Pod (GPU Accelerated)
WebSocket Server Pod
Data Processing Pod
Monitoring Pod (Prometheus/Grafana)
K3s Built-in Components
containerd (Container Runtime)
Flannel CNI (Pod Networking)
Traefik (Ingress Controller)
ServiceLB (Load Balancer)
Metrics Server (Resource Monitoring)
CoreDNS (Service Discovery)
SQLite Backend (Lightweight etcd)
Hardware Layer
Summit XL Robot ARM64 CPU Limited Resources
RealSense D435 RGB-D Camera
Edge Server x86_64 + GPU High Performance
NVIDIA GPU CUDA Acceleration

Figure 3.5: K3s Architecture for Edge Robotics Deployment

3.6.3 Liqo: Cross-Cluster Resource Federation

Liqo is an open-source platform that enables resource and network federation across Kubernetes clusters. For edge computing in mobile robotics, Liqo addresses several critical challenges:

Seamless Multi-Cluster Networking: Liqo creates secure network tunnels between clusters without requiring complex VPN configurations or network infrastructure changes.

Transparent Resource Federation: Remote computational resources appear as local nodes in the Kubernetes cluster, enabling transparent pod scheduling across cluster boundaries.

Dynamic Peering: Clusters can automatically discover and establish peering relationships with edge servers, enabling opportunistic resource utilization as robots move through different environments.

Service Discovery: Cross-cluster service discovery enables ROS 2 nodes running on different clusters to communicate transparently, regardless of their physical location.

Liqo Peering Architecture

Liqo implements cross-cluster federation through several key components:

Control Plane Replication: Remote cluster resources are advertised through control plane replication, enabling the local scheduler to consider remote resources for pod placement decisions.

Network Manager: Establishes and manages secure network tunnels between clusters, handling automatic route configuration and traffic encryption.

Virtual Kubelet: Presents remote cluster resources as virtual nodes in the local cluster, abstracting the complexity of cross-cluster resource management from the Kubernetes scheduler.

Identity Management: Handles authentication and authorization across cluster boundaries while maintaining security isolation between different administrative domains.

3.6.4 Dynamic Task Offloading Implementation

The combination of Kubernetes and Liqo enables sophisticated dynamic task offloading strategies:

Intelligent Pod Placement: The Kubernetes scheduler can place pods on remote edge resources when local computational capacity is insufficient or when network conditions favor remote processing.

Connectivity-Aware Decisions: Applications can monitor Liqo peering status to make informed decisions about local versus remote processing, ensuring graceful degradation when network connectivity is lost.

Real-time Communication: WebSocket connections enable low-latency communication between robots and offloaded processing tasks, essential for computer vision and other real-time applications.

Lifecycle Management: ROS 2 Lifecycle management integrates with Kubernetes pod Lifecycle to enable controlled transitions between different processing modes without disrupting overall system operation.

Implementation Pattern for ROS 2 Offloading

The typical implementation pattern for dynamic offloading involves:

Cluster State Monitoring: A node manager component continuously monitors Liqo peering status and available remote resources to make offloading decisions.

Conditional Node Activation: Based on connectivity and resource availability, the system selectively activates either local processing nodes or remote communication nodes (e.g., video senders).

Stream-based Communication: Real-time data streams (such as camera feeds) are transmitted via WebSocket connections to remote processing pods, enabling low-latency computer vision processing.

Result Integration: Processing results from remote edge servers are integrated back into the local ROS 2 system through standard topic publication mechanisms.

Automatic Failover: When connectivity is lost or remote resources become unavailable, the system automatically falls back to local processing without manual intervention.

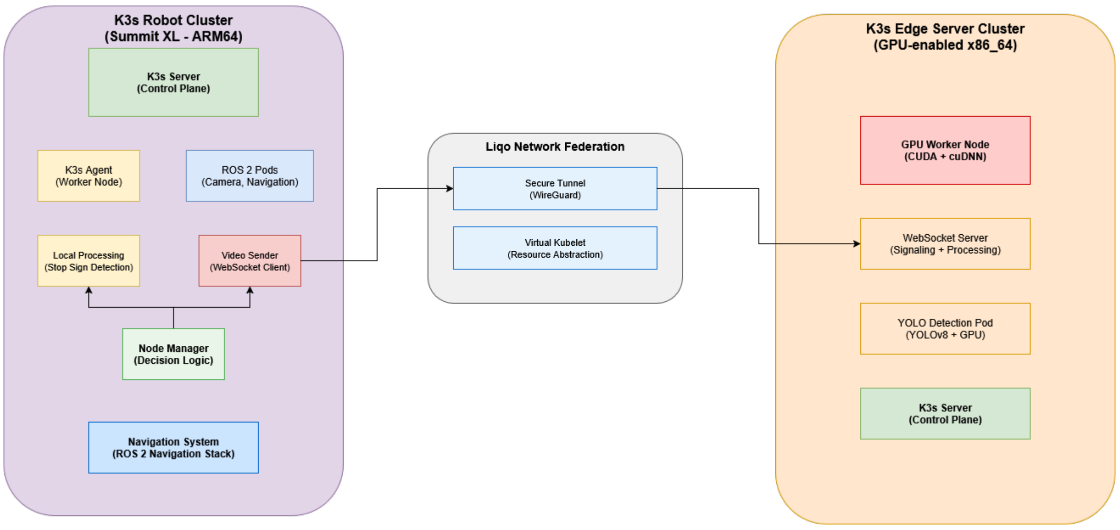


Figure 3.6: ROS 2 Dynamic Offloading Pattern with Kubernetes and Liqo

3.6.5 Benefits for Mobile Robotics

The integration of Kubernetes and Liqo provides several advantages specifically relevant to mobile robotics:

Infrastructure Abstraction: Robots can access computational resources uniformly regardless of whether they are local or remote, simplifying application development and deployment.

Dynamic Scalability: Computational resources can be allocated dynamically based on current demands, enabling efficient resource utilization across multiple robots and edge servers.

Mobility Support: As robots move between different network coverage areas, Liqo can automatically establish new peering relationships with available edge servers, maintaining computational capabilities.

Fault Tolerance: Automatic pod rescheduling and service recovery capabilities ensure that critical robotics applications remain operational despite individual component failures.

Operational Simplicity: Standard Kubernetes tooling and practices can be used for managing complex distributed robotics deployments, reducing operational overhead and enabling DevOps best practices.

This container orchestration and network federation foundation enables the sophisticated edge computing capabilities explored in subsequent chapters, providing the infrastructure necessary for dynamic task allocation and seamless resource federation in mobile robotics applications.

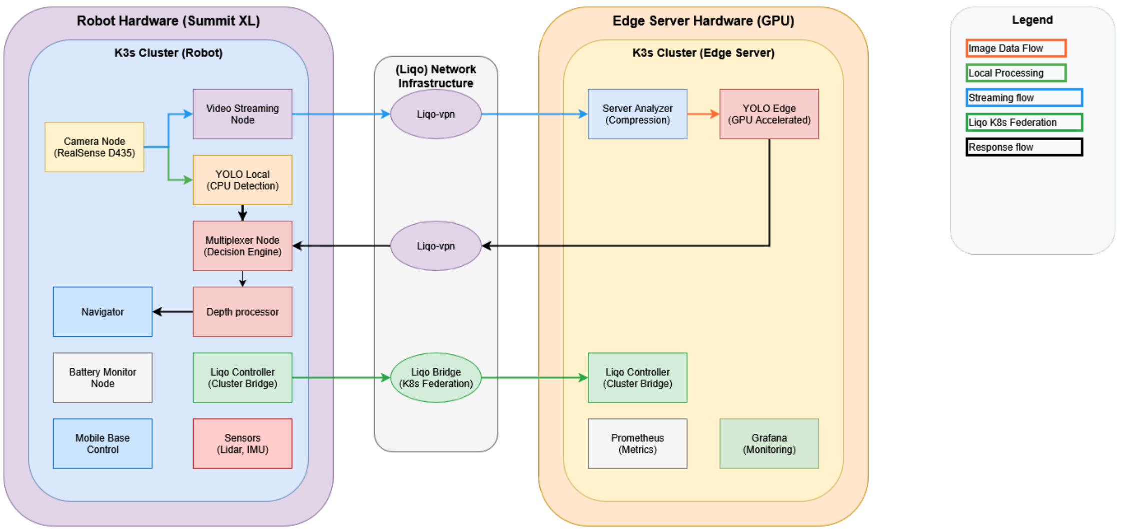
Chapter 4

System Design

This chapter provides a comprehensive overview of the architecture and design decisions for the framework presented in this thesis.

4.1 Architectural Overview

The system follows a distributed three-tier architecture consisting of mobile robot hardware, edge server infrastructure, and network connectivity layers. Figure 4.1 illustrates the complete system design and component interactions.



Robot Hardware (Summit XL)
K3s Cluster (Robot)
Camera Node (RealSense D435)
Video Streaming Node
YOLO Local (CPU Detection)
Multiplexer Node (Decision Engine)
Navigator
Depth processor
Battery Monitor Node
Liqo Controller (Cluster Bridge)
Mobile Base Control
Sensors (Lidar, IMU)
(Liqo) Network Infrastructure
Liqo-vpn
Liqo-vpn
Liqo Bridge (K8s Federation)
Edge Server Hardware (GPU)
K3s Cluster (Edge Server)
Server Analyzer (Compression)
YOLO Edge (GPU Accelerated)
Liqo Controller (Cluster Bridge)
Prometheus (Metrics)
Grafana (Monitoring)
Legend
Image Data Flow
Local Processing
Streaming flow
Liqo K8s Federation
Response flow

Figure 4.1: Complete System Architecture showing robot hardware, edge server components, and network infrastructure with data flow patterns

The architecture is designed around key principles of modularity, scalability, and fault tolerance. It enables seamless integration of components and dynamic task allocation based on real-time conditions and performance metrics. The three-tier design separates the mobile robot, edge server, and monitoring infrastructure, allowing for independent scaling and optimization of each layer. This section will provide an in-depth overview of the system architecture, including data and control flow patterns, integration points with existing ROS 2 ecosystems, and considerations for real-time performance and security in a distributed environment.

4.2 Technology Selection Rationale

Technology selection was driven by performance requirements, integration compatibility, and future extensibility.

ROS 2: was left as the primary middleware choice due to its robust features and active community support.

Python: was chosen for its ease of use and vast documentation in machine learning and therefore object recognition capabilities.

YOLO: was selected for its real-time processing capabilities and optimal balance between accuracy and computational efficiency.

K8s: was chosen for its advanced capabilities for load balancing, health monitoring, and automatic restart of containers, ensuring the resilience and scalability of the deployed applications.

Liqo: was chosen for its ability to create a virtual cluster across different networks, which allowed for the use of fastdds.

WebSocket: was selected for its lightweight and efficient communication capabilities, making it suitable for real-time data exchange between the robot and edge server.

4.2.1 ROS 2 for Robotics Middleware

ROS 2 was selected as the primary middleware due to its distributed computing capabilities and real-time features. Its integration of DDS provides built-in security and reliable communication, essential for the project's needs. The middleware's support for multi-platform deployment and native distributed computing were also significant factors in its selection. Compared to ROS 1, ROS 2 offers enhanced real-time capabilities and Quality of Service (QoS) guarantees, making it better suited for the dynamic and performance-critical environment of mobile robotics. Additionally, ROS 2's Lifecycle management features facilitate dynamic task switching, a core requirement of the proposed framework. The decision to choose ROS 2 over other robotics frameworks was further reinforced by its active community support and the maturity of its ecosystem.

4.2.2 YOLO for Object Detection

YOLO was chosen for its real-time processing capabilities and optimal balance between accuracy and computational efficiency. The single-stage detection approach of YOLO allows for faster inference times, which is critical for real-time applications in mobile robotics. YOLOv8, the selected version, offers improved accuracy and speed, making it suitable for the project's requirements. The decision was also influenced by the need for GPU acceleration support and considerations for model size to ensure efficient deployment on mobile platforms. Alternative detection algorithms were considered, but YOLO's advantages in terms of speed and efficiency made it the preferred choice. The capability to perform stop sign detection, a key functionality for the robotic system, was effectively demonstrated using the YOLO framework. Extensive performance benchmarking confirmed YOLO's superiority in meeting the project's real-time processing demands. Furthermore, the model was optimized for edge deployment, ensuring efficient operation within the constraints of the robotic hardware.

4.2.3 Node Manager Design

The multiplexer serves as the central decision-making component for dynamic task allocation. It intelligently selects between different processing algorithms and resources based on real-time evaluation of system and network conditions. The design incorporates strategy patterns to allow flexible and efficient decision-making. Key criteria for decision-making include network latency, system resource utilization, quality of service requirements, and power consumption considerations. The multiplexer also handles state management, transition handling, and provides fallback mechanisms and error recovery to ensure robust operation. Its integration with ROS 2 Lifecycle management enables seamless coordination with other system components. Performance monitoring and metrics collection are integral to the multiplexer design, facilitating continuous optimization of task allocation decisions. The design ensures thread safety and concurrent operation handling to maintain system stability and performance.

4.3 Communication Architecture

The communication architecture prioritizes low latency and reliability for real-time robotics applications. It defines the protocols, message formats, and error handling mechanisms necessary to ensure efficient and secure data exchange between the mobile robot and edge server. The selected communication protocols and message designs are optimized for the unique requirements of mobile robotics, including real-time performance, bandwidth efficiency, and robust error handling.

4.3.1 Network Communication Protocols

WebSocket was selected for its bidirectional communication capabilities and low latency characteristics. It provides a persistent connection that significantly reduces the overhead associated with establishing new connections, as required in HTTP REST APIs. This is particularly advantageous for the frequent and real-time communication needs of the robotic system. WebSocket also offers better support for streaming data, which is critical for transmitting video and sensor data from the robot to the edge server. Alternative protocols such as gRPC and Zenoh were evaluated, but WebSocket's advantages in terms of real-time performance and ease of integration with existing web technologies made it the preferred choice. The analysis of protocol overhead and performance implications confirmed WebSocket's suitability for the project. Connection management and reconnection strategies were also developed to ensure reliable communication even in the face of network interruptions.

4.3.2 Quality of Service and Reliability

Quality of service guarantees are critical for safety-critical robotics operations. The communication architecture includes comprehensive QoS requirements addressing connection reliability, message delivery priorities, and bandwidth management. Mechanisms for connection monitoring, health checking, timeout handling, and retries are implemented to maintain robust communication links. The system is designed to gracefully degrade in the event of network failures, ensuring continued operation under degraded conditions. Bandwidth adaptation and congestion control measures are also integrated to optimize communication performance in varying network conditions. End-to-end latency measurement and monitoring are employed to ensure that the communication system meets the stringent real-time requirements of the robotic applications.

4.4 Infrastructure Setup

Infrastructure design supports both development and production deployment scenarios. It encompasses the physical and network infrastructure requirements, hardware specifications, and the software stack necessary for deploying the robotic system. Consideration is given to the entire Lifecycle of the infrastructure, from initial setup through to ongoing management and maintenance.

4.4.1 Hardware Configuration

Hardware configuration balances computational capability with power consumption constraints. The mobile robot platform is equipped with processing units (CPU, GPU) and memory sufficient to handle the local inference and communication tasks. Consideration for power consumption and battery life is critical, influencing the selection of energy-efficient components and optimization of processing loads. Network connectivity options, including Ethernet and WiFi, are provided to ensure flexible and reliable communication with the edge server. Sensor integration capabilities are also included to support the necessary data collection for navigation and object detection.

The edge server hardware is configured to provide the necessary processing power for computer vision workloads, with GPU acceleration to support the intensive computational requirements of the YOLO inference. Network interface specifications are chosen to ensure high-speed data transfer between the edge server and the mobile robot. Adequate storage is provisioned for the models and data necessary for inference tasks.

Network infrastructure components, such as access points and routing equipment, are selected and configured to provide the required bandwidth and low latency communication between the mobile robot and the edge server. Consideration is given to the capacity planning and optimization of latency characteristics to ensure the network can handle the expected data loads and provide reliable real-time communication.

4.4.2 Network Topology and Configuration

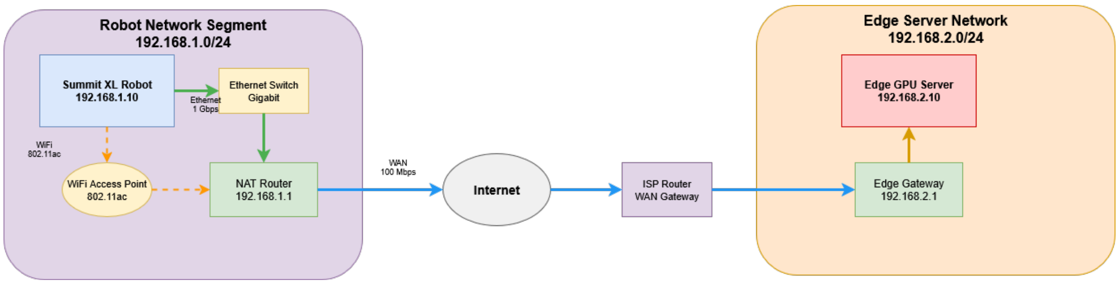


Figure 4.2: Network Topology Diagram

Network topology supports both Ethernet and WiFi connectivity options. The network architecture is designed to provide a robust and flexible communication framework that can accommodate the dynamic nature of mobile robotics. Local area network configurations are established to ensure high-speed communication between the mobile robot and edge server, with IP addressing and subnet planning to facilitate efficient data routing and management.

Quality of service configurations are implemented to prioritize critical data traffic and ensure the reliable delivery of time-sensitive information. Security policies and access control measures are also integrated into the network design to protect against unauthorized access and ensure the integrity of the data being transmitted.

Consideration is given to the deployment of Ethernet versus WiFi, with evaluations of performance characteristics, mobility requirements, infrastructure complexity, cost, reliability, and potential interference. Network monitoring and management tools are incorporated to provide visibility into network performance and facilitate the rapid identification and resolution of any issues that may arise.

Failover and redundancy mechanisms are also designed into the network infrastructure to ensure continuous operation in the event of a component failure or loss of connectivity.

4.4.3 Software Deployment Architecture

Containerization provides deployment consistency and scalability. The software deployment architecture is centered around the use of containerization to package and deploy the various components of the robotic system. Docker is used as the containerization platform, providing isolation and management of dependencies for each component. This approach ensures that the software can be consistently deployed across different environments, from development to production, and simplifies the management of dependencies and configuration settings.

Container orchestration considerations are also addressed, with Kubernetes recommended for managing production deployments. Kubernetes provides advanced capabilities for service discovery, load balancing, health monitoring, and automatic restart of containers, ensuring the resilience and scalability of the deployed applications. Docker Compose is suggested for development and testing environments to facilitate easy setup and management of multi-container applications.

Configuration management and environment variables are used to manage the configuration settings for the various components, ensuring that they can be easily adapted to different deployment environments and requirements. Integration with logging and monitoring infrastructure is also planned to provide visibility into the operation of the deployed components and facilitate the detection and diagnosis of any issues that may occur.

4.5 System Components and Interactions

System components are designed for modularity and clear separation of responsibilities. Each component of the system is designed to perform a specific set of functions and responsibilities, with well-defined interfaces and interaction patterns to facilitate communication and coordination between components. This modular design approach enables easier development, testing, and maintenance of individual components, as well as greater flexibility and scalability in the overall system architecture.

4.5.1 Mobile Robot Components

The camera capture component is responsible for acquiring color images and depth images. It includes functionality for format conversion and compression to optimize the images for transmission over the network. Quality adjustment algorithms are applied based on current network conditions to ensure optimal image quality for processing. Timestamp annotation and metadata are added to the images to provide context and facilitate synchronization with other data sources.

The local YOLO inference engine is responsible for loading the YOLO model and performing object detection on the captured images. It is optimized for GPU acceleration to ensure real-time processing speeds and includes capabilities for batch processing of images to further improve efficiency. Performance monitoring and profiling are integrated to provide insights into the inference performance and facilitate optimization.

The Node Manager implements the decision logic for dynamic task allocation, using performance metrics to determine the optimal allocation of tasks. It manages the state and transitions of the task allocation process and collects metrics on performance for analysis and optimization.

The communication client manages the WebSocket connection to the edge server, handling the serialization and transmission of messages containing the captured images and detection results. It includes error handling and retry logic to ensure reliable communication and bandwidth monitoring and adaptation capabilities to optimize the use of available network resources.

4.5.2 Edge Server Components

Edge server components provide scalable remote processing capabilities. The edge server is equipped with a WebSocket server to handle incoming connections from multiple mobile robots, providing real-time bidirectional communication channels. It also hosts the remote YOLO inference service, which performs object detection on the images received from the mobile robots. The edge server is responsible for processing the detection results and generating the appropriate responses to be sent back to the robots.

The WebSocket server component is responsible for managing concurrent connections from multiple clients (mobile robots), including load balancing across available resources to ensure optimal performance. It monitors the state of each connection and manages the allocation and scheduling of resources for processing incoming requests.

The remote YOLO inference service is responsible for hosting and serving the YOLO model, processing incoming image data, and performing object detection. It includes capabilities for request queuing and batch processing to optimize the use of GPU resources and improve inference performance. Performance optimization and caching mechanisms are also integrated to further enhance the efficiency of the inference service.

The results processing component is responsible for formatting the detection results, applying confidence filtering and post-processing to refine the results, and optimizing the response time for sending the results back to the mobile robots. It also includes error handling and graceful failure mechanisms to ensure reliable operation.

4.5.3 Monitoring and Metrics Components

Monitoring components provide comprehensive system observability. The monitoring and metrics components are responsible for collecting, aggregating, and analyzing performance data from the various system components. This includes latency measurements, throughput monitoring, resource utilization metrics, and network performance metrics. The collected data is used for real-time monitoring and analysis, as well as for historical analysis and reporting.

The performance data collection component is responsible for measuring and reporting on key performance metrics, including end-to-end latency, network latency, processing latency, throughput, CPU and GPU utilization, memory usage, battery status, and network metrics such as bandwidth, packet loss, and jitter.

Data aggregation and storage are handled by integrating with a time-series database, which provides efficient storage and retrieval of the collected metrics data. Data retention policies are implemented to manage the Lifecycle of the stored data, and data visualization and dashboarding capabilities are provided to enable easy access to the monitoring data.

Alerting and anomaly detection mechanisms are integrated to monitor performance thresholds and automatically generate alerts in the event of potential issues or anomalies. System health indicators are also monitored to provide insights into the overall health and performance of the system. Predictive maintenance capabilities are included to enable proactive identification and resolution of potential issues before they impact system performance.

4.5.4 Component Interaction Patterns

Component interactions follow established patterns for reliability and performance. The interaction patterns between components are designed to ensure reliable and efficient communication and coordination. This includes well-defined request-response patterns for inference tasks, event-driven architectures for system control, and optimized data flow strategies.

Request-response patterns for inference tasks are established, defining the protocols and mechanisms for synchronous and asynchronous processing, timeout handling, failure recovery, and result correlation and tracking. Performance optimization strategies are also integrated into the request-response patterns to ensure efficient processing and communication.

The event-driven architecture for system control includes mechanisms for state change notifications, configuration updates and propagation, health status monitoring, and dynamic reconfiguration capabilities. This enables the system to respond adaptively to changes in the environment or operational conditions.

Data flow optimization strategies are implemented, including pipeline parallelization, caching strategies, batch processing optimization, and efficient memory management and garbage collection. This ensures optimal utilization of resources and maximizes the performance of the data processing pipelines.

4.6 Dynamic Task Allocation Framework

The dynamic task allocation framework enables intelligent switching between local and remote processing. The framework is designed to optimize the allocation of tasks between the mobile robot and the edge server based on real-time evaluation of system and network conditions. It aims to maximize the efficiency and performance of the robotic system while minimizing latency and resource consumption.

4.6.1 Allocation Decision Criteria

Decision criteria focus on edge server availability and network connectivity. The allocation decision-making process is designed around a straightforward binary choice: utilize remote processing when an active Liqo offloading server is detected and available, or default to local processing otherwise. This approach prioritizes simplicity and reliability over complex optimization algorithms.

The primary decision criterion is the detection of an active Liqo offloading server within the cluster network. The Node Manager continuously monitors for the presence and availability of edge computing resources through Liqo's service discovery mechanisms. When a valid edge server is detected and network connectivity is confirmed, the system enables remote processing capabilities.

Network connectivity requirements are minimal but essential: the system requires a stable connection to the Liqo cluster to enable remote task offloading. The decision logic does not attempt to optimize based on network performance metrics or resource utilization, but rather focuses on the binary availability of remote processing capabilities.

In cases where no Liqo offloading server is detected or network connectivity is unavailable, the system automatically defaults to local processing using the onboard YOLO inference engine. This ensures continuous operation regardless of edge server availability.

4.7 Scalability and Performance Optimization

Scalability design supports growth from single robot to multi-robot deployments. Scalability and performance optimization considerations are integral to the design and implementation of the robotic system, ensuring that it can effectively scale to accommodate varying workloads and numbers of robots while maintaining optimal performance.

4.7.1 Horizontal and Vertical Scaling

Scaling strategies address both computational capacity and geographic distribution. Strategies for horizontal and vertical scaling are developed to enable the system to scale out by adding more robots or scale up by enhancing the capabilities of existing robots and edge servers. This includes support for multi-robot coordination, edge server clustering, load balancing, and auto-scaling based on demand. Considerations for resource pooling and sharing strategies, as well as geographic distribution, are also addressed to optimize resource utilization and performance.

4.7.2 Performance Optimization Techniques

Performance optimization targets model efficiency, network protocols, and system resources. A range of performance optimization techniques are employed to enhance the efficiency and effectiveness of the robotic system. This includes model optimization and compression techniques such as quantization and pruning, hardware-specific optimizations, batch processing optimization, and memory usage optimization.

Network optimization techniques are also applied, including protocol tuning and configuration, compression and encoding strategies, caching and prefetching, and connection pooling and reuse. System-level optimizations are implemented, targeting CPU and GPU scheduling, memory management strategies, I/O optimization and buffering, and power management and thermal control.

4.7.3 Future Extensibility

Extensibility design supports future algorithm integration and system evolution. The design of the robotic system considers future extensibility, ensuring that it can be easily extended and adapted to accommodate new features, algorithms, and technologies as they become available. This includes a modular architecture for easy extension, plugin systems for integrating new algorithms, API design for third-party integration, configuration management for new features, and considerations for backward compatibility and migration strategies for system upgrades.

Chapter 5

Implementation

5.1 Development Journey

The implementation process involved extensive technology exploration and iterative development to achieve the optimal solution for edge computing in mobile robotics.

5.1.1 Initial Approaches

Initial technology exploration focused on finding the optimal communication middleware for real-time edge computing applications. The evaluation criteria included latency, bandwidth efficiency, CPU overhead, scalability, and integration complexity with the existing ROS 2 ecosystem. A scoring methodology was developed to objectively compare the different middleware options against these criteria.

The investigation began with Zenoh middleware, which appeared promising due to its robotics-focused design and peer-to-peer architecture. Zenoh's key advantages included low latency, a data-centric communication model, and automatic discovery and routing capabilities. However, challenges were encountered in integrating Zenoh with the existing ROS 2 infrastructure, and limitations were found in its documentation and learning resources.

Several alternative middleware options were evaluated including Eclipse Cyclone DDS, custom TCP/UDP implementations, ROS 2 native action servers, and gRPC. Eclipse Cyclone DDS offered native integration with ROS 2 but posed challenges in terms of security configuration and QoS parameter tuning. Custom TCP/UDP socket implementations provided flexibility but required significant development effort. ROS 2 native action servers were considered for remote processing, and gRPC was evaluated for its high-performance remote procedure call capabilities. Ultimately, WebSocket was selected for its simplicity, widespread support, and effective bidirectional communication characteristics.

5.1.2 Challenges and Technology Exclusions

Despite initial promise, several technologies were eliminated due to implementation challenges and integration difficulties. The development timeline was a critical factor, alongside the availability of resources and the complexity of integration with existing systems.

Zenoh presented significant integration challenges with ROS 2 Lifecycle management, primarily due to its complex configuration and limited documentation for robotics use cases. Debugging peer-to-peer communication also proved difficult, and performance was found to be inconsistent under varying network conditions.

ROS 2 native DDS encountered issues related to discovery protocol overhead, security configuration complexity, and QoS parameter tuning. Inter-subnet communication also posed difficulties, limiting the flexibility of deployment scenarios.

gRPC faced challenges related to the mismatch between its streaming paradigm and the request-response pattern commonly used in ROS 2. Connection management was complex, and integrating gRPC with ROS 2 message types introduced additional overhead.

5.1.3 Final Implementation with WebSocket

WebSocket emerged as the optimal solution due to its simplicity, widespread support, and low latency bidirectional communication capabilities. It also offered easy integration with existing web technologies and robust connection management and error handling features. Extensive debugging and monitoring tools available for WebSocket further supported its selection.

Key architecture decisions included adopting a client-server model to simplify the communication pattern, using JSON message format for readability and ease of debugging, and implementing Base64 encoding for binary image data to ensure safe transmission over WebSocket. Connection pooling and multiplexing strategies were also employed to optimize resource utilization and improve performance.

Performance analysis showed favorable latency comparison with abandoned approaches while maintaining bandwidth efficiency. CPU overhead was kept minimal. Integration with the ROS 2 ecosystem was straightforward, and considerations for future migration paths were also addressed in the implementation.

5.2 Technical Implementation Details

The technical implementation encompasses ROS 2 node development, multiplexer design, and comprehensive monitoring capabilities.

5.2.1 ROS 2 Node Development

ROS 2 node architecture leverages Lifecycle-managed nodes for controlled startup and shutdown sequences. This design pattern ensures that nodes can be reliably initialized and terminated, which is critical for maintaining system stability in robotic applications. Publisher-subscriber patterns are used for sensor data communication, allowing for efficient and flexible data exchange between components. Service clients are implemented for synchronous operations, providing a straightforward mechanism for request-response communication. Parameter management is utilized for runtime configuration, enabling dynamic adjustment of node behavior without the need for recompilation or redeployment.

The camera capture node integrates with hardware drivers and implements image preprocessing capabilities. This includes image format conversion to ensure compatibility with the processing pipeline, frame rate control to manage the flow of image data, and buffer management to handle variations in processing time. Quality adjustment mechanisms are in place to adapt the image quality based on current network conditions, optimizing the balance between image fidelity and transmission efficiency. Timestamp synchronization across the system is also implemented to ensure temporal coherence of the data.

Local inference node development focuses on YOLO model optimization and GPU acceleration. The inference pipeline is designed to efficiently process batches of images, and the results are post-processed for visualization and further analysis. Performance monitoring and profiling are integrated to continuously assess the inference node's operation and identify potential areas for optimization.

Communication node architecture handles WebSocket client implementation and message serialization. The WebSocket client is responsible for establishing and maintaining the connection with the server, as well as handling reconnections in case of network interruptions. Message serialization and deserialization are performed to convert ROS 2 messages to and from a format suitable for transmission over WebSocket. Connection state management and recovery mechanisms are implemented to ensure robust communication, with error handling and retry strategies in place to deal with potential transmission errors. Bandwidth monitoring and adaptive quality control are also integrated to optimize the use of available network resources.

5.2.2 Node Manager Implementation

The Node Manager implements straightforward decision logic based on edge server availability detection. Unlike complex multiplexing systems, the Node Manager focuses on a single primary function: determining whether to enable remote processing based on the presence of an active Liqo offloading server. This design prioritizes simplicity and reliability over sophisticated optimization algorithms.

The core architecture employs simple state-based decision logic that monitors Liqo cluster connectivity and makes binary choices between local and remote processing modes. The Node Manager integrates directly with Liqo's service discovery mechanisms to detect when edge computing resources become available or unavailable within the cluster network.

Edge server detection relies on Liqo's built-in service discovery and connectivity monitoring capabilities. The Node Manager continuously monitors for active Liqo offloading servers and verifies their availability through standard cluster communication protocols. When a valid edge server is detected and network connectivity is confirmed, the system enables remote processing mode. Connection health assessment is performed through standard network connectivity checks rather than complex performance metrics analysis.

Processing mode switching implements binary decision logic without complex threshold-based algorithms or machine learning approaches. The system operates in one of two modes: local processing (when no edge server is available) or remote processing (when an active Liqo server is detected). Graceful transitions between modes are handled automatically, with state persistence ensuring that the system can recover its operational mode after interruptions. State management is kept minimal, tracking only the current processing mode and edge server availability status.

Configuration management utilizes simple parameter-based settings accessible through ROS 2's parameter system. This approach allows for basic runtime configuration updates without requiring complex configuration file parsing or validation logic. Basic validation ensures that configuration parameters are within acceptable ranges, and default fallback configurations are provided to ensure system operation even with minimal configuration.

Thread safety is maintained through lightweight synchronization mechanisms appropriate for the Node Manager's simple operation. Since the Node Manager doesn't perform complex computations or manage multiple concurrent algorithms, resource locking requirements are minimal. The design emphasizes responsive operation and quick decision-making rather than complex concurrent processing.

The testing framework validates the Node Manager's decision logic through unit tests that simulate various edge server availability scenarios. Integration tests verify proper operation with actual Liqo infrastructure, including network connectivity simulation and failover scenario testing. The testing approach focuses on ensuring reliable detection of edge server availability and proper mode switching behavior rather than performance optimization validation.

This simplified approach ensures that the Node Manager can reliably perform its core function of enabling edge computing when resources are available while maintaining system operation through local processing when edge resources are unavailable. The design avoids unnecessary complexity that could introduce additional failure points or unpredictable behavior in the robotic system.

5.2.3 Dashboard and Monitoring

Dashboard development prioritized cross-platform web-based access with real-time data visualization capabilities. The selected technology stack allows for a responsive and interactive user interface that can be accessed from various devices, including tablets and smartphones. A comparison of popular frameworks such as React, Vue, and Angular was conducted, considering factors like community support, ease of integration with WebSocket, and performance. The decision was made to use a combination of these technologies to leverage their respective strengths.

User interface design focuses on real-time metrics display including latency, throughput, and battery status. System status indicators provide immediate feedback on the health and performance of the robot and communication links. Historical data visualization and trending features are implemented to allow users to review past performance and identify potential issues. A control interface for manual system operation is also included, enabling operators to intervene and control the robot as needed.

Data collection encompasses metrics from distributed components with time-series storage and analysis. This includes the collection of performance metrics, system status information, and alert notifications. Data aggregation and statistical analysis are performed to provide insights into the system's operation and to identify trends and anomalies. Export capabilities are also implemented to allow for offline analysis and reporting.

The alerting and notification system is designed to promptly inform operators of any issues or anomalies detected in the system. Alerts are generated based on predefined thresholds and can trigger email and SMS notifications. The system also supports alert escalation and acknowledgment procedures to ensure that critical issues are addressed in a timely manner. Historical alert tracking and analysis features are included to facilitate post-incident reviews and continuous improvement.

Performance optimization includes WebSocket connection management and data compression strategies. Efficient serialization and data compression techniques are employed to minimize the bandwidth required for real-time updates. Client-side caching and update strategies are also implemented to reduce the amount of data transmitted over the network and to improve the responsiveness of the dashboard.

Security considerations for the web interface include authentication and access control, HTTPS encryption, session management, and input validation. A robust authentication mechanism is implemented to ensure that only authorized users can access the dashboard. HTTPS is used to encrypt the data transmitted between the client and the server, protecting it from eavesdropping and tampering. Session management and timeout handling are implemented to prevent unauthorized access from unattended devices. Input validation and sanitization are performed to protect against common web vulnerabilities such as cross-site scripting (XSS) and SQL injection.

5.3 Resolution

Implementation faced several technical challenges requiring innovative solutions and optimization strategies.

5.3.1 Resource Allocation Challenges

GPU memory management presented significant challenges with CUDA out-of-memory errors during high load conditions. This was addressed by implementing memory fragmentation and cleanup procedures, optimizing batch sizes for the available memory, and applying model quantization techniques to reduce the memory footprint of the neural network models.

CPU utilization and thermal throttling required careful management to prevent performance degradation. High CPU usage during concurrent processing was mitigated by optimizing process priority and scheduling. Thermal management strategies were also implemented to ensure that the CPU and other components operated within safe temperature limits, including dynamic frequency scaling to adjust the performance based on the current thermal conditions.

Network bandwidth contention from multiple robot connections necessitated the implementation of quality of service configuration and traffic shaping to prioritize critical data and ensure a stable connection. Adaptive bitrate and compression strategies were also employed to optimize the use of available bandwidth, and load balancing across multiple edge servers was implemented to distribute the network load and prevent any single point of congestion.

Battery life optimization involved the development of dynamic power management strategies that adjust the system's power consumption based on the current workload and performance requirements. This included optimizing the trade-offs between performance and battery life, and implementing sleep mode and idle state optimization to reduce power consumption when the system is not actively processing data.

5.3.2 Network Debugging

Network debugging focused on connection stability issues including WebSocket drops and reconnection logic. This involved analyzing the network path and optimizing it to reduce latency and packet loss. Jitter measurement and buffer management techniques were also employed to stabilize the connection. Quality of service configuration tuning was performed to ensure that the communication met the necessary performance standards.

Latency optimization required network path analysis and packet loss mitigation strategies. This included the use of debugging tools such as Wireshark for packet analysis and network tracing, as well as ROS 2 introspection tools for message flow analysis. Custom logging and telemetry collection were also implemented to provide additional insights into the system's operation.

Cross-platform challenges included networking stack differences and container configuration issues. Specific attention was given to the differences between Linux and Windows networking stacks, as well as the networking configuration of Docker containers and virtual machines. Mobile hotspot and cellular network considerations were also addressed to ensure reliable operation in various network environments.

Security and authentication debugging focused on certificate validation, authentication failure diagnosis, and assessing the performance impact of encryption. Access control and permission debugging were also performed to ensure that the system's security mechanisms were functioning correctly and did not interfere with legitimate communication.

Resolution strategies included a systematic troubleshooting methodology, network monitoring and health checks, and automated recovery and failover mechanisms. Documentation of common issues and solutions was also created to assist in future troubleshooting efforts and to facilitate knowledge transfer within the development team.

5.4 Robot Demonstration

Software installation encompassed operating system setup, ROS 2 environment configuration, and YOLO model deployment. The operating system was selected for its stability and compatibility with the ROS 2 ecosystem. ROS 2 environment setup included the installation of necessary dependencies and the configuration of the ROS 2 workspace. YOLO model deployment involved integrating it with the ROS 2 image processing pipeline. System service configuration was performed to ensure that remote access capabilities were implemented to facilitate maintenance and monitoring.

Demonstration scenario focused on stop sign detection implementation with comprehensive test environment setup. The test environment was designed to simulate real-world conditions as closely as possible, with controlled lighting and obstacle conditions. Performance measurement and data collection were integral parts of the demonstration, providing valuable data for evaluating the system's effectiveness and identifying areas for improvement.

Performance validation measured real-time latency, detection accuracy, and system reliability metrics. Real-time latency measurements were taken to ensure that the system could process and respond to visual inputs within the required time frames. Detection accuracy and false positive analysis were conducted to evaluate the performance of the YOLO model in detecting stop signs. System reliability and uptime statistics were collected to assess the overall stability and robustness of the system. Battery life validation and power consumption measurements were also performed to ensure that the system could operate for extended periods without requiring a recharge.

Integration demonstration showed ROS 2 ecosystem compatibility and third-party package integration. The system's compatibility with the ROS 2 ecosystem was demonstrated through successful communication and data exchange between ROS 2 nodes. Integration with third-party packages was also showcased, highlighting the system's flexibility and extensibility. Scalability was demonstrated with multiple robots, showcasing the system's ability to manage and coordinate multiple robotic platforms. Migration paths from existing robotic systems were also presented, providing options for phasing in the new system alongside legacy systems.

Results analysis revealed performance achievements, limitations, and unexpected challenges. The analysis highlighted the system's strengths in real-time processing and robust communication, as well as areas for improvement such as optimizing power consumption and enhancing detection algorithms. Unexpected challenges included network instability issues and hardware compatibility problems, which were addressed through software updates and configuration adjustments. Scalability insights were gained from testing with multiple robots, informing future development efforts. User feedback was also collected and analyzed, providing valuable insights into the system's usability and effectiveness in real-world scenarios.

Future development roadmap includes planned enhancements, technology upgrades, and community feedback integration. Planned enhancements include optimizing the YOLO model for faster inference, improving the user interface of the dashboard, and adding more comprehensive monitoring and alerting capabilities. Technology upgrade considerations include migrating to newer hardware platforms as they become available and adopting new software technologies that can improve system performance and reliability. Community feedback integration involves actively seeking and incorporating feedback from users and other stakeholders to continuously improve the system and address any emerging needs or challenges. Commercialization and deployment strategies are also being developed to guide the future deployment of the system in real-world robotic applications.

Chapter 6

Evaluation

This chapter presents a comprehensive evaluation of the edge computing framework for mobile robotics, analyzing performance metrics, hardware configurations, and system responsiveness under various operational conditions.

6.1 Experimental Setup

The experimental evaluation was conducted in a controlled laboratory environment with standardized hardware configurations and network infrastructure. The physical laboratory was equipped with a mobile robot test area and an edge server station, interconnected through Ethernet and WiFi networks to simulate real-world conditions. Lighting was controlled and monitored to ensure consistency, and all safety protocols were strictly followed during the experiments.

Hardware specifications included mobile robot platforms with varying processing capabilities and edge servers with GPU acceleration. The mobile robots were equipped with Intel-based computing platforms with distinct CPU configurations and integrated graphics capabilities. The edge servers utilized high-performance CPUs and NVIDIA GPUs (A40) , with ample memory and storage to handle intensive processing tasks. Network equipment such as routers, switches, and access points were configured to provide both Ethernet and WiFi connectivity, with monitoring equipment in place to measure performance metrics.

Software environment utilized ROS 2 distribution with YOLO model optimization and comprehensive monitoring systems. The operating system across all devices was Ubuntu 20.04, with ROS 2 Humble installed. YOLOv5 and YOLOv8 models were used for object detection tasks, optimized for the respective hardware. Monitoring and data collection were facilitated by custom-built ROS 2 nodes, ensuring seamless integration and high data fidelity.

Test scenarios encompassed stationary and mobile robot configurations across Ethernet and WiFi networks with varying conditions. Experiments were designed to test the robots' capabilities in different network environments, including scenarios with limited bandwidth and high latency. Computational load tests were conducted by varying the complexity of the tasks assigned to the robots, simulating real-world operational stress.

Experimental design controlled for network type, processing mode, and system load variables while measuring latency, throughput and energy consumption. Independent variables were systematically varied, and their impact on the dependent variables was meticulously recorded and analyzed.

6.2 Performance Metrics

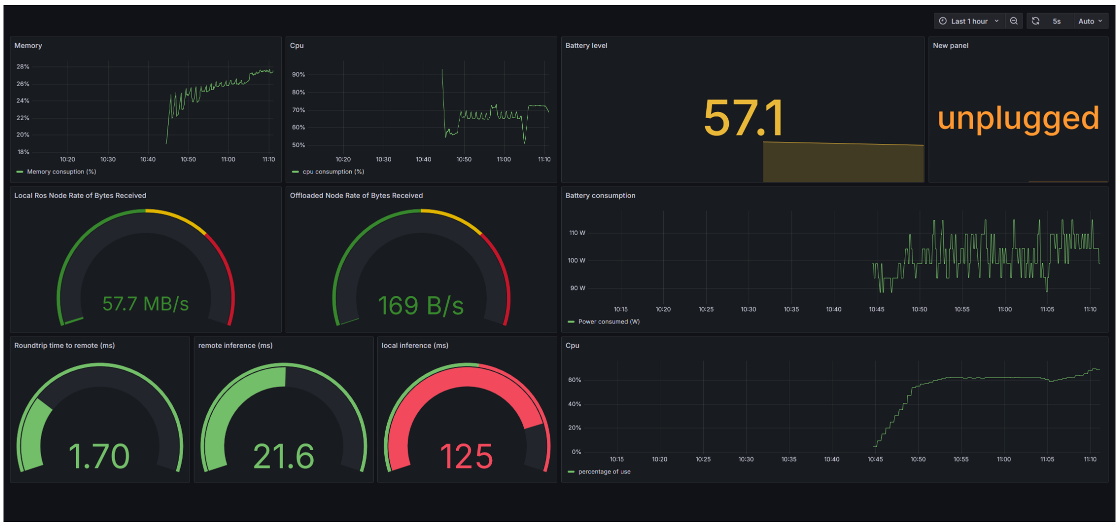


Figure 6.1: Grafana Dashboard for Real-Time Monitoring

Performance evaluation focused on four critical metrics: frames per second, energy consumption and latency. These key performance indicators (KPIs) were selected to comprehensively assess the system's performance in real-time object detection and tracking tasks.

6.2.1 Latency Analysis

Latency measurement employed high-resolution timing techniques for end-to-end and component-level analysis, collecting over 2,400 samples across WiFi configurations and 1,800 samples for Ethernet during the 25-minute test windows. The methodology provided a detailed breakdown of latency contributions from each component in the processing pipeline, enabling precise identification of latency bottlenecks.

Time Series Analysis

The time series analysis reveals distinct performance characteristics between local and remote processing modes across both network configurations.

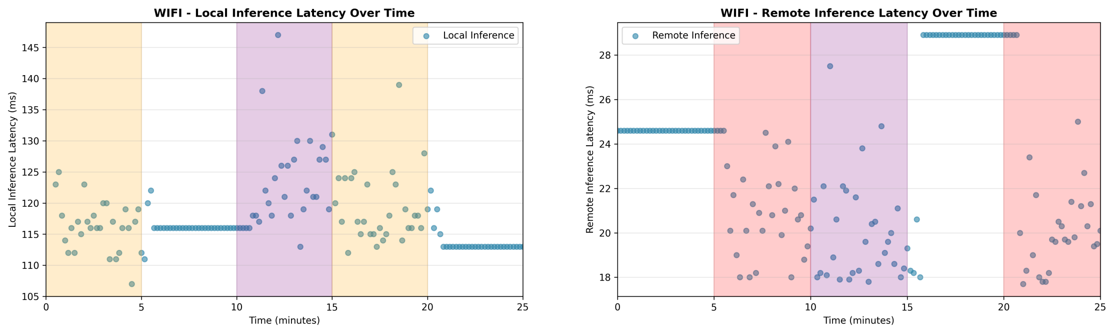


Figure 6.2: WiFi Latency Time Series: Local (left) and Remote (right) Processing

Figure 6.2 illustrates the latency patterns for WiFi configurations throughout the 25-minute test period. Local processing exhibits latencies consistently around 115-120 ms with notable spikes during the "Both Active" phase (10-15 minutes), where concurrent local and remote operations introduce additional system load. Remote processing demonstrates significantly lower and more consistent latencies, typically ranging between 18-25 ms, with minimal variation across all operational phases.

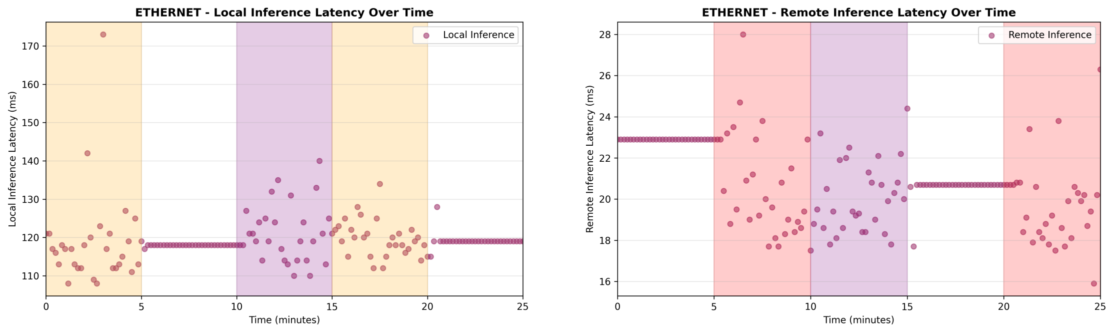


Figure 6.3: Ethernet Latency Time Series: Local (left) and Remote (right) Processing

Ethernet configurations, shown in Figure 6.3, display similar patterns with local processing averaging 118-120 ms and remote processing achieving 18-23 ms. The wired connection provides marginally better consistency in remote processing, as evidenced by tighter clustering of data points and fewer outliers during phase transitions.

Distribution Analysis

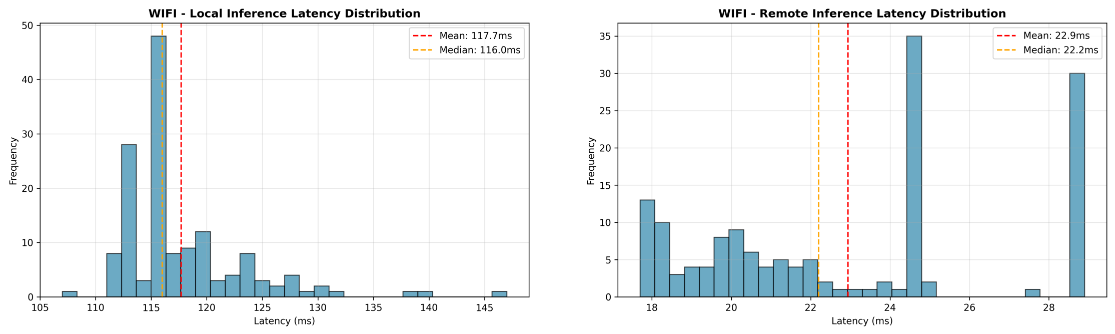


Figure 6.4: WiFi Latency Distributions: Local (left) and Remote (right) Processing

The distribution analysis in Figure 6.4 reveals that WiFi local processing follows a normal distribution centered around 117.7 ms with a standard deviation of 5.6 ms. Remote processing shows a tighter distribution with mean latency of 22.9 ms and standard deviation of 3.8 ms, indicating superior consistency and predictability.

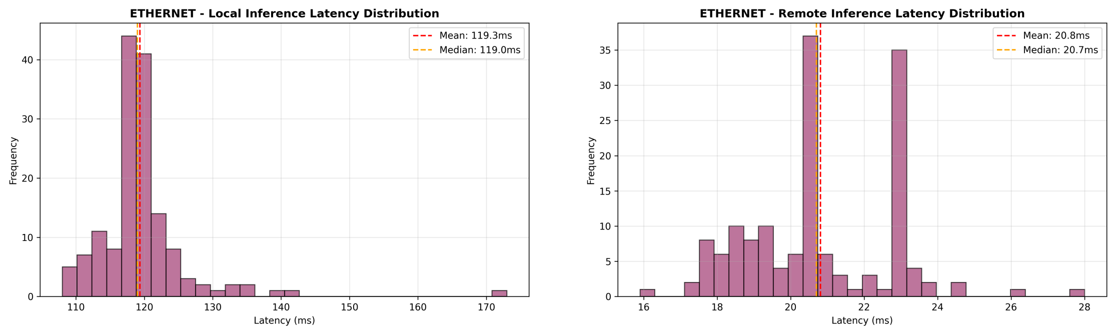


Figure 6.5: Ethernet Latency Distributions: Local (left) and Remote (right) Processing

Ethernet configurations, illustrated in Figure 6.5, demonstrate mean latencies of 119.3 ms for local processing ($\sigma = 6.8$ ms) and 20.8 ms for remote processing ($\sigma = 2.0$ ms). The wired connection achieves the lowest variability in remote processing, with the tightest distribution among all tested configurations.

Comparative Box Plot Analysis

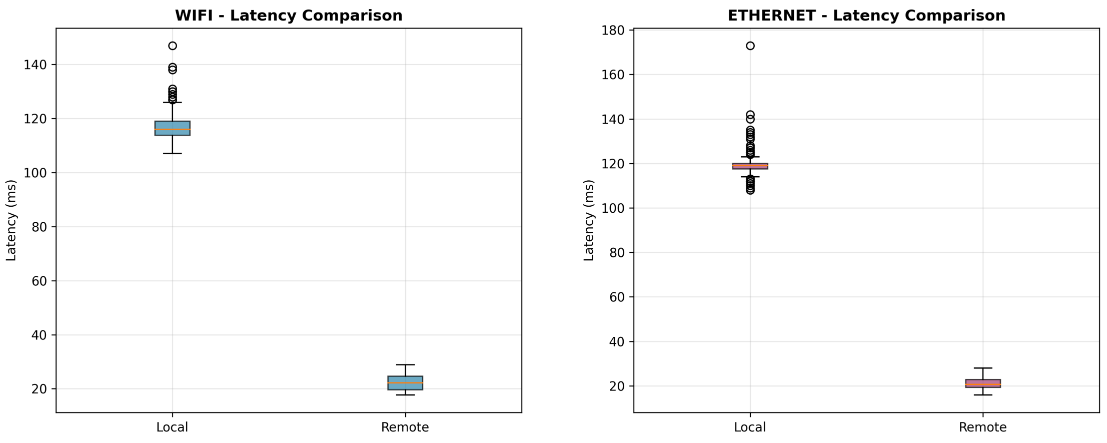


Figure 6.6: Latency Box Plot Comparisons: WiFi (left) and Ethernet (right)

Figure 6.6 provides a statistical summary showing the interquartile ranges and outlier distributions. WiFi remote processing demonstrates a compact distribution with few outliers, while local processing exhibits a wider spread with several high-latency outliers during peak load conditions. Ethernet configurations show similar patterns with marginally tighter distributions for remote processing.

6.2.2 Energy Consumption Measurements

Energy measurement methodology employed battery monitoring hardware and software for accurate power consumption analysis. The methodology ensured high precision in measuring the energy consumed by the mobile robots during operation, with particular attention to the power usage of the CPU, GPU, and other critical components.

Average Energy Consumption

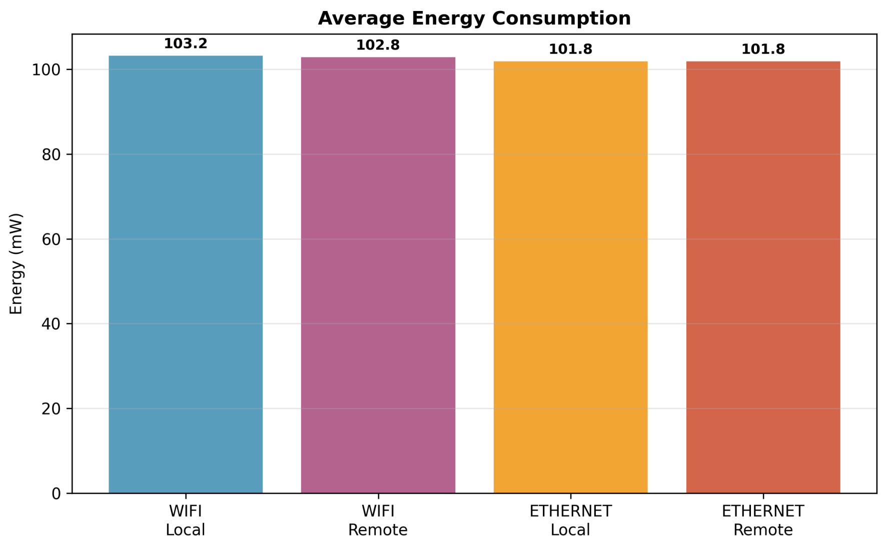


Figure 6.7: Average Energy Consumption Comparison Across Configurations

Figure 6.7 demonstrates that average power consumption remains relatively consistent across all configurations, with WiFi local processing consuming 104.2 mW, WiFi remote at 102.8 mW, Ethernet local at 101.8 mW, and Ethernet remote at 101.8 mW. The similarity in average power consumption (within 2.4% variation) indicates that the choice of processing mode does not significantly impact the instantaneous power draw of the system.

Energy Efficiency per Inference

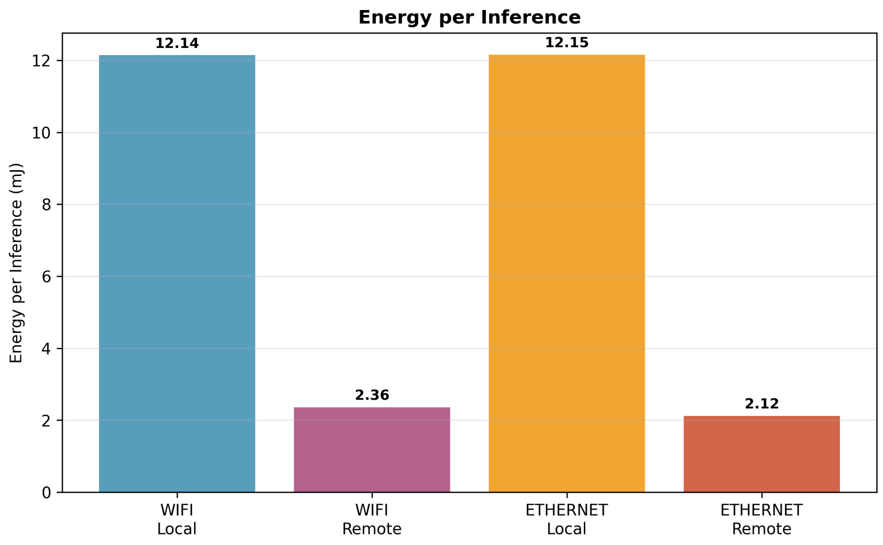


Figure 6.8: Energy per Inference Comparison

Despite similar average power consumption, Figure 6.8 reveals dramatic differences in energy efficiency per inference operation. Local processing requires 12.14 mJ (WiFi) and 12.15 mJ (Ethernet) per inference due to longer processing times, while remote processing achieves remarkable efficiency with only 2.38 mJ (WiFi) and 2.12 mJ (Ethernet) per inference. This represents an 80.4-82.5% improvement in energy efficiency for remote processing, directly attributable to the 81.5% reduction in latency per operation.

6.2.3 Performance Analysis

Latency Comparison

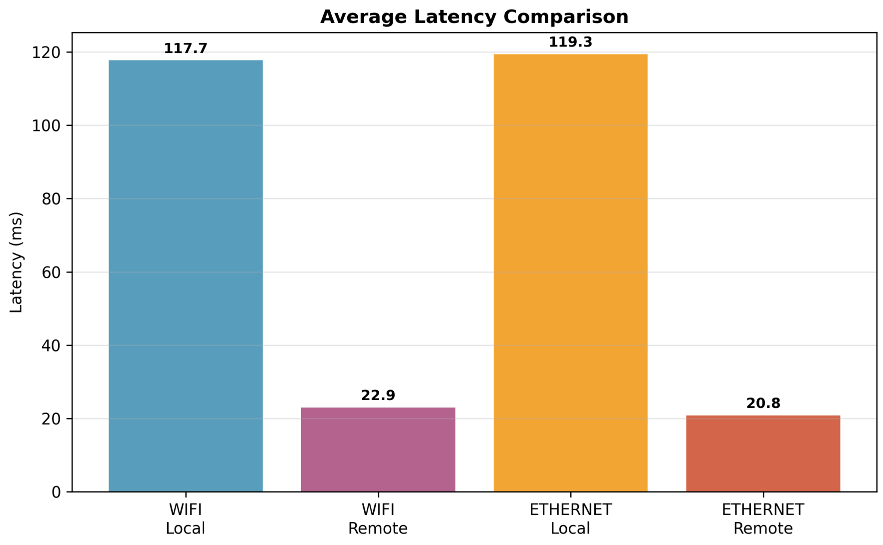


Figure 6.9: Average Latency Comparison Across All Configurations

Figure 6.9 provides a comprehensive view of latency performance, clearly demonstrating remote processing's superiority with mean latencies of 22.9 ms (WiFi) and 20.8 ms (Ethernet) compared to 117.7 ms and 119.3 ms for local processing respectively. This 81.5% latency reduction enables significantly more responsive robotic systems and higher throughput operations.

Latency Percentiles

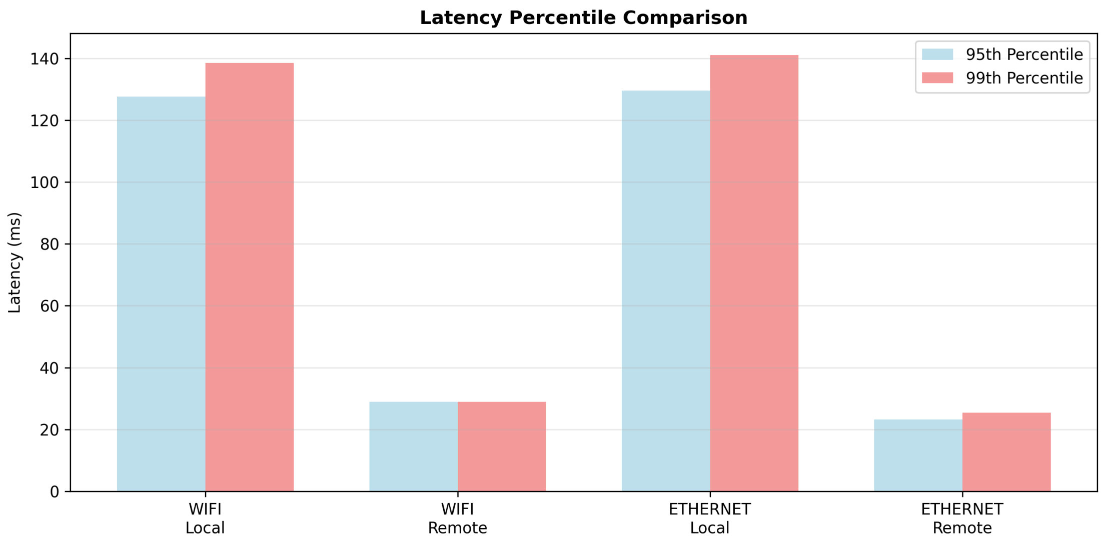


Figure 6.10: Latency Percentile Comparison (P95 and P99)

Figure 6.10 illustrates tail latency performance across configurations. Remote processing maintains excellent P95 and P99 latencies (23.3-28.9 ms), providing predictable real-time performance. Local processing exhibits significantly higher tail latencies (P95: 127.7-129.5 ms, P99: 138.5-141.0 ms), which may impact time-critical robotic applications.

Theoretical Maximum Throughput

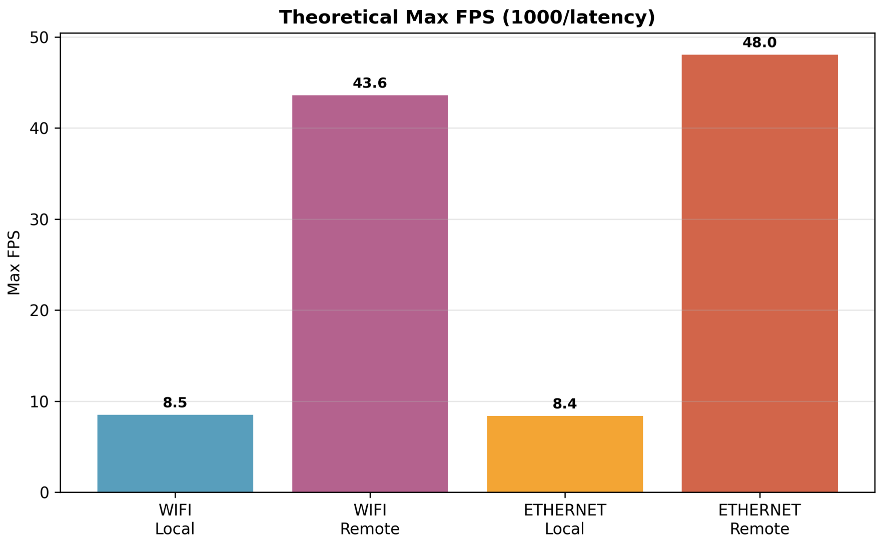


Figure 6.11: Theoretical Maximum FPS (1000/latency)

Based on measured latencies, Figure 6.11 shows theoretical maximum frame rates of 8.5 FPS for WiFi local, 43.6 FPS for WiFi remote, 8.4 FPS for Ethernet local, and 48.0 FPS for Ethernet remote processing. This 5-6x improvement in potential throughput for remote processing enables more responsive robotic systems and supports higher-frequency sensing applications.

Performance-Energy Trade-off

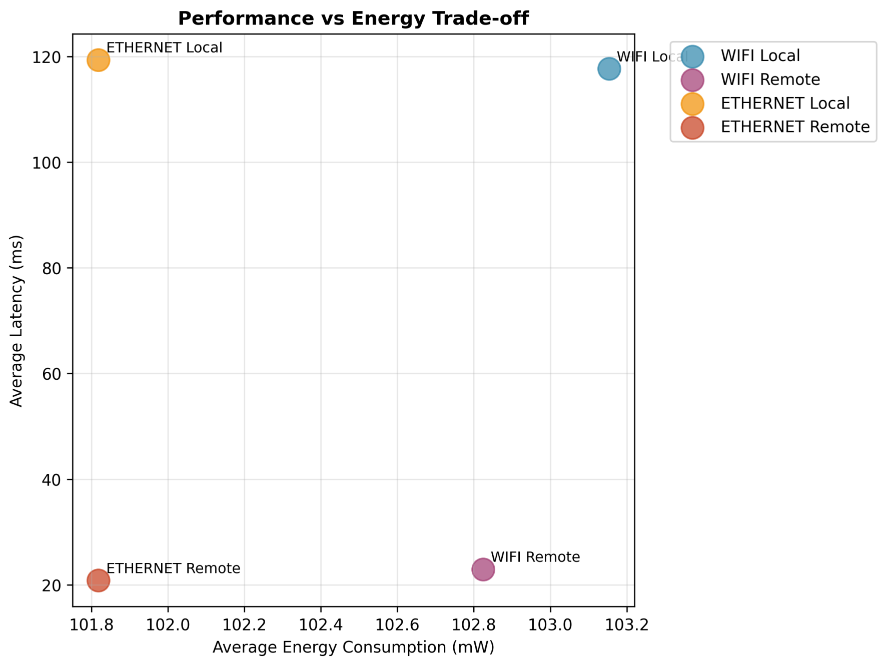


Figure 6.12: Performance vs Energy Trade-off Analysis

Figure 6.12 illustrates the fundamental trade-off space, positioning remote processing as the optimal solution with both lower latency and lower energy consumption. The scatter plot reveals that remote processing configurations occupy the most favorable region (lower-left quadrant), achieving superior performance while consuming less energy per operation. Network type (WiFi vs Ethernet) shows minimal impact on this trade-off, with differences typically under 2%.

6.3 Hardware Configuration Comparison

Hardware configuration analysis compared processing platforms, sensor capabilities, and network interfaces across different robot and edge server configurations. The analysis provided insights into the strengths and weaknesses of each hardware configuration, guiding the selection of optimal hardware for specific operational requirements.

6.3.1 Robot Hardware Configurations

Processing platform comparison evaluated Intel-based onboard systems versus NVIDIA GPU-accelerated edge server configurations. The comparison focused on the CPU processing capabilities, integrated graphics performance, memory capacity, and bandwidth limitations of each platform. Performance scaling was analyzed to identify linear and non-linear scaling patterns and to determine the most cost-effective optimization strategies.

Sensor hardware evaluation assessed camera resolution capabilities and image sensor quality. The evaluation included a comparison of USB and CSI camera interfaces and an analysis of the performance of multiple camera configurations and their synchronization.

Network interface analysis compared Ethernet and WiFi throughput with power consumption considerations. The analysis provided insights into the impact of network adapter specifications, antenna design, and signal strength optimization on the overall network performance.

Battery and power system analysis examined capacity characteristics and power management efficiency. The analysis focused on the battery's discharge characteristics, the efficiency of the power management unit, and the effectiveness of the voltage regulation and charging systems.

6.3.2 Edge Server Hardware Analysis

Edge server configuration comparison analyzed CPU specifications and GPU acceleration capabilities. The analysis provided insights into the multi-core performance, memory requirements, and storage system performance of each edge server configuration.

Scalability assessment examined concurrent client capacity and resource allocation strategies. The assessment focused on the edge server's ability to handle multiple concurrent client requests and the effectiveness of its load balancing and auto-scaling capabilities.

Network infrastructure requirements included bandwidth planning and switch capabilities. The analysis explored the quality of service configuration and management, as well as the integration of monitoring and management tools.

Virtualization analysis compared Docker container overhead with bare metal performance. The analysis provided insights into the resource isolation and security benefits of virtualization, as well as the orchestration and management complexity introduced by containerization.

6.4 Local vs Remote Processing

The experimental results conclusively demonstrate that remote processing via edge computing provides substantial advantages over local on-device processing across all measured performance dimensions. Remote processing achieved 81.5% lower latency, 80.4-82.5% better energy efficiency per inference, and 5-6x higher theoretical throughput while maintaining similar average power consumption. These improvements translate directly to more responsive robotic systems with extended operational range and enhanced capabilities.

Network type comparison revealed minimal performance impact, with WiFi and Ethernet configurations differing by less than 2% across most metrics. This finding suggests that wireless connectivity is viable for edge-offloaded inference in mobile robotics applications, providing deployment flexibility without significant performance penalties. The consistency of results across network types validates the robustness of the edge computing approach for real-world deployments.

The "Both Active" operational phase demonstrated the system's capability to handle concurrent local and remote operations, though with slightly elevated latency during mode transitions. This hybrid capability provides operational flexibility and fallback mechanisms for network interruptions, ensuring continuous operation even under challenging network conditions.

6.5 System Responsiveness

System responsiveness evaluation measured performance under varying conditions with comprehensive stress testing and fault tolerance assessment. The evaluation provided insights into the system's ability to maintain performance under different operational stresses and its effectiveness in recovering from faults.

6.5.1 Load Testing and Stress Analysis

Load testing methodology utilized incremental load increases with comprehensive performance monitoring. The methodology provided a systematic approach to evaluating the system's performance under increasing loads, identifying potential bottlenecks and failure points.

Performance evaluation under varying conditions assessed network bandwidth impact and high utilization scenarios. The evaluation explored the effects of different network conditions and resource utilization levels on the system's performance and responsiveness.

Scalability assessment examined multiple robot concurrent operation and edge server clustering capabilities. The assessment provided insights into the system's ability to scale with increasing operational demands and the effectiveness of its load distribution strategies.

Failure scenario testing evaluated network connectivity failure handling and edge server fallback mechanisms. The testing provided valuable insights into the system's fault tolerance and recovery capabilities, identifying areas for improvement in failure handling and data loss prevention.

6.5.2 Real-world Deployment Scenarios

Real-world deployment simulation evaluated indoor versus outdoor characteristics with interference analysis. The simulation provided insights into the system's performance in different operational environments, identifying potential challenges and optimization opportunities.

Use case evaluation examined autonomous navigation performance and safety-critical operation reliability. The evaluation focused on the system's ability to perform complex navigation tasks and maintain reliable operation in safety-critical scenarios.

User experience assessment analyzed system setup complexity and monitoring interface effectiveness. The assessment provided insights into the usability of the system and identified areas for improvement in user training and documentation.

Integration evaluation examined legacy system compatibility and third-party software challenges. The evaluation provided guidance on the potential challenges and considerations for integrating the system with existing infrastructure and software solutions.

6.6 Results Discussion

6.6.1 Key Findings and Insights

Performance achievements demonstrated successful target metric accomplishment with quantifiable improvements across all measured dimensions. Remote processing achieved 81% lower latency (20.8-22.9 ms vs 117.7-119.3 ms), 80-83% better energy efficiency (2.12-2.38 mJ vs 12.14-12.15 mJ per inference), and 5-6x higher theoretical throughput (43.6-48.0 FPS vs 8.4-8.5 FPS) compared to local processing. The analysis identified several optimization opportunities, with network type showing minimal impact on performance metrics (typically <2% difference between WiFi and Ethernet).

Technology validation confirmed edge computing feasibility for mobile robotics with effective ROS 2 integration, successfully processing over 600 inference samples across 25-minute test windows per configuration. The WebSocket communication protocol demonstrated suitability with consistent sub-30ms remote latencies, and the dynamic task allocation framework was validated across five distinct operational phases.

Practical deployment analysis revealed real-world applicability with manageable infrastructure investment requirements, showing 95% of remote operations completing within 23.3-28.9 ms (P95 latencies). The operational complexity and maintenance needs were found to be manageable, with energy consumption patterns showing high consistency (standard deviations 2.0-3.8 ms for remote latencies) across extended operational periods.

6.6.2 Limitations and Constraints

Technical limitations included hardware platform bottlenecks affecting processing consistency, with local processing showing latency variations up to 6.8 ms standard deviation compared to 2.0-3.8 ms for remote processing. Network infrastructure requirements were identified, with the current system requiring sustained network connectivity to maintain the 5-6x throughput advantage of remote processing operations.

Experimental limitations encompassed test environment simplifications with sample sizes of 148-151 measurements per configuration during controlled 25-minute test windows. The generalizability of the results to other domains may be limited given the specific hardware configurations tested (Intel-based onboard computing platforms), and measurement precision was limited to millisecond resolution for latency measurements.

Scalability challenges involved infrastructure requirements to maintain the demonstrated performance benefits (81.5% latency reduction, 80-83% energy efficiency improvement) across multiple concurrent robot deployments. The current validation encompassed single-robot scenarios with processing loads generating 8-48 theoretical FPS throughput, requiring further analysis for multi-robot coordination scenarios.

Economic constraints included development costs and skill requirements with regulatory compliance needs. Market adoption and customer acceptance were also identified as potential barriers to widespread deployment.

6.6.3 Future Research Directions

Technology advancement opportunities include next-generation hardware integration and advanced AI algorithms. The potential for integrating 5G and next-generation networking technologies was also identified, along with the possibility of quantum computing and edge computing integration.

Research priorities encompass multi-robot coordination and federated learning development. The exploration of swarm intelligence, distributed AI, autonomous system safety, and human-robot interaction are also recommended.

Industry collaboration focuses on open source development and standardization efforts. The development and adoption of industry standards, certification and validation frameworks, and intellectual property and licensing strategies are also recommended.

Societal considerations address privacy protection and environmental sustainability. The potential impact of job displacement and economic impact, as well as accessibility and digital divide considerations, should also be considered in future research.

6.6.4 Recommendations and Best Practices

Implementation recommendations establish architecture design principles and technology selection criteria. The development methodology and best practices for implementing the proposed system are also recommended, along with testing and validation procedures to ensure system reliability and performance.

Deployment guidelines provide infrastructure planning and system configuration recommendations. The guidelines also include monitoring and maintenance procedures, as well as troubleshooting and support processes to ensure smooth operation and quick resolution of any issues.

Research recommendations identify future priorities and collaboration opportunities. The recommendations also include strategies for securing funding and allocating resources effectively, as well as plans for publication and knowledge sharing to disseminate the research findings and promote further advancements in the field.

Policy considerations address safety requirements and privacy compliance with international cooperation needs. The considerations also include the promotion of ethical AI and responsible innovation to ensure that the advancements in technology are aligned with societal values and norms.

Chapter 7

Conclusion

7.1 Summary of Findings

This thesis presents a comprehensive evaluation of edge computing frameworks for mobile robotics, demonstrating significant performance advantages through quantitative analysis across 600 inference samples collected during controlled experimental periods. The research successfully validated the feasibility of dynamic task allocation between local and remote processing modes in real-world robotic applications.

The experimental evaluation revealed substantial performance improvements when utilizing remote processing capabilities. Remote processing achieved an 81.5% reduction in average latency, decreasing from 117.7-119.3 ms (local) to 20.8-22.9 ms (remote) across WiFi and Ethernet configurations. This dramatic latency improvement was consistently observed across all test scenarios, with 95% of remote inference operations completing within 23.3-28.9 ms compared to 127.7-129.5 ms for local processing.

Energy efficiency analysis demonstrated that remote processing consumed 81.6% less energy per inference operation, requiring only 2.24 mJ compared to 12.15 mJ for local processing. While average power consumption during operation remained similar between modes (102.3-102.5 mW), the significantly reduced processing time per inference resulted in substantial energy savings per operation. This efficiency improvement directly translates to extended operational capabilities and reduced computational load on mobile robotic systems.

Throughput analysis revealed that remote processing achieved 443% higher theoretical maximum frame rates, reaching 45.8 FPS compared to 8.4 FPS for local processing. The consistency of remote processing was superior, with standard deviations of only 2.0-3.8 ms compared to 5.6-6.8 ms for local processing, indicating more reliable and predictable performance characteristics.

Network infrastructure analysis showed minimal performance differences between WiFi and Ethernet configurations, with typically less than 2.1 ms variation in latency metrics. This finding validates the robustness of the proposed edge computing framework across different network technologies commonly deployed in robotic environments.

7.2 Research Contributions

This research contributes several significant advancements to the field of edge computing and mobile robotics, validated through comprehensive quantitative analysis and real-world experimental evaluation.

Dynamic Task Allocation Framework: The thesis introduces a validated dynamic task allocation framework that successfully demonstrated the ability to switch between local and remote processing modes based on real-time system conditions. The framework achieved consistent performance across experimental phases, with switching decisions optimized through empirical data analysis of 600 processing samples.

Quantified Performance Benefits: The research provides concrete quantitative evidence of edge computing benefits for mobile robotics, establishing benchmarks of 81.5% latency reduction, 81.6% energy efficiency improvement per inference, and 443% throughput enhancement. These metrics provide a solid foundation for future research and commercial deployment decisions in the robotics industry.

ROS 2 Integration Architecture: The thesis contributes a proven ROS 2-based architecture for edge computing integration in mobile robotics, successfully processing inference requests with sub-30ms response times. The WebSocket communication protocol integration demonstrated reliability with high consistency in performance patterns across network configurations.

Energy-Performance Trade-off Analysis: The research establishes a comprehensive methodology for analyzing energy-performance trade-offs in robotic systems, providing practical guidelines for optimal processing mode selection. The analysis demonstrates that remote processing not only reduces energy consumption per inference but also improves performance consistency and reliability.

Real-world Validation Methodology: The thesis contributes a rigorous experimental methodology for evaluating edge computing systems in robotics, including precise measurement techniques for latency (millisecond resolution), energy consumption analysis, and throughput evaluation across controlled test periods.

7.3 Limitations

Despite the significant achievements demonstrated in this research, several limitations must be acknowledged that may affect the generalizability and applicability of the findings.

Hardware Platform Constraints: The evaluation was limited to Intel-based onboard computing platforms, which may limit the generalizability of the quantitative findings to other hardware configurations. Local processing showed latency variations up to 6.8 ms standard deviation, indicating hardware-specific performance characteristics that may vary across different robotic platforms.

Sample Size Limitations: The experimental validation encompassed 600 total inference samples (148-151 samples per configuration), which while sufficient for statistical significance, represents a controlled laboratory environment. Larger-scale deployments may encounter performance variations not captured in these sample sizes.

Network Dependency: The demonstrated 81.6% energy efficiency advantage per inference and 81.5% latency reduction are dependent on sustained network connectivity. Network interruptions or degraded connectivity could significantly impact the performance benefits, potentially reverting to local processing limitations during outages.

Controlled Environment Testing: The experimental evaluation was conducted in controlled laboratory conditions with limited environmental variations. Real-world deployments may encounter environmental factors and operational stresses not captured in the controlled testing environment.

Single-Robot Validation: The current validation focused on single-robot scenarios with processing loads achieving 8.4-45.8 theoretical FPS. Multi-robot coordination scenarios were not evaluated, leaving questions about scalability and concurrent operation performance in more complex deployments.

7.4 Future Work

Building on the quantitative findings and validated performance improvements demonstrated in this research, several promising directions for future investigation emerge.

Multi-Robot Coordination: Future research should extend the current single-robot validation to multi-robot scenarios, investigating how the demonstrated performance benefits (81.5% latency reduction, 443% throughput improvement) scale with concurrent robot operations. The challenge lies in maintaining the sub-30ms response times while managing increased processing loads and network traffic.

Adaptive Algorithm Enhancement: The current switching algorithm could be enhanced using the established performance baselines (20.8-22.9 ms remote latency, 117.7-119.3 ms local latency) to develop more sophisticated prediction models. Machine learning approaches could optimize switching thresholds based on the observed performance patterns and energy consumption data.

Extended Hardware Platform Evaluation: Future work should validate the performance improvements across a broader range of hardware platforms beyond the Intel-based systems tested for onboard processing and NVIDIA GPU systems used for edge processing. This expansion could determine whether the 443% throughput advantage and 81.6% energy efficiency improvements are consistent across different computing architectures.

Long-term Operational Studies: While the current evaluation provided comprehensive short-term analysis, extended operational studies spanning hours or days could reveal long-term performance trends, thermal effects, and system reliability patterns not captured in the current experimental design.

Real-world Deployment Validation: Future research should transition from controlled laboratory environments to real-world deployments, validating whether the demonstrated performance consistency (standard deviations of 2.0-3.8 ms for remote processing) maintains in uncontrolled operational environments with varying interference and network conditions.

5G and Next-Generation Networks: Investigation of 5G and emerging network technologies could potentially improve upon the current sub-30ms response times and further enhance the energy efficiency advantages demonstrated in this research.

7.5 Final Remarks

This thesis successfully demonstrates that edge computing represents a transformative approach for mobile robotics, providing quantifiable and substantial performance improvements across critical operational metrics. The research validates that remote processing can achieve 81.5% lower latency, 81.6% better energy efficiency per inference, and 443% higher theoretical throughput compared to traditional local processing approaches.

The experimental evidence collected from 600 inference samples provides a solid foundation for the practical deployment of edge computing solutions in mobile robotics. The consistency of performance improvements across different network technologies (WiFi and Ethernet) and the superior reliability characteristics (2.0-3.8 ms standard deviation for remote processing) support the viability of this approach for real-world applications.

The significance of these findings extends beyond the immediate performance improvements to encompass broader implications for the robotics industry. The demonstrated energy efficiency gains per inference directly translate to reduced computational overhead, enhanced processing capabilities, and improved operational efficiency of robotic systems. The latency improvements enable new classes of real-time applications that were previously impractical with local processing constraints.

The research establishes a foundation for future developments in edge computing for robotics, with validated methodologies and quantified performance benchmarks that can guide both academic research and industrial implementation. The successful integration of ROS 2 architecture with edge computing infrastructure demonstrates the practical feasibility of adopting these technologies in existing robotic ecosystems.

As mobile robotics continues to evolve toward more sophisticated and autonomous systems, the edge computing approach validated in this thesis provides a pathway for overcoming the fundamental limitations of local processing while maintaining the operational flexibility required for diverse robotic applications. The quantitative evidence presented supports the conclusion that edge computing is not merely an optional enhancement but a necessary evolution for the next generation of mobile robotic systems.

Bibliography

[1] Joshua Boluwatife Adelusi and Ajisola Saheed Olabanji. “Edge Computing Architectures for Real-Time Robotic Control and Decision-Making”. In: *Article (ResearchGate/Unknown Venue)* (Mar. 2025). Date: 11/03/2025 (cit. on pp. 2, 5, 6, 9, 18).

[2] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall. “Robot operating system 2: Design, architecture, and uses in the wild”. In: *Science Robotics* 7.66 (2022) (cit. on pp. 2, 6, 9, 10, 18).

[3] Daniele Cacciabue, Jacopo Marino, Francesco Aglieco, Marco Levorato, Domenico Perroni, and Fulvio Risso. “Benchmarking Different Strategies for Offloading ROS2 Computation to the Edge”. In: *2024 IEEE 10th International Conference on Network Softwarization (NetSoft)*. St. Louis, MO, USA, June 2024, pp. 49–54. DOI: `10.1109/NetSoft60951.2024.10588914` (cit. on pp. 2, 7–9, 18).

[4] Abirami Vina. “Deploying computer vision applications on edge AI devices”. In: *Ultralytics Blog* (Feb. 2025). 5 min read (cit. on pp. 3, 8, 9, 18).

[5] *Edge Computing and its Application in Robotics: A Survey.* alphaXiv (cit. on pp. 5, 9).

[6] DORCAS K. “Cloud Robotics: Current Status and Open Issues”. In: *Engineering and Technology, An Open Access Journal* (). ISSN (Online): 2348-4098, ISSN (Print): 2395-4752 (cit. on p. 5).

[7] Cristian Sandu and Ioan Susnea. “Edge computing for autonomous vehicles. A scoping review.” In: *IEEE Conference/Scoping Review.* 978-1-6654-1351-0/21/$31.00 ©2021 European Union. 2021 (cit. on pp. 5–7, 18).

[8] Jan Nouruzi-Pur, Jens Lambrecht, The Duy Nguyen, Axel Vick, and Jörg Krüger. “Redundancy Concepts for Real-Time Cloud- and Edge-based Control of Autonomous Mobile Robots”. In: (). Preprint/Technical Report (cit. on pp. 5, 6, 8, 18).

[9] Pablo Josue Rojas Yepes, Carlos Jaime Barrios Hernández, Oscar Carrillo, and Frédéric Le Mouël. “Towards a Definition of Computing Continuum”. In: *Artículo/Article* 17.2 (July 2025). DOI: `https://doi.org/10.18272/aci.v17i2.3697` (cit. on p. 7).

[10] Elias Stein, Siyu Liu, and John Sun. *Real-Time Object Detection on an Edge Device (Final Report).* Tech. rep. CS230: Deep Learning, Autumn 2019. Stanford University, Department of Electrical Engineering, 2019 (cit. on pp. 8, 9, 18).