Throughput and Security Improvements in RoboRebound-Based Embedded Multi-Robot Systems

Yadnik Sanjay Bendale

A THESIS

in

Electrical Engineering

Presented to the Faculties of the University of Pennsylvania
in Partial Fulfillment of the Requirements for the
Degree of Master of Science in Engineering

2025

Linh Thi Xuan Phan
Supervisor of Thesis

Nicholas McGill-Gardner
Co-Supervisor of Thesis

Tania Khanna
Program Director, Electrical Engineering Master's Program

Acknowledgments

I would like to sincerely thank Professor Linh Thi Xuan Phan for her outstanding mentorship and guidance throughout the course of this thesis. Her expertise in real-time embedded systems and system security has deeply shaped the direction and quality of this work.

I am also immensely grateful to Neeraj Gandhi for his foundational work on RoboRebound and for offering continued support and insight during the architectural transition phases.

I would also like to thank Professor Nicholas McGill-Gardner for his invaluable assistance in designing and refining the printed circuit board layout that forms the core hardware foundation of this project.

Finally, I wish to thank the Resilient Real-Time Dependable Distributed Systems (R2D2) Lab at the University of Pennsylvania for providing a stimulating and collaborative research environment.

Abstract

RoboRebound is a secure, real-time protocol designed for coordination in embedded multi-robot systems. This thesis enhances the original RoboRebound architecture by replacing the legacy PIC32 and TinkerBoard S platform with a high-throughput, cryptographically accelerated hardware stack based on STM32H753ZI microcontrollers and the Raspberry Pi 5. The upgraded system enables real-time JPEG image streaming, hardware-assisted SHA-256 hashing, and HMAC-verified actuation within bounded latency constraints.

Key contributions include a modular embedded system design, efficient inter-node communication using UART and MQTT over Wi-Fi 6, and protocol-level improvements that enforce message integrity and freshness. The updated architecture significantly improves throughput, reduces jitter, and enhances resilience against spoofing and replay attacks. This thesis presents the system architecture, firmware pipeline, benchmarking results, and a full security analysis aligned with the RoboRebound protocol.

Contents

List of Figures

List of Tables

1 Introduction

Modern warehouse automation increasingly relies on fleets of autonomous mobile robots (AMRs) that collaborate to perform tasks like material transport, inventory monitoring, and real-time environmental sensing. These systems require precise coordination, trustable inter-node communication, and low-latency sensing-to-actuation pipelines. However, when deployed in shared environments with potential for wireless interference or compromised devices, these robots become susceptible to delays, spoofing, or even malicious control. This motivates the need for embedded systems that enforce both timing correctness and security guarantees across all stages of the sensing, decision, and actuation pipeline.

The original RoboRebound implementation was developed using the PIC32MX130F064B microcontroller and a TinkerBoard S single-board computer. While sufficient for basic logic and simple sensing, the platform quickly became a bottleneck for high-throughput, vision-based coordination. It lacked hardware cryptographic acceleration, had limited RAM and flash, and could not reliably process or verify images in real time. To address this, we migrated the architecture to use STM32H753ZI microcontrollers for the sensor and actuation nodes, and the Raspberry Pi 5 (8 GB) as the control node. This new architecture unlocks high-speed UART communication, real-time JPEG image streaming, and hardware-assisted SHA-256 hashing, which together form the basis for robust data integrity enforcement.

At the heart of the RoboRebound protocol is a three-node structure:

- A **Sensor Node (S-node)** that aggregates data from a GPS (MTK3333), IMU (ICM20948), and OpenMV camera, hashes the full payload using hardware SHA-256, and sends it forward.
- A **Control Node (C-node)** on Raspberry Pi 5 that verifies hashes, performs Canny edge detection on incoming images to detect obstacles, and signs motion commands with HMAC-SHA256.
- An **Actuation Node (A-node)** that verifies HMACs, drives the motors via PWM, and forwards telemetry over UART to an ESP32-C6 module, which broadcasts it using Wi-Fi 6 over MQTT.

The system is designed such that all telemetry - GPS location, IMU readings, vision frame metadata, and obstacle alerts, is published under a single MQTT topic per bot. This enables other robots to subscribe and react to state changes in a decentralized but verifiable manner. The inclusion of cryptographic primitives, hardware-based protection, and structured communication makes RoboRebound resilient to packet injection, spoofing, replay attacks, and desynchronization.

This thesis focuses specifically on warehouse robotics applications, where the need for secure, low-latency robot-to-robot coordination is most acute. The following chapters detail the full system architecture, firmware integration, security protocol, and experimental validation of the RoboRebound platform.

2 Related Work

The design of secure, real-time multi-robot coordination systems has long been a subject of active research across embedded systems, distributed computing, and robotics. RoboRebound builds on three distinct strands of prior work: (1) embedded systems for real-time sensing and actuation, (2) cryptographic enforcement in low-power devices, and (3) distributed multi-agent consensus under fault-prone or adversarial conditions. This chapter surveys the relevant work in these areas and highlights how RoboRebound offers a unique integration of these elements for use in warehouse-specific multi-robot systems.

2.1 Real-Time Embedded Coordination Systems

A wide range of systems have explored real-time sensing-to-actuation pipelines in embedded and cyber-physical systems. Platforms like OpenWSN and RIOT-OS offer deterministic scheduling and resource-constrained networking, targeting wireless sensor networks. Similarly, autopilot frameworks like PX4 and ArduPilot provide modular flight stacks for UAVs, focusing on sensor fusion and control loop timing.

However, these systems typically assume trusted infrastructure and do not account for Byzantine nodes or cryptographic verification of sensor data. In contrast, RoboRebound explicitly addresses scenarios where sensor input may be tampered with, delayed, or spoofed, and enforces integrity checks at each step of the pipeline.

Real-time operating systems such as FreeRTOS and Zephyr have also been used in distributed robotics [3]. While they provide deterministic task scheduling and low interrupt latency, they lack built-in support for integrity-verified communication. RoboRebound complements these platforms by introducing a cryptographic layer that operates within the real-time constraints of such systems.

2.2 Cryptographic Enforcement on Embedded Hardware

Securing microcontroller-based systems is challenging due to resource constraints on computation, memory, and power. Prior work has examined how to implement cryptographic primitives like AES and SHA on low-power MCUs [5]. Projects such as TockOS and TrustZone-M offer platform-level security primitives but require specific hardware features or trusted execution environments.

RoboRebound takes a lightweight approach by leveraging the native cryptographic hardware accelerators present on the STM32H753ZI. The use of SHA-256 and HMAC-SHA256 is tailored to real-time operation with minimal software overhead. This is similar in spirit to systems like SANCUS [4], which use hardware-isolated modules for secure code execution, but RoboRebound focuses on securing the data flow between nodes rather than just the code itself.

Protocols like TESLA and µTESLA have attempted to introduce time-based authentica-

tion in embedded environments [2]. While these approaches are promising, they assume synchronized clocks and pre-distribution of keys. RoboRebound assumes a simpler trust model and provides in-line HMAC verification without strict synchronization, making it more practical for distributed warehouse robots.

2.3 Secure Multi-Agent Coordination and Consensus

Consensus algorithms such as Paxos, Raft, and PBFT (Practical Byzantine Fault Tolerance) have long addressed the challenge of achieving agreement in fault-prone distributed systems. However, these are often too heavyweight or communication-intensive for embedded use [1].

In robotics, multi-agent coordination is frequently approached using distributed SLAM, behavior trees, or auction-based scheduling. Many of these assume reliable communication and trust between agents. RoboRebound diverges from this assumption by enforcing message integrity and authenticity using lightweight cryptography, allowing untrusted or partially compromised nodes to be safely integrated into a larger fleet.

Systems like F1/10 and Duckietown demonstrate coordinated autonomous navigation with multiple embedded agents, but again rely on trusted infrastructure. RoboRebound builds in resilience to adversarial settings by validating every sensor input and command before further propagation.

2.4 Telemetry and Communication in Distributed Robots

Wireless communication between robots has been explored using mesh networks (ZigBee, LoRa), peer-to-peer Wi-Fi, and cellular links. Early approaches relied on ad-hoc protocols with limited security. Modern systems like ROS 2 have introduced DDS-based communication with authentication and encryption, but these often assume high compute resources and reliable networks [6].

RoboRebound instead adopts a minimalist MQTT-based model using ESP32-C6 modules with Wi-Fi 6, broadcasting telemetry on a shared topic with HMAC validation. This allows other bots to verify the origin and freshness of the data without needing heavyweight stacks or OS-level security policies.

3 System Design and Firmware Integration

The RoboRebound system is architected around a secure, real-time, and modular framework to support coordination between autonomous mobile robots in warehouse environments. This chapter presents a detailed discussion of the embedded system architecture, hardware components, communication protocol, and firmware-level design that enables integrity-enforced sensing, control, and actuation. Each node is designed with specific roles and integrated through a structured pipeline built on lightweight cryptographic checks and UART-based communication.

3.1 Overall System Architecture

The system follows a three-node embedded model:

- **Sensor Node (S-node)**: STM32H753ZI microcontroller responsible for aggregating GPS, IMU, and vision data.
- **Control Node (C-node)**: Raspberry Pi 5 (8 GB) that performs image analysis and validates sensor integrity.
- **Actuation Node (A-node)**: STM32H753ZI microcontroller that receives verified control commands and drives actuators.

Each robot is also equipped with an ESP32-C6 module connected to the A-node via UART, which broadcasts telemetry over Wi-Fi 6 using a single MQTT topic. Other bots subscribe to this topic to receive updates and maintain fleet-level awareness, enabling secure, decentralized coordination [1].

3.2 Hardware Components and Interfaces

The hardware configuration is purposefully selected to maximize cryptographic performance and low-latency data handling:

- **STM32H753ZI**:
  - ARM Cortex-M7 @ 480 MHz
  - 1 MB SRAM, 2 MB Flash
  - Hardware SHA-256, AES, TRNG
  - 6 USART + 3 UART interfaces
- **GPS (MTK3333)**: Communicates over I^2C at 0x10; $GNRMC sentences are parsed for UTC time, location, and fix.

- **IMU (ICM20948)**: Interfaced over I^2C; collects 9-axis motion data (accelerometer, gyroscope, magnetometer).
- **OpenMV Camera**: Sends grayscale JPEG frames via UART to the STM32 S-node.
- **Raspberry Pi 5 (C-node)**: Receives sensor data via UART from S-node, performs processing, and transmits decisions to A-node.
- **ESP32-C6 Module**: Communicates with A-node over UART; uses MQTT to publish telemetry (e.g., `telemetry/botX`) using Wi-Fi 6.

All UART links operate at a baud rate of 115200 to maintain determinism and reduce CPU overhead.

3.3 Data Flow and Cryptographic Enforcement

At the S-node:

- Acquires GPS and IMU data over I^2C.
- Receives image frames from OpenMV via UART.
- Concatenates metadata (timestamp, GPS, IMU, vision).
- Computes a SHA-256 hash of the payload using the STM32 hardware accelerator.
- Sends the full payload and hash to the C-node over UART.

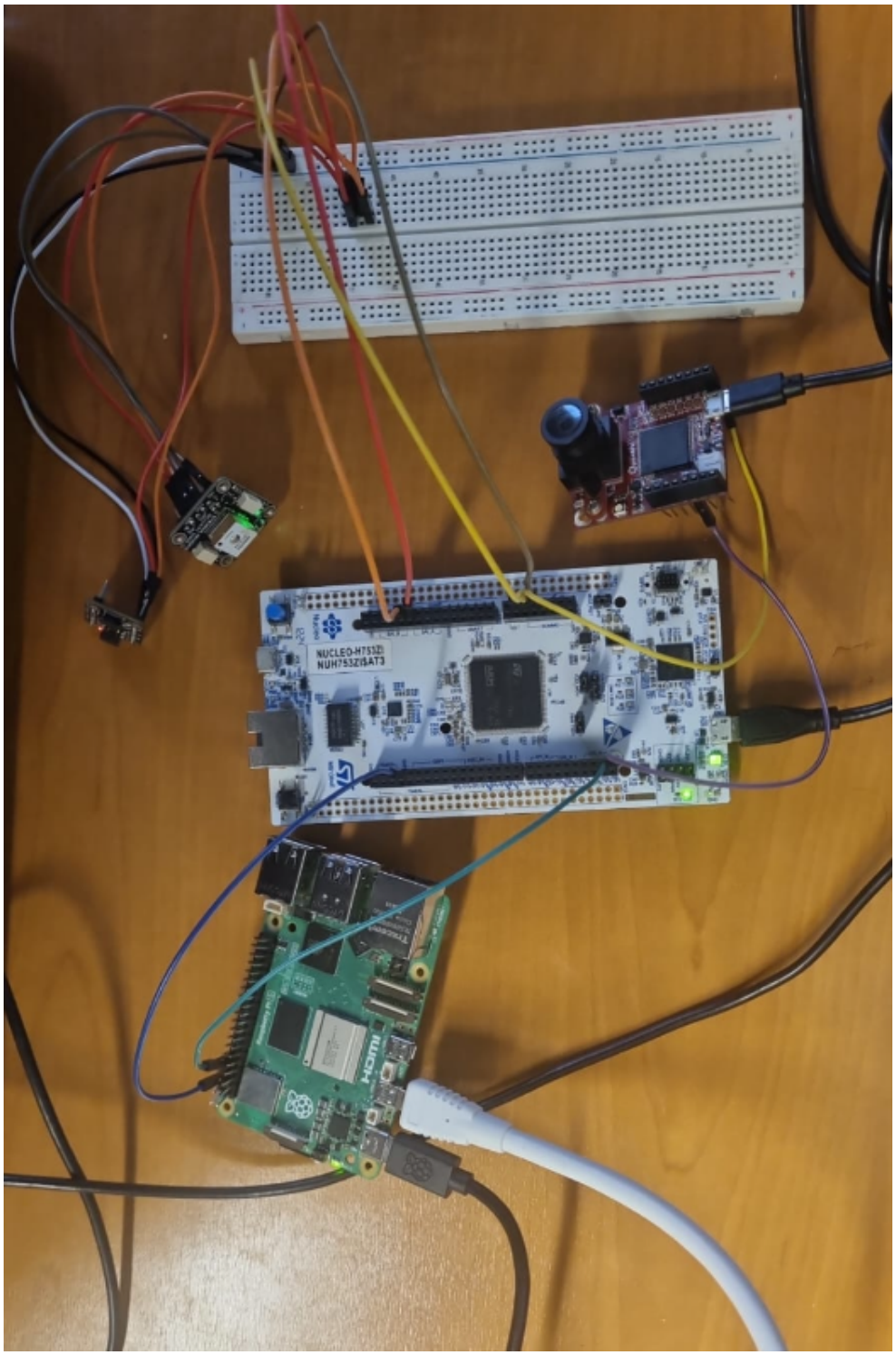

Figure 1: Firmware verification and data flow from S-node to C-node

At the C-node:

- Verifies the SHA-256 hash using Python's hashlib.

- Applies Canny edge detection to incoming images.
- Upon obstacle detection, creates a signed control command with the GPS coordinates and timestamp.
- Generates an HMAC-SHA256 signature over the command using a shared symmetric key.
- Sends the command + timestamp + HMAC to the A-node via UART.

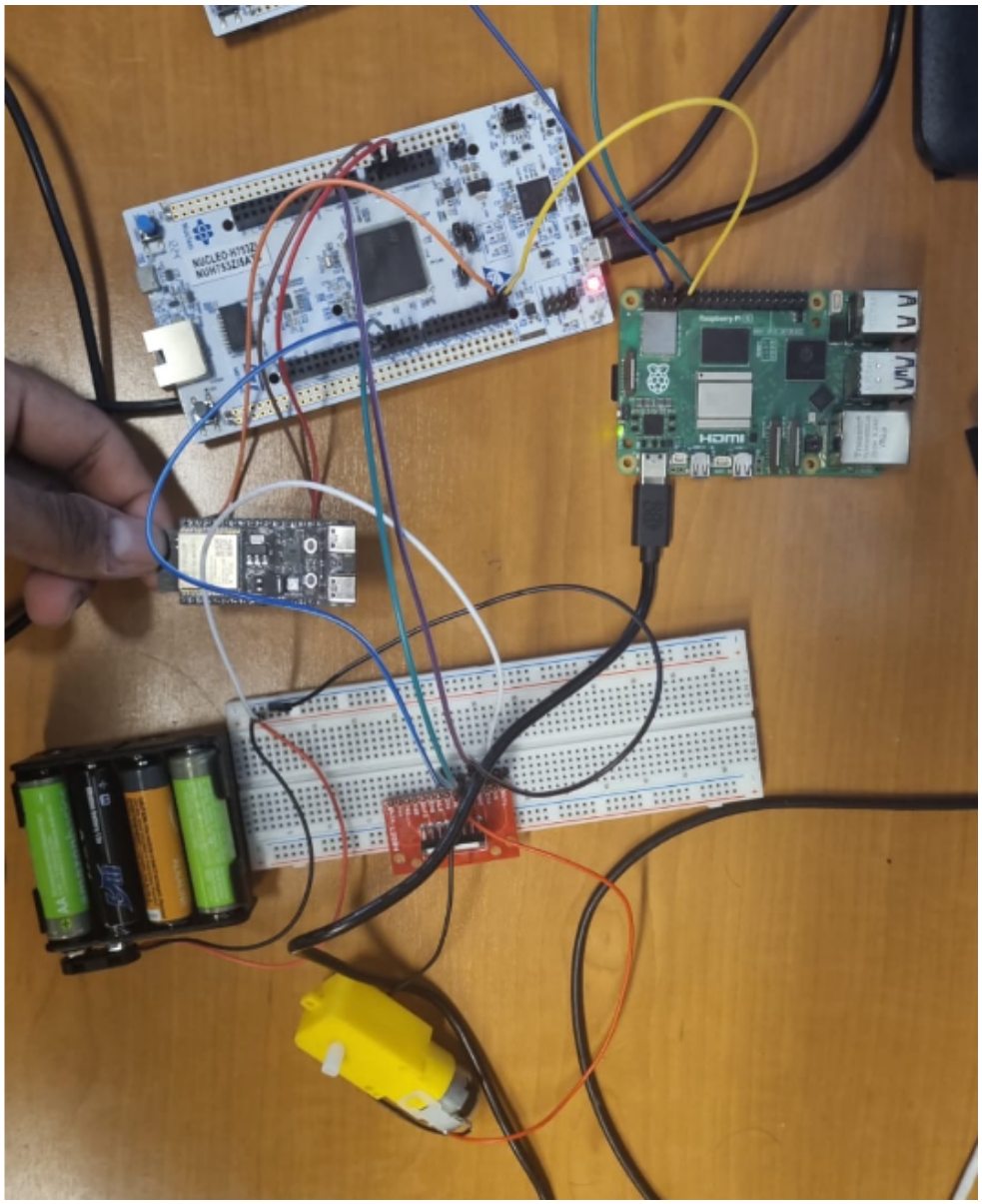

Figure 2: Firmware verification and control flow from C-node to A-node

At the A-node:

- Verifies HMAC using its local copy of the key.
- Verifies freshness via timestamp.
- Drives motors via PWM (using the L298N motor controller).
- Assembles telemetry including GPS, IMU, vision metadata, and obstacle status.
- Sends telemetry over UART to the ESP32-C6.

At the ESP32-C6:

- Publishes the telemetry packet to a shared MQTT topic.
- Adds a layer of HMAC if needed before broadcast.
- Enables other robots to verify message integrity in real-time.

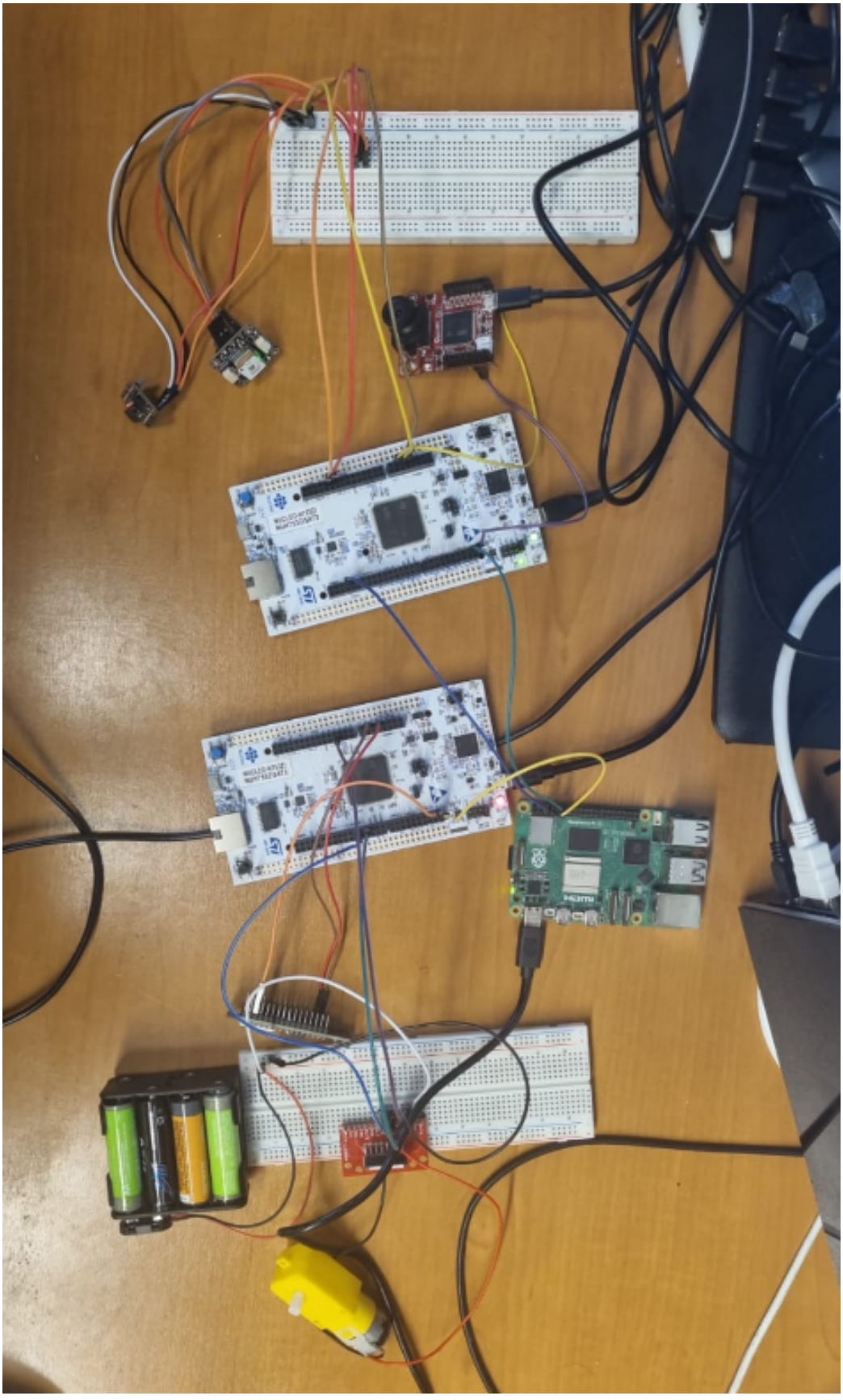

Figure 3: Overall firmware architecture combining S-node, C-node, and A-node

This tightly integrated cryptographic workflow ensures bounded-time interaction between sensing, decision-making, and actuation, even in adversarial settings [1].

3.4 Protocol Framing and Resilience

Each communication segment adheres to strict framing rules:

- **S-node to C-node:** `[START][LEN][PAYLOAD][SHA256_HASH][END]`
- **C-node to A-node:** `[START][COMMAND][TIMESTAMP][HMAC][END]`
- **A-node to ESP32:** Binary packet at fixed intervals

This structured framing allows boundary recovery in case of transmission loss and protects against buffer overflows or desynchronization. Each firmware state machine includes timeout logic that triggers fallback behavior (e.g., stopping motors) on communication failure.

3.5 Migration from PIC32 Architecture

The original RoboRebound prototype was implemented using a PIC32MX130F064B MCU and TinkerBoard S SBC. However, it presented several limitations:

- No hardware support for SHA-256 or HMAC operations
- Only 32 KB RAM and 64 KB flash - insufficient for image buffering
- Only 2 UARTs; difficult to interface GPS, IMU, camera, and output devices simultaneously
- TinkerBoard S had weak UART/SPI bandwidth and lacked secure memory features

Benefits of Migration to STM32H7 and Raspberry Pi 5:

- Hardware acceleration for SHA and HMAC significantly reduces CPU cycles for verification
- STM32H7's expanded memory accommodates large buffers for image and sensor data
- Up to 9 UART interfaces simplify modular integration of peripherals
- Raspberry Pi 5 provides high-speed USB, GPIO, and native Python support for edge detection and telemetry handling

The upgraded hardware architecture was pivotal in increasing system throughput, reducing image-to-action latency, and enabling real-time multi-robot coordination over secure wireless channels [5, 6].

4 Hardware Benchmarking and Throughput Improvement

This chapter presents a detailed benchmarking of the RoboRebound system's hardware, focusing on throughput, latency, and processing capabilities. It contrasts the original implementation, based on the PIC32 microcontroller and TinkerBoard S, with the upgraded system featuring STM32H753ZI microcontrollers and the Raspberry Pi 5. Key metrics such as data rate, real-time communication performance, and cryptographic processing are analyzed to quantify the impact of the migration. The goal is to demonstrate the practical benefits in terms of reliability, data fidelity, and scalability when applying RoboRebound in warehouse-scale multi-robot coordination.

4.1 Overview of Legacy Architecture

The original prototype of RoboRebound used the following hardware configuration:

- **S-node and A-node: PIC32MX130F064B**
  - 40 MHz clock
  - 32 KB RAM / 64 KB flash
  - No cryptographic acceleration
  - Limited UART support (2 UARTs, low speed)
- **C-node: TinkerBoard S**
  - 1.8 GHz quad-core ARM Cortex-A17
  - 2 GB RAM
  - 16 GB eMMC
  - Limited support for cryptographic libraries
- **Wireless Communication: 915 MHz RF modules**
  - Max data rate: 20 Kbps
  - Required custom software stack for encryption and retransmission
  - Highly sensitive to interference; suffered from jitter

This configuration faced significant limitations:

- Round-trip latencies often exceeded 200 ms
- Could not handle compressed image or high-frequency sensor data
- Lacked any hardware support for hashing, verification, or secure communication

4.2 Upgraded RoboRebound Hardware Stack

The new hardware architecture significantly improves the system's performance and scalability:

- **S-node and A-node: STM32H753ZI**
  - ARM Cortex-M7 @ 480 MHz
  - 1 MB SRAM, 2 MB Flash
  - Hardware support: SHA-256, AES, TRNG
  - 6 USART + 3 UART interfaces at 115200 baud
  - DMA support for UART and cryptographic operations
- **C-node: Raspberry Pi 5 (8 GB)**
  - Quad-core ARM Cortex-A76 @ 2.4 GHz
  - High-speed GPIO/UART interfacing
  - Supports OpenCV image processing and HMAC verification via Python
- **Wireless Communication: ESP32-C6**
  - Wi-Fi 6 support with data rates up to 8 Mbps
  - Secure MQTT with optional TLS and WPA3
  - Interfaces to A-node via UART at 115200 baud
- **Camera and Sensors**
  - OpenMV grayscale JPEG frames (80x60) at 2–3 FPS
  - GPS (MTK3333) and IMU (ICM20948) over I^2C

4.3 Throughput and Latency Comparison

Metric	**PIC32 + TinkerBoard S + RF Modules**	**STM32H7 + RPi 5 + ESP32-C6 (Wi-Fi 6)**
UART Baud Rate	38400 baud	921600 baud
Wireless Bandwidth	20 Kbps	4–8 Mbps (Wi-Fi 6)
Image Streaming	Not supported	JPEG (80x60) @ 2–3 FPS
Sensor Data Rate	Unreliable, async	Real-time, image-synced
SHA-256 Processing	Software only	Hardware-accelerated
HMAC Verification	Manual software routines	`hashlib` + HMAC on RPi
Round-trip Latency	200 ms	50 ms

Table 1: Throughput and Latency Benchmarking: Legacy vs Upgraded Hardware

4.4 Implications on Real-Time Performance

The transition from RF-based telemetry and limited compute capability to a high-throughput architecture has had a transformative impact. With Wi-Fi 6 and structured MQTT telemetry, bots can now:

- React to obstacle alerts within 50–100 ms.
- Stream vision data with synchronized GPS and IMU in real-time.
- Maintain decentralized control and coordination with cryptographic guarantees.

The STM32H7's peripheral DMA allows seamless, low-latency handling of UART streams, including JPEG and sensor payloads. Concurrently, the Raspberry Pi 5 runs lightweight Python scripts for hash verification and image classification, all while sustaining MQTT traffic.

In contrast, the legacy system's polling loops, slow hash routines, and RF-induced jitter made tight coordination infeasible. Additionally, packet loss and inability to recover desynchronized frames led to unrecoverable state inconsistencies.

Overall, the new stack brings warehouse-scale robot swarms closer to operational viability, meeting the performance and security requirements of modern cyber-physical systems [1,5].

5 Security Protocol and Threat Mitigation

The RoboRebound protocol is designed to address the dual challenges of real-time robotic coordination and adversarial resilience in warehouse environments. Given the increasing use of wireless communication, embedded sensors, and distributed control in autonomous systems, ensuring data integrity, message authenticity, and resistance to malicious interference is crucial. This chapter presents a detailed breakdown of the cryptographic mechanisms employed in RoboRebound, the associated threat models, and the formalization of risk using the STRIDE and DREAD frameworks.

5.1 Security Goals

RoboRebound is architected to satisfy the following core security goals:

- **Data Integrity:** Ensure that sensor, image, and telemetry data is unaltered in transit.
- **Command Authenticity:** Guarantee that actuation commands originate from a verified control source.
- **Message Freshness:** Protect against replay attacks using timestamps.

- **Bounded-Time Validity:** All security checks must execute within fixed deadlines.
- **Resilience to Node Compromise:** Isolate compromised nodes while preserving overall system functionality.

5.2 Cryptographic Foundations

The system utilizes two main cryptographic primitives:

1. SHA-256 (Sensor-to-Control Integrity)

- Computed on the STM32H7 S-node over every sensor payload.
- Includes GPS coordinates, IMU readings, and image metadata.
- Uses STM32H7 hardware hashing peripheral to minimize latency.
- Appends 32-byte hash to each UART packet before transmission to the C-node.
- C-node verifies the hash and rejects mismatches.

2. HMAC-SHA256 (Control-to-Actuation Authentication)

- Used to sign actuation commands from the C-node.
- Shared symmetric key between C-node and A-node.
- Validates authenticity and freshness (via timestamp).
- Packets failing HMAC verification are discarded.

5.3 Threat Model

- **Adversary Type:** Remote attacker or partial physical compromise.
- **Attack Surface:** UART, MQTT wireless, and command interfaces.
- **Capabilities:** Sniffing, injection, spoofing, and replay.
- **Defense:** All messages cryptographically bound to sources and verified at each hop.

5.4 STRIDE Threat Model Analysis

Threat	Mitigation
Spoofing	HMAC-SHA256 ensures commands originate from a known controller
Tampering	SHA-256 hash on sensor payloads detects any modification
Repudiation	Commands and telemetry are signed with cryptographic evidence
Information Disclosure	ESP32 supports Wi-Fi 6 encryption; MQTT payloads can be encrypted
Denial of Service	Watchdogs and bounded-time logic prevent lock-up or delay abuse
Elevation of Privilege	Node-specific keys and hardware IDs restrict unauthorized access

Table 2: STRIDE Threat Model Mapping in RoboRebound

5.5 DREAD Scoring

Category	Score	Justification
Damage Potential	9	Unauthorized commands could lead to collisions or dropped payloads
Reproducibility	8	Packet injection attacks are easily replicated without secure channels
Exploitability	7	Moderate technical skill enables spoofing without cryptographic checks
Affected Users	9	A single compromised node can disrupt the entire warehouse fleet
Discoverability	6	UART and MQTT topics are exposed on physical interfaces
Total	**39/50**	High-priority need for layered, embedded security

Table 3: DREAD Risk Assessment for RoboRebound

5.6 Role of Timestamps and Timeout Enforcement

To defend against replay attacks and stale data propagation, RoboRebound embeds a precise UTC-based timestamp into every data and command packet. This timestamp is generated at the S-node (using GPS-derived time) and is verified at both the C-node and A-node. Each receiving node compares the embedded timestamp with its own local clock. If the packet age exceeds a preset timeout window (typically 200 ms), the packet is discarded without processing.

This bounded freshness window ensures that no delayed, duplicated, or replayed packets can influence actuator behavior. For instance, even if a malicious actor captures and replays

a previously valid control packet, the outdated timestamp will trigger automatic rejection.

To further enforce timing guarantees, watchdog timers are configured on each STM32H7 node. These are hardware-based countdown timers that reset or interrupt the system if expected signals are not received within the configured interval. Their use enforces fail-safe behavior:

- **A-node Watchdog:** If a valid, HMAC-authenticated command is not received from the C-node within 250 ms, the A-node triggers a motor shutdown and enters a passive state.
- **C-node Timeout:** If the C-node does not receive a fresh SHA-verified payload from the S-node in 200 ms, it drops the current decision cycle and flags a sensor failure.
- **Bot-wide Telemetry Timeout:** If any bot does not receive telemetry from its peers via MQTT within 500 ms, it assumes a degraded network state and triggers conservative behaviors like obstacle stop or low-speed mode.

This distributed timing model allows all bots to maintain real-time coordination even under intermittent network failures, sensor dropouts, or intentional jamming attacks.

5.7 Wireless Security

Although MQTT is typically used over unencrypted channels in embedded systems, RoboRebound hardens this communication layer through multiple mechanisms:

- **Embedded HMAC-SHA256:** Every telemetry packet transmitted from the A-node to the ESP32-C6 contains an HMAC signature computed using a shared secret key. This signature binds the telemetry to the source bot and prevents forgery by third parties.
- **MQTT-over-TLS:** The ESP32-C6 module natively supports Transport Layer Security (TLS). When enabled, telemetry packets are encrypted before they are broadcast to the broker, preventing eavesdropping or message sniffing.

Together, these mechanisms prevent unauthorized access to robot telemetry, ensure that only authenticated bots can share data, and eliminate risks of man-in-the-middle (MITM) attacks in Wi-Fi environments.

5.8 Security in Real-Time Firmware

RoboRebound's real-time firmware is built with separation of privileges and strict execution ordering. Each node has well-defined security responsibilities and avoids privilege escalation:

- **S-node (STM32H7):** Performs sensor aggregation, builds a complete data packet, computes SHA-256 hash using the hardware accelerator, and transmits the data over UART. No command decisions or actuation logic are implemented on this node.
- **C-node (RPi 5):** Receives sensor data, verifies SHA-256 hash, performs image processing and edge detection, and formulates motion commands. It then computes HMAC signatures for the commands. No direct hardware-level control is issued from this node.
- **A-node (STM32H7):** Accepts HMAC-verified commands only if the timestamp is fresh. It generates PWM signals for motor drivers and transmits telemetry data. If HMAC verification fails, the actuators are immediately disabled.
- **ESP32-C6:** Acts solely as a communication relay. It cannot issue commands and is not wired to motor control. It only publishes telemetry to MQTT after verifying the structure of the telemetry packet and optionally adding a second layer of HMAC or encryption.

This modular separation and trust minimization ensures that even if one node is compromised, the damage is isolated, and core functions like motion control remain protected.

5.9 Attack Vector Reduction

RoboRebound significantly reduces the number of viable attack surfaces through the introduction of embedded security primitives. The following threats are explicitly eliminated:

- **Unsigned Sensor Packets:** Every sensor payload is hashed using SHA-256 before transmission, ensuring tampering is immediately detectable.
- **Blind Command Injection:** Commands without a valid HMAC are ignored by the A-node, rendering spoofed packets ineffective.
- **Replay Attacks:** All packets include timestamps and are checked against expiry windows before acceptance. Old commands or telemetry cannot influence system behavior.
- **Rogue Bot Telemetry Injection:** MQTT brokers accept packets only from pre-approved MAC addresses, and each telemetry message must carry a valid HMAC. This prevents adversaries from publishing false obstacle alerts or position data.

The remaining surface area such as physical tampering, firmware downgrades, or simultaneous compromise of all secret keys requires high sophistication and physical access. Even in such cases, firmware-level resets and node-unique keys provide a path to secure recovery.

6 PCB Design and Hardware Architecture

This chapter describes the complete hardware and mechanical architecture of the RoboRebound platform, including the custom PCB layout, power delivery system, sensor and actuator interconnections, and structural design of the embedded robotic unit. The design considerations prioritize modularity, secure power management, low electromagnetic interference, and seamless integration of STM32, ESP32, and RPi-based control logic.

6.1 PCB Design Overview

The custom RoboRebound PCB was designed in Altium Designer to support modular mounting of STM32H753ZI Nucleo boards, sensor interfaces, motor drivers, power supplies, and an ESP32-C6 wireless module. The PCB is two-layered, features dedicated ground planes for improved signal integrity, and is routed to support SPI and UART communication buses.

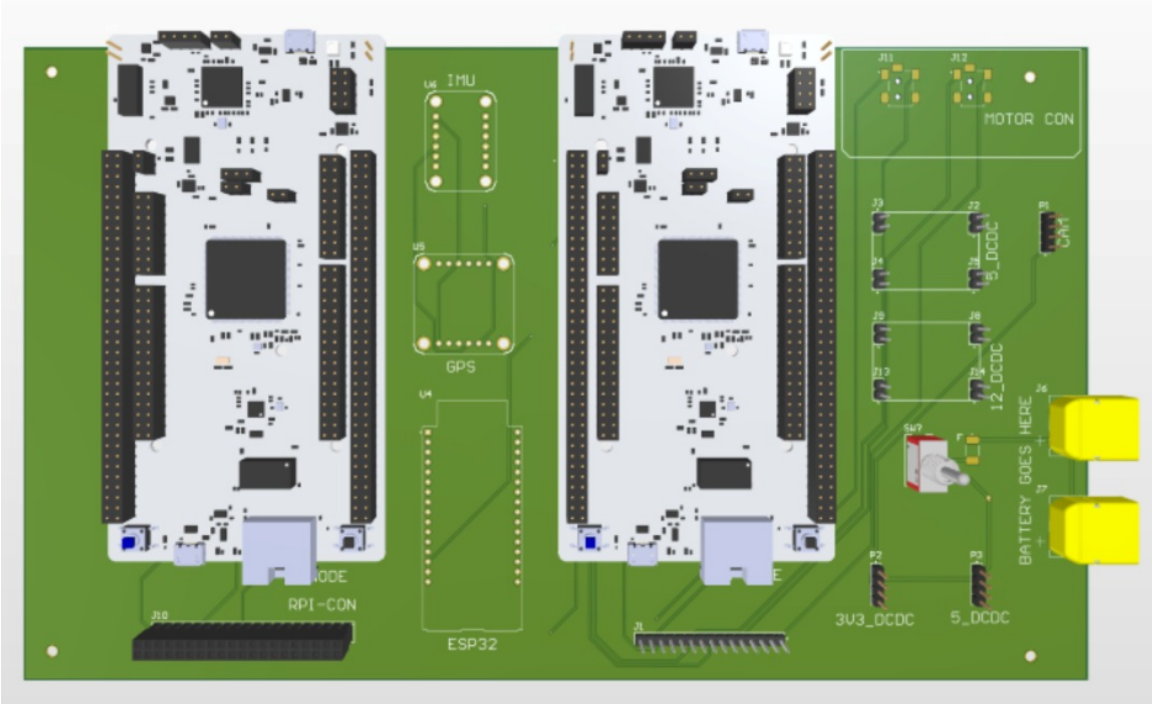

Figure 4: 3D PCB layout of the RoboRebound bot exported from Altium Designer showing module positioning

6.2 Power Delivery and Voltage Regulation

The robot is powered by two 7.4V, 2200 mAh, 50C LiPo batteries in series, yielding 14.8V. The system employs two DFRobot DC-DC Multi-Output Buck Converters and two MPM3610 buck regulators to distribute voltage efficiently across components.

Main Conversion Path

- **DFRobot Buck Converter 1:** 14.8V → 12V @ 5A (Motor driver: L298N)
- **DFRobot Buck Converter 2:** 14.8V → 5V @ 5A (Raspberry Pi 5)
- **MPM3610 Regulator 1:** 5V → 3.3V @ 1.2A (GPS and IMU sensors)
- **MPM3610 Regulator 2:** 5V → 5V @ 1.2A (STM32, ESP32, Camera)

Power Allocation and Estimated Current Draw

Component	Estimated Current Draw
STM32H753ZI Nucleo Boards (2)	500 mA @ 5V
ESP32-C6 (Wi-Fi TX Peak)	240 mA @ 5V
OpenMV Camera	140 mA @ 5V
MTK3333 GPS Module	50 mA @ 3.3V
ICM20948 IMU	10 mA @ 3.3V
Total (5V rail)	**880 mA (Supply: 1200 mA)**
Total (3.3V rail)	**60 mA (Supply: 1200 mA)**

Table 4: Power Distribution and Estimated Draw per Rail

This setup provides sufficient thermal margin and current headroom for burst operations, ensuring robust performance under load.

6.3 Data Flow and Communication Interfaces

RoboRebound follows a tightly structured data pipeline:

- **S-node (STM32H7):**
  - Captures GPS, IMU, and camera data
  - Computes hardware-accelerated SHA-256 on metadata
  - Transmits [DATA + HASH] over UART to the C-node (Raspberry Pi)
  - Also connected via SPI (slave) to A-node for validation polling
- **C-node (Raspberry Pi 5):**
  - Connected via UART to S-node and A-node
  - Runs OpenCV-based Canny edge detection
  - Verifies SHA-256, creates command packet, computes HMAC
  - Sends [COMMAND + TIMESTAMP + HMAC] over UART to A-node
- **A-node (STM32H7):**

  - Verifies HMAC and command freshness
  - Drives PWM motor signals if verification passes
  - Publishes telemetry via UART to ESP32-C6
  - Periodically polls S-node over SPI for verified data

- **ESP32-C6:**
  - Publishes telemetry packets under `telemetry/botX`
  - May optionally rewrap in TLS or attach a second-layer HMAC

6.4 Structural Design and Mounting

The mechanical structure consists of a dual-layer acrylic chassis fabricated via laser cutting.

Lower Acrylic Plate

- Holds LiPo battery pack and motors
- Velcro and acrylic cutouts used for vibration isolation and wire routing
- Holes cut for airflow and thermal safety

Figure 5: Laser-cut dual acrylic plates for PCB mounting and structural support

Upper Acrylic Plate

- Hosts custom PCB and Raspberry Pi 5
- M2 holes spaced to align with all component footprints

Camera Mount

An L-shaped acrylic column is fastened to the upper plate to mount the OpenMV camera. This setup ensures:

- Stable forward-facing field of view
- Unobstructed path for vision processing
- Easy vertical alignment using M2 screws

Figure 6: L-shaped laser-cut acrylic camera mount holding the OpenMV camera in forward-facing position

7 Conclusion and Future Work

7.1 Conclusion

The RoboRebound platform presented in this thesis demonstrates a significant leap in secure, real-time multi-robot coordination, particularly tailored for warehouse environments. The core motivation stemmed from the limitations of the legacy PIC32-based system, which exhibited severe constraints in throughput, latency, and cryptographic capability. By transitioning to a robust architecture built around STM32H753ZI microcontrollers, Raspberry Pi 5, and ESP32-C6 with Wi-Fi 6 connectivity, the system has substantially improved its communication bandwidth, actuation latency, and fault tolerance under adversarial conditions.

The original system, based on RF communication and TinkerBoard S, suffered from low wireless throughput (˜20 Kbps), lack of image processing capabilities, software-based cryptographic routines, and synchronization bottlenecks. Through our redesigned pipeline, RoboRebound now achieves:

- Image streaming at 2–3 frames per second via UART
- End-to-end SHA-256 verified sensor data transfer within 50 ms
- HMAC-authenticated command relays for secure actuation
- MQTT-based telemetry broadcasting at 4–8 Mbps on Wi-Fi 6

Moreover, the integration of hardware-based hashing (SHA-256) and keyed message authentication (HMAC-SHA256) allows each stage of the sensing-actuation loop to be validated cryptographically. This enforces not only trust in inter-node communication but also ensures bounded-time security without sacrificing performance.

The PCB design and power delivery strategy also play a central role. A dual-stage buck conversion system, capable of driving high-current loads like the Raspberry Pi and motor controller, ensures voltage stability across all components. Careful current budgeting for STM32 boards, sensors, and communication modules ensures thermal and electrical safety.

Structurally, the robot benefits from a modular two-layer acrylic frame, with designated mounting holes and laser-cut alignment features that promote robustness and serviceability. The addition of SPI-based coordination between A-node and S-node aligns with the original RoboRebound protocol design for secure polling, enhancing fault containment and redundancy.

From a security perspective, the system was evaluated using the STRIDE and DREAD models. Threats including spoofing, tampering, and replay attacks are successfully mitigated through timestamped packets, watchdog timeouts, and layered cryptographic validation. These safeguards, when combined with deterministic firmware design, create a platform suitable for real-world autonomous deployment in distributed robotic systems.

RoboRebound achieves its original goal: to vastly increase the throughput, reliability, and security of real-time robot coordination in warehouse settings. It stands as a blueprint for embedded multi-agent systems that must balance timing guarantees with adversarial resilience.

7.2 Future Work

While the current implementation marks a substantial improvement, several avenues exist for expanding the RoboRebound framework:

1. **Scalability Testing Across Larger Fleets**
   All current benchmarks are based on a small team of robots. Future tests should evaluate throughput, jitter, and packet loss when operating in environments with 10+ bots concurrently publishing and subscribing to MQTT topics. This would provide insights into scalability under Wi-Fi congestion and channel contention.

2. **OTA Firmware Updates and Key Management**
   Introducing secure over-the-air (OTA) firmware updates via the ESP32-C6 will enhance maintainability. Coupled with a key provisioning system, this could allow dynamic key rotation and revocation in case of node compromise.

3. **End-to-End Encryption with TLS and MQTT 5.0**
   While HMAC currently secures payload integrity, encryption of payload contents via TLS or hybrid cryptosystems (e.g., ECC-based key exchange + AES encryption) could further protect telemetry data from passive sniffing attacks.

4. **ROS2 Integration and Interoperability**
   Adapting the system for integration with ROS2 nodes would open pathways to broader adoption and compatibility with commercial fleets. DDS-based secure messaging over Wi-Fi 6E could be benchmarked against the current MQTT pipeline.

5. **Machine Learning at the Edge**
   The Raspberry Pi 5 enables deployment of lightweight object detection models (e.g., MobileNet or YOLOv5-nano). Integrating such models with obstacle classification could enhance autonomous decision-making without increasing latency significantly.

6. **Physical Tamper Detection and Secure Bootloaders**
   Adding sensors for physical tamper detection and secure boot mechanisms (e.g., STM32 TrustZone) would further harden the device against low-level compromise. A secure bootloader can ensure only signed firmware is executed on all STM32 nodes.

7. **Energy Profiling and Runtime Adaptation**
   The current system operates on fixed power profiles. Incorporating real-time energy monitoring and dynamic scheduling (e.g., slowing actuation when battery is low) would prolong operational time and enhance mission resilience.

8. **Multi-Hop Communication with Dynamic Topologies**
   The MQTT model currently assumes direct broker access. Introducing a mesh-based or peer-to-peer fallback mechanism (e.g., ESP-NOW or Thread) could enhance communication in scenarios with limited infrastructure or dynamic spatial layouts.

References

[1] N. Gandhi, Y. Cai, A. Haeberlen, and L. T. X. Phan. RoboRebound: Multi-Robot System Defense with Bounded-Time Interaction. In *Proceedings of the 20th European Conference on Computer Systems (EuroSys '25)*, 2025.

[2] H. Khalid, P. Kirisci, and M. Ali. Security framework for industrial collaborative robotic cyber-physical systems. *Computers in Industry*, 97:132–142, 2018.

[3] M. Gleirscher, N. Johnson, P. Karachristou, R. Calinescu, J. Law, and J. Clark. Challenges in the Safety-Security Co-Assurance of Collaborative Industrial Robots. *arXiv preprint arXiv:2007.11099*, 2020.

[4] V. Mayoral Vilches, L. Alzola Kirschgens, A. Bilbao Calvo, A. Hernández Cordero, R. Izquierdo Pisón, and D. Mayoral Vilches et al. Introducing the Robot Security Fra.mework (RSF), a standardized methodology to perform security assessments in robotics. *arXiv preprint arXiv:1806.04042*, 2018.

[5] Z. Rahman, I. Khalil, X. Yi, and M. Atiquzzaman. Blockchain-based Security Framework for Critical Industry 4.0 Cyber-physical System. *arXiv preprint arXiv:2106.13339*, 2021.

[6] F. Alshammari, Y. Luo, and Q. A. Chen. On the Cyber-Physical Security of Commercial Indoor Delivery Robot Systems. *arXiv preprint arXiv:2412.10699*, 2024.

[7] A. Shostack. *Threat Modeling: Designing for Security.* Wiley, 2014.

[8] L. Kohnfelder and P. Garg. The threats to our products. *Microsoft Interface*, 1999.

[9] M. Howard and D. LeBlanc. *Writing Secure Code (2nd ed.).* Microsoft Press, 2002.

[10] P. Slovic. Perception of Risk Posed by Extreme Events. *Decision Research*, 2000.