Mong Phan

Development of Custom Hardware and Software for Real-time Multi-robot System

Metropolia University of Applied Sciences

Bachelor of Engineering

Smart IoT Systems

Bachelor’s Thesis

April 28, 2025

Abstract

Author:	Mong Phan
Title:	Development of Custom Hardware and Software for Real-time Multi-robot System
Number of Pages:	58 pages + 1 appendix
Date:	April 28, 2025
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communication Technology
Professional Major:	IoT and Smart Systems
Supervisors:	Joeseph Hotchkiss, Senior Lecturer
	Daniel Korhonen, Project Engineer

The Raspberry Pi 5, a standard single-board computer, offers powerful performance and compact size for a variety of industrial and educational applications. However, the diverse requirements of these applications often demand features beyond its existing hardware design. This thesis addresses these challenges by designing a custom I/O board for the Raspberry Pi Compute Module 5 (CM5), a custom three-port switch board, and by developing an embedded application to control robot motion. The study aims to overcome the Raspberry Pi 5's limitations, offering an optimized solution for robotic applications.

The methodology encompassed both hardware and software development stages. Hardware design involved analyzing the official CM5 I/O and KSZ8794 Evaluation Board reference designs, providing insights for the creation of a new custom hardware solution. Software development focused on a Python-based embedded application to remotely control robots via a laptop using a PlayStation 4 (PS4) controller.

A series of tests was conducted to validate the work. The results indicated that the custom I/O board and switch board delivered strong performance, confirming the solution's efficacy within a three-host network communication setup.

This thesis demonstrates a reliable, cost-effective approach to multi-robot control, enhancing system efficiency. Future work could explore the implementation of Gigabit Ethernet for multi-camera streaming and further improve the hardware design to achieve even better performance. The study highlights the potential of customized embedded systems in advancing robotics.

Keywords: Raspberry Pi, Compute Module 5, custom I/O board, embedded application, motor control, Ethernet switch

Contents

List of Abbreviations

CM5:	Compute Module 5
CSA:	Current State Analysis
GPIO:	General-Purpose Input/Output
HDMI:	High-Definition Multimedia Interface
IO:	Input/Output
PCB:	Printed Circuit Board
PWM:	Pulse Width Modulation
RPi:	Raspberry Pi
USB:	Universal Serial Bus
WebSocket:	A communication protocol for real-time data exchange over Ethernet.
PS4 Controller:	PlayStation 4 game controller
GUI:	Grapthic User Interface
AC:	Alternating current
DC:	Direct current
NAT:	Network Address Translation
PoE:	Power over Ethernet

EMI:	Electromagnetic Interference
EDA:	Electronic Design Automation
DMM:	Digital Multimeter
CPU:	Central Processing Unit
GPU:	Graphic Processing Unit
RPM:	Revolutions Per Minute

1 Introduction

The rapid evolution of robotic systems has highlighted the need for compact and efficient control solutions, particularly in applications requiring coordinated operation within constrained environments. Standard single-board computers, such as the Raspberry Pi 5, offer versatility and performance for a range of uses; however, their fixed hardware designs often fall short of meeting specialized demands. Limitations in communication capabilities, expandability, and power management have driven the development of customized alternatives to enhance robotic functionality.

This thesis focuses on designing a custom I/O board for the Raspberry Pi Compute Module 5 (CM5), a custom three-port Fast Ethernet switch board, and developing an embedded application to enable remote control of robot motion. Tasks beyond this focus, such as robot chassis design, video streaming, or graphical user interface (GUI) development, are excluded. The custom I/O board was created by analyzing the official CM5 I/O board reference design, removing unnecessary components, repositioning the I/O ports, and integrating a 5V/5A step-down circuit. Similarly, the KSZ8794 evaluation board reference design was analyzed to develop a compact three-port network switch board to support communication among three hosts: two robots and a remote-control device. The embedded application, built using Python, processes PlayStation 4 (PS4) controller inputs via a laptop to control the robots' movements remotely.

The objective of this work is to address the Raspberry Pi 5's constraints and deliver an optimized solution for robotic applications requiring efficient hardware and robust network connectivity. Conducted as part of the Smart IoT Systems programme at Metropolia University of Applied Sciences, the project aims to enhance robotic control efficiency through customized embedded systems. The context involves a multi-robot environment that demands effective coordination, compact hardware, and dependable communication within limited physical space. The thesis contributes to the field by demonstrating the potential of tailored embedded hardware and software solutions for robotics.

This document is structured as follows: Section 2 explores the theoretical foundations of embedded systems and networked robotics underpinning the solution. Section 3 analyzes the current state of relevant technologies, identifying key limitations. Section 4 details the methods and materials employed in the project. Section 5 presents the proposed solution and its implementation. Section 6 discusses the results and their implications. Section 7 summarizes the findings and offers conclusions.

2 Theoretical Foundations of Embedded Systems and Networked Robotics

2.1 Power Electronics

Power electronics play a pivotal role in embedded systems, particularly in robotic applications where efficient voltage regulation, stable power delivery, and compact design are essential for reliable operation. In such systems, devices like microcontrollers and embedded Linux platforms, such as the Raspberry Pi Compute Module 5 (CM5), often require precise, low-voltage power supplies (e.g., 5V), while robotic setups typically operate on higher voltages (e.g., 24V) sourced from batteries or industrial power systems. A buck converter, a type of DC-DC step-down converter, addresses this need by transforming higher input voltages into lower, stable outputs, making it a critical component for integrating power-efficient solutions into compact embedded designs. This section provides a theoretical foundation for buck converter technology, covering its operating principles, topology, advantages, limitations, efficiency factors, output voltage ripple, thermal management, and layout considerations, with relevance to the custom CM5 I/O board design (Section 5.1). The discussion draws insights from general buck converter theory [6], efficiency analysis [4], ripple characteristics [5], and the TPS5450 buck converter datasheet [3].

2.1.1 Operating Principle and Topology

The fundamental operation of a buck converter relies on controlled energy transfer through periodic switching. A semiconductor switch, typically a MOSFET, is turned on and off using PWM to regulate the voltage applied to an inductor. When the switch is on, the inductor stores energy from the input voltage, increasing its current; when the switch is off, the inductor releases this energy to the load, producing a regulated output voltage lower than the input [6]. The key components of a buck converter include a high-side switch (MOSFET), a low-side switch (diode or MOSFET in synchronous designs), an inductor to smooth the current, and an output capacitor to filter the voltage, ensuring a stable DC output. Figure 1 illustrates a simplified buck converter topology, highlighting these core components.

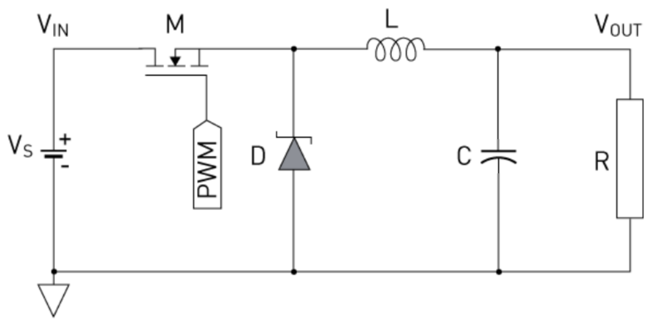


Figure 1: Simplified Buck Converter Topology [6].

As shown in Figure 1, the input voltage (V_{IN}) is applied to the switch (MOSFET), which connects to the inductor (L) and the diode (D). The inductor is connected to the output capacitor (C) and the load (R), with the output voltage (V_{OUT}) taken across the load. Figure 2 illustrates the current flow in a buck converter when the switch is turned on and off.

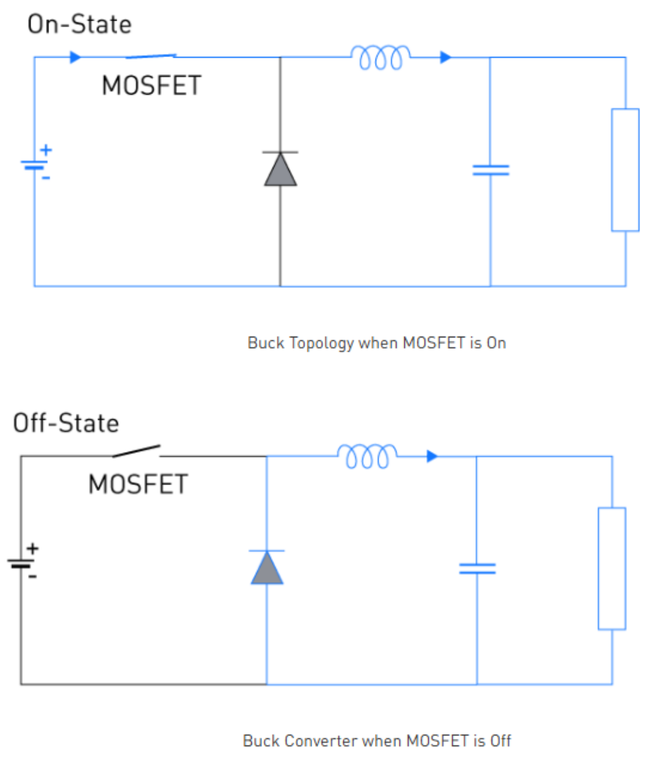

On-State
MOSFET
+
-
Buck Topology when MOSFET is On
Off-State
MOSFET
+
-
Buck Converter when MOSFET is Off

Figure 2: Energy conversion and control of a buck converter [6]

As illustrated in Figure 2, when the switch is on, current flows from V_{IN} through the inductor to the load, storing energy in the inductor's magnetic field. When the switch turns off, the inductor maintains current flow by releasing its stored energy through the diode to the load, with the output capacitor smoothing voltage fluctuations. This topology ensures efficient voltage conversion with a relatively simple design, making buck converters suitable for compact systems like the custom CM5 I/O board, where a 24V robotic supply is stepped down to 5V for the CM5.

The TPS5450, a 5A step-down converter used in this project, exemplifies this functionality, supporting input voltages up to 36V and delivering a stable 5V output, aligning with the CM5's power requirements [3]. Figure 3 presents a simplified schematic of a buck converter using the TPS5450, illustrating its practical implementation.

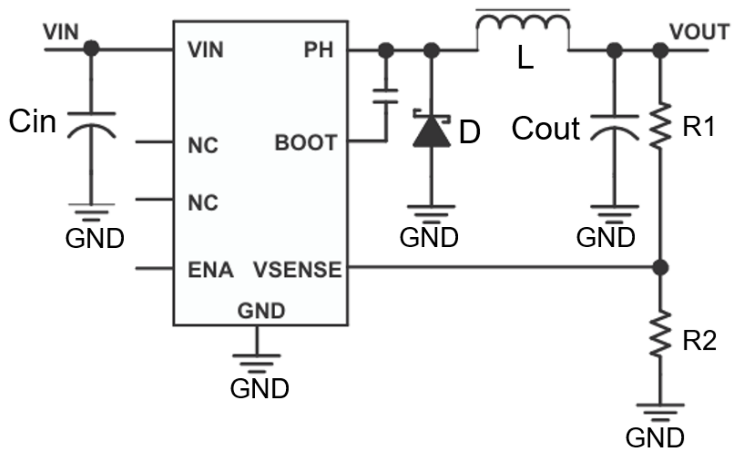


Figure 3: Simplified Schematic of a Buck Converter Using the TPS5450 [3].

As illustrated in Figure 3, the TPS5450 schematic highlights the core components of a buck converter circuit. The input voltage (VIN) is connected to the VIN pin of the TPS5450 IC, representing a higher voltage source, such as the 24V supply typical in robotic systems. An input capacitor (Cin), connected between VIN and ground (GND), stabilizes the input by filtering noise and voltage spikes, ensuring a steady supply to the TPS5450 chip. The TPS5450 IC, centrally positioned in the schematic, manages the voltage conversion process through its key pins: VIN receives the input voltage, the PH (phase) pin generates PWM signals, the VSENSE pin receives feedback voltage from the output to stabilize the output voltage, and the GND pin connects to the ground of the power supply. The inductor, placed between the PH pin and the output (VOUT), stores energy during the switching cycle and smooths the output current, a fundamental mechanism of buck converters. The freewheeling diode, connected between the PH pin and GND, provides a path for the inductor current when the internal MOSFET turns off, preventing voltage spikes and ensuring continuous conduction.

2.1.2 Advantages and Limitations

Buck converters offer several advantages that make them a preferred choice in embedded applications. They provide high efficiency in step-down operations, often exceeding 90%, due to minimal energy losses during switching [6]. Their compact design, requiring fewer components, suits space-constrained robotic systems, such as the custom CM5 I/O board. Additionally, buck converters exhibit fast transient responses, enabling them to handle dynamic loads, which is beneficial for multi-robot control systems (Section 5.2). They also produce relatively low output ripple, ensuring stable power delivery to sensitive components like microcontrollers [6]. However, buck converters are limited to step-down applications, restricting their use to scenarios where the output voltage must be lower than the input. They may face challenges with wide input voltage ranges, potential electromagnetic interference (EMI) from fast switching, and thermal management issues, necessitating careful design for robotic environments [6]. These advantages and limitations directly influenced the decision to integrate a buck converter into the custom CM5 I/O board, addressing the Raspberry Pi 5's lack of native high-voltage support (Section 3.1).

2.1.3 Efficiency Factors

The efficiency of a buck converter determines how effectively it converts a higher input voltage to a lower output voltage while minimizing power loss. Efficiency is particularly significant in robotic applications, where reduced power loss extends battery life and minimizes heat generation, supporting compact and reliable designs. Several factors influence the efficiency of a buck converter, impacting its performance in systems like the custom CM5 I/O board [4].

Switching Loss in the MOSFET: This occurs during the on-off transitions of the converter’s internal low-side and high-side MOSFETs. These losses increase with higher switching frequencies, as the MOSFETs switch more frequently, dissipating energy each cycle.

Conduction Loss in the MOSFET: This results from the on-resistance of the high-side and low-side MOSFETs. When current flows through these components, their inherent resistance causes power dissipation as heat.

Inductor Resistance Loss: This stems from the resistance within the inductor’s winding. As current passes through the inductor, this resistance generates heat, particularly at higher output currents, such as the 5A required by the CM5.

Gate Drive Loss: This arises from the energy needed to charge and discharge the gate capacitance of the MOSFETs during each switching cycle, becoming more pronounced at higher frequencies.

Dead Time Loss: This occurs during the brief delay between turning off one MOSFET and turning on the other to prevent short circuits, where the inductor current flows through the MOSFET’s body diode, leading to additional losses.

Capacitor ESR Loss: This is due to the equivalent series resistance (ESR) of the input and output capacitors, which dissipates power as current flows through them, potentially affecting stability under dynamic loads.

MOSFET Output Capacitance Loss: This occurs as the output capacitance of the high-side MOSFET charges and discharges during each switching cycle, dissipating energy.

IC Control Circuit Loss: This results from the power consumed by the chip's internal control circuitry, particularly noticeable at light loads.

Reverse Recovery Loss: This occurs in the freewheeling diode during the switching transition as it recovers from its conducting to non-conducting state, increasing with higher frequencies and currents.

These factors contribute to the overall efficiency of a buck converter. Figure 4 illustrates this relationship between efficiency and output current for the TPS5450 across various input voltages.

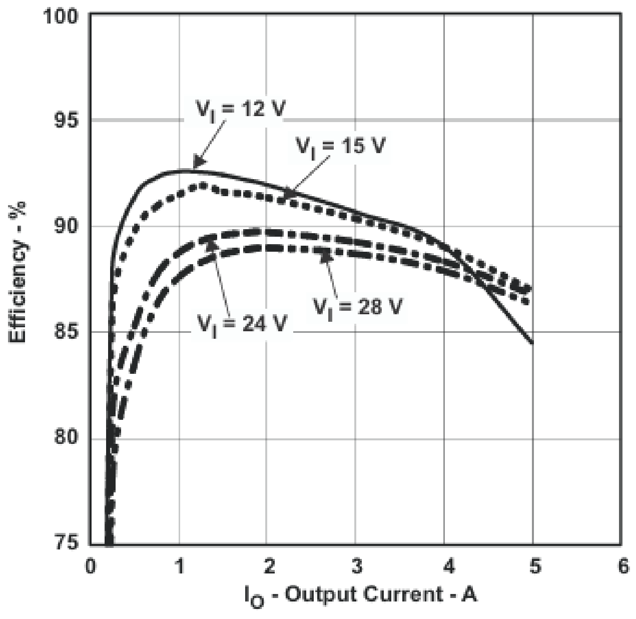


Figure 4: Efficiency vs. Output Current for the TPS5450 Buck Converter [3].

Figure 4 shows the relationship between efficiency and output current under specific conditions: an output voltage (Vo) of 5V, a switching frequency (fs) of 500 kHz, and an ambient temperature (TA) of 25°C, with input voltages (Vi) of 12V, 15V, 24V, and 28V. For the 24V input, which aligns with the robotic system's power supply in this project, the TPS5450 achieves its highest efficiency of approximately 90% at an output current (Io) of around 2A. At the maximum output current of 5A, required by the CM5, the efficiency decreases to approximately

87%, reflecting the impact of increased losses at higher loads [3], as discussed earlier.

2.1.4 Output Voltage Ripple

Output voltage ripple is the residual AC fluctuation on the DC output, a critical consideration for stable power delivery to sensitive components like the CM5. It primarily arises from the switching operation, where the switching action of the low-side and high-side MOSFETs creates a triangular waveform of current in the inductor, causing fluctuations in the voltage across the output capacitor [5]. The magnitude of the output voltage ripple is influenced by several factors, including the ESR and capacitance value of the output capacitor, the duty cycle, and the switching frequency of the buck converter chip, such as the TPS5450 [5]. Low-ESR capacitors reduce ripple by more effectively filtering the current, while high ESR increases ripple due to voltage drops, which also reduces efficiency [5]. Minimizing ripple is essential to ensure stable power delivery to components like microcontrollers and sensors, which can be disrupted by voltage fluctuations. Figure 5 shows the waveform of the output voltage ripple and phase node voltage of the TPS5450 at an output current of 5A.

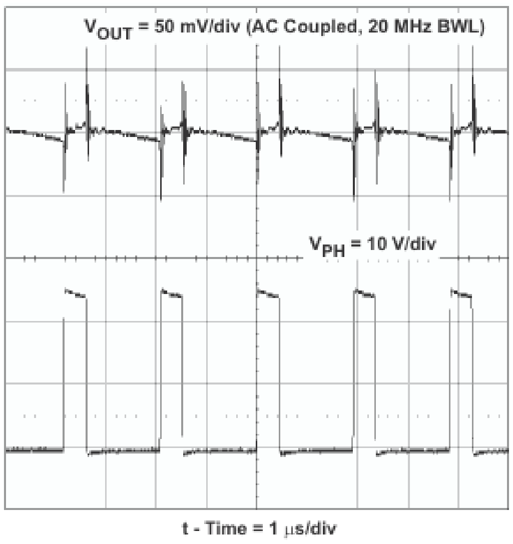


Figure 5: Output Voltage Ripple and Phase Node Waveform of the TPS5450 at Io = 5A [3].

As illustrated in Figure 5, the output voltage ripple (V_{OUT}) is specified at 50mV peak-to-peak with a 330µF output capacitor, a level suitable for the CM5's operation in robotic systems. The waveform also displays the phase node voltage (V_{PH}) at 10V per division, reflecting the switching action over a time scale of 1µs per division. The triangular waveform of the inductor current, driven by the switching at the phase node, directly contributes to the ripple observed in the output voltage, as explained earlier.

2.1.5 Thermal Management and Layout Considerations

Thermal management is crucial in buck converter design, as heat generation, especially at high currents, affects the durability of the entire electronic board. The TPS5450, housed in an SOIC PowerPAD package, can operate up to a 125°C junction temperature and relies on PCB copper area for heat dissipation [3]. At a 5A output, confined robotic enclosures, common in multi-robot setups, limit natural cooling, making layout design critical to avoid overheating without bulky active cooling solutions like fans. A large ground plane enhances thermal conductivity, while careful component placement minimizes heat buildup, influencing the compact design of the custom CM5 I/O board (Section 5.1).

Layout considerations, guided by datasheets like the TPS5450's, optimize performance and integration. Figure 6 shows the layout example of TPS5450 buck converter.

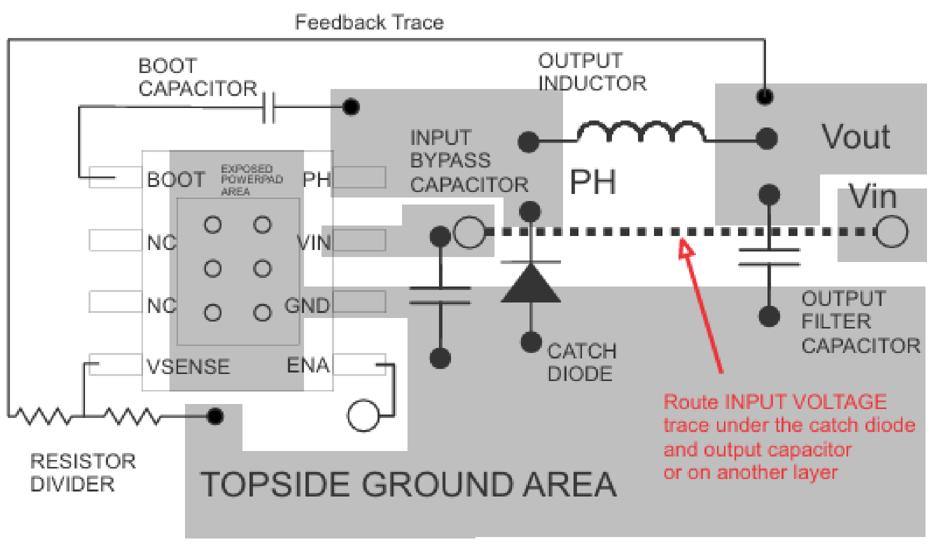


Figure 6: Layout example of TPS5450 buck converter [3].

As depicted in Figure 6, the input capacitor is placed close to the VIN and GND pins to reduce parasitic inductance, improving transient response and minimizing ripple [3]. A compact loop between the output inductor, capacitor, and switch node limits EMI, which is critical in networked robotics with Ethernet communication (Section 5.2). Short high-current paths reduce resistance losses, enhancing efficiency. These principles shaped the power stage layout of the custom CM5 I/O board, addressing the Raspberry Pi 5's fixed power constraints by integrating a tailored 5V/5A solution directly onto the board.

2.2 Network Communication

Network communication is the foundation of modern robotic systems, enabling seamless data exchange between devices to achieve coordinated operation, real-time control, and remote monitoring. In multi-robot systems, such as the one developed in this thesis, reliable and efficient communication is essential for synchronizing actions, transmitting control signals, and streaming video data. This section explores the theoretical foundations of network communication relevant to the custom CM5 I/O board design, focusing on Ethernet communication in robotics and the WebSocket protocol. These technologies underpin the three-host network connecting two robots and a remote control, addressing the Raspberry Pi 5’s connectivity limitations (Section 3.1) and supporting the embedded application for motor control and video streaming (Section 5.3).

2.2.1 Ethernet Communication in Robotics

Ethernet communication has become a vital technology in robotics, offering high-speed, reliable, and standardized data transfer for real-time control and coordination. Unlike wireless alternatives like Wi-Fi, Ethernet provides a stable, low-latency connection, making it ideal for applications where timing and reliability are critical, such as industrial automation and multi-robot systems. In robotics, Ethernet is often used to connect robots, controllers, and external devices, forming a network that supports data exchange for tasks like motion control, sensor feedback, and video streaming.

Ethernet’s suitability for robotics stems from its ability to handle high-bandwidth data with minimal latency. For instance, in industrial settings, Ethernet-based protocols like Ethernet/IP (EIP) enable real-time communication between programmable logic controllers (PLCs), robots, and control systems. A study on industrial communication highlights that EIP, built on the Common Industrial Protocol (CIP), supports seamless data exchange in cooperative robotic systems by ensuring interoperability and real-time control over Ethernet TCP/IP [7]. This is particularly relevant to this thesis, which integrates a three-port Fast Ethernet switch to create a three-host network connecting two robots and a remote control. Figure 7 illustrates this network topology, emphasizing Ethernet’s role in facilitating communication.

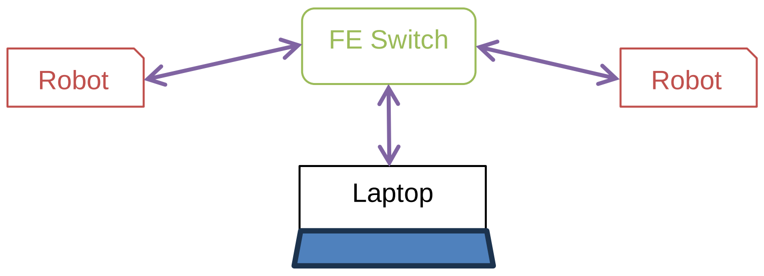


Figure 7: Ethernet Network Topology for a three-host Robotic System.

Figure 7 illustrates the Ethernet network topology for the three-host robotic system developed in this project. The custom three-port Fast Ethernet switch board, operating at 100 Mbps, serves as the central hub, connecting Robot 1, Robot 2, and the remote control. Each device is linked via a dedicated Ethernet connection, enabling low-latency data exchange for motor control signals, speed feedback, and video streaming.

Ethernet communication in robotics also offers scalability and flexibility. In a multi-robot system, a single Ethernet network can connect multiple devices, allowing them to share data efficiently. For example, a study on real-time communication for robotic control compared protocols like FINS, CIP, UDP, and OPC over Fast Ethernet (100 Mbps), finding that Ethernet-based protocols can achieve low-latency data exchange suitable for real-time robot control [8]. However, Ethernet in robotics has challenges, such as the need for careful network design to manage bandwidth and avoid congestion, especially when streaming video data alongside control signals.

2.2.2 WebSocket Protocol Theory

The WebSocket protocol is a standardized communication technology that enables real-time, bidirectional data exchange between a client and a server over a single, persistent TCP connection. Unlike traditional HTTP, which follows a request-response model, WebSocket provides full-duplex communication, allowing both the client and server to send data at any time without the overhead of repeated HTTP requests [11]. This makes WebSocket particularly suitable for applications requiring continuous, low-latency data transfer, such as the embedded application in this project, which processes PS4 controller inputs, reports motor speeds, and streams video data between the robots and the remote control.

The WebSocket protocol, standardized by the Internet Engineering Task Force (IETF) in 2011 as RFC 6455, operates over HTTP ports 80 and 443, ensuring compatibility with existing web infrastructure [9]. The connection begins with an opening handshake message, where the client sends a HTTP request with an "Upgrade: websocket" header, and the server responds with a "101 Switching Protocols" status, upgrading the connection to WebSocket [11]. Figure 8 depicts the WebSocket communication process, highlighting the transition from an HTTP handshake to a persistent, bidirectional WebSocket connection.

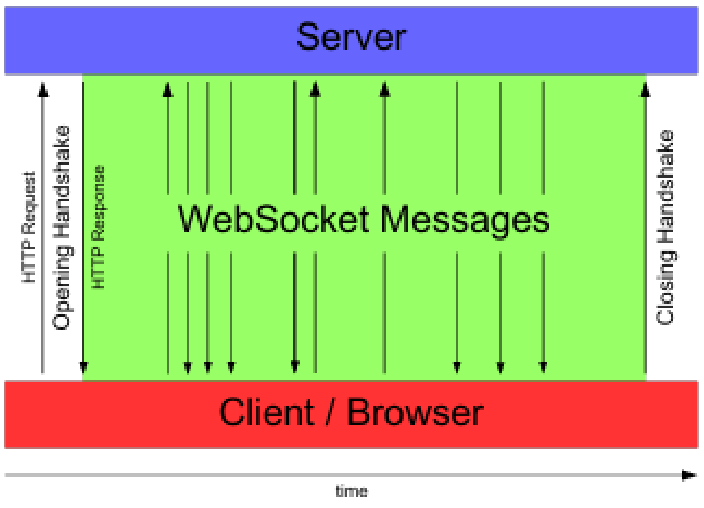


Figure 8: WebSocket communication flow [11].

Figure 8 illustrates the WebSocket connection flow between a client (robot) and a server (remote control). Initially, the client initiates an HTTP request containing the "Upgrade: websocket" header, marking the Opening Handshake. The server responds with an HTTP response confirming the protocol switch ("101 Switching Protocols"), establishing the WebSocket connection. Once established, WebSocket messages are exchanged in full-duplex mode, allowing continuous real-time communication between the client and server. This persistent connection ensures efficient data transfer, such as sending PS4 controller inputs and receiving motor feedback or video streams. Finally, when communication is no longer needed, the Closing Handshake terminates the WebSocket session.

WebSocket also offers scalability and security features relevant to robotic applications. It can handle multiple concurrent connections efficiently, which is useful for multi-robot systems where several devices need to communicate simultaneously [10]. For security, WebSocket Secure (WSS) uses Transport

Layer Security (TLS) to encrypt data, protecting against eavesdropping and tampering, which is critical in networked robotics to ensure safe operation [10]. However, WebSocket has limitations, such as the need for a stable network to maintain the persistent connection and potential compatibility issues with older systems that do not support the protocol [9]. In this project, these challenges are mitigated by using a wired Fast Ethernet network, ensuring reliability, and leveraging modern Python libraries (e.g., socket) that support WebSocket communication (Section 4.2).

2.3 Motor Control

Motor control is a critical aspect of robotic systems, enabling precise movement, speed regulation, and synchronization of mechanical components. In multi-robot applications, such as the one developed in this thesis, effective motor control ensures coordinated operation, efficient power usage, and reliable performance within constrained environments. This section explores the theoretical foundations of motor control, focusing on Brushless DC (BLDC) motors equipped with Hall-effect sensors, which are used in the project to drive two robots. The discussion specifically addresses Hall-effect sensor-based BLDC motors, as opposed to sensorless BLDC motors, covering their construction, working principles, commutation methods, control techniques, and the role of PWM in motor speed control, supported by illustrative figures to clarify key concepts.

Brushless DC (BLDC) Motors

BLDC motors are synchronous motors powered by direct current (DC) that operate without the mechanical brushes and commutators found in traditional DC motors. Instead, they rely on electronic controllers to manage the current flow in the motor windings, offering significant advantages over brushed DC motors, such as higher efficiency, longer lifespan, and reduced maintenance. These characteristics make BLDC motors well-suited for robotic applications requiring precise and reliable motion control.

Construction and Working Principle

A BLDC motor consists of two primary components: a stator and a rotor, with an electronic speed controller (ESC) as a crucial external element. The stator (the stationary part) is made of laminated steel cores with slots that house copper windings, typically arranged in a three-phase configuration connected in a star or delta pattern. The rotor (the rotating part) contains permanent magnets, often made of high-strength materials like neodymium, arranged in alternating north and south poles. The number of poles on the rotor can vary, typically ranging from two to eight, depending on the motor’s design and application requirements [12]. The ESC, connected externally, controls the current flow to the stator windings, ensuring proper operation of the motor.

The operation of a BLDC motor relies on the interaction between the magnetic fields generated by the stator windings and the rotor’s permanent magnets. When the ESC energizes the stator windings in a specific sequence, a rotating magnetic field is created. This field attracts and repels the rotor’s permanent magnets, producing a torque that causes the rotor to rotate as it aligns with the stator’s field. To sustain continuous rotation, the ESC electronically commutates the current through the windings, keeping the magnetic field ahead of the rotor. This electronic commutation eliminates the need for mechanical brushes, reducing wear, friction, and electrical noise compared to brushed DC motors [13]. Figure 9 illustrates the basic construction of a BLDC motor, highlighting the stator, rotor, and their interaction.

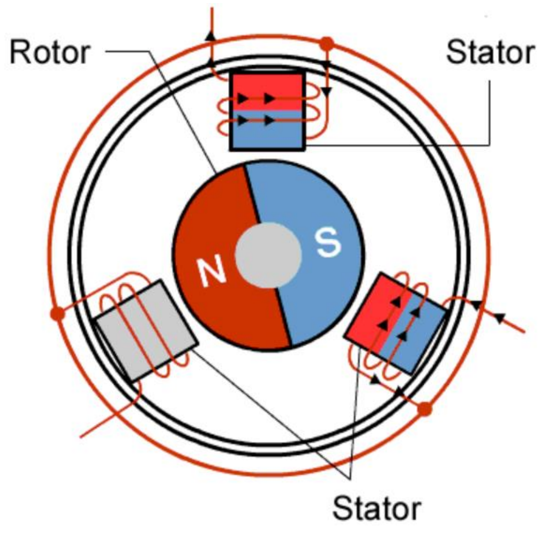


Figure 9: Construction of a BLDC Motor [12].

Figure 9 shows a cross-sectional view of a three-phase BLDC motor. The stator (outer ring) consists of steel laminations with three sets of windings, spaced 120 degrees apart. Arrows within the windings indicate the direction of current flow, which generates a magnetic field. The rotor (inner circle) has permanent magnets with alternating north (N) and south (S) poles, in this case, a two-pole configuration (one N-S pair). The magnetic field produced by the current in the stator windings interacts with the rotor magnets to produce torque. This figure clarifies the structural arrangement and the basis for the motor's operation.

Commutation

Commutation in a BLDC motor involves switching the current through the stator windings to maintain a rotating magnetic field that drives the rotor. Unlike brushed DC motors, which rely on mechanical commutation, BLDC motors use electronic commutation, requiring precise knowledge of the rotor's position to energize the correct windings at the appropriate time. This is achieved using Hall-effect sensors mounted on the stator to detect the rotor's magnetic pole positions. Three Hall sensors, labeled Hall Sensor A, Hall Sensor B, and Hall Sensor C, are placed 120^o apart. Each sensor produces a binary signal (1 for high, 0 for low) based on the polarity of the rotor's magnetic field (north or south), forming a 3-bit code (e.g., 100, 101) that changes every 60^o as the rotor rotates. This code indicates the rotor’s position within a 60^o sector, allowing the electronic speed controller (ESC) to determine the appropriate commutation step [17].

The commutation method used in this project is the six-step (or trapezoidal) commutation, where two of the three phases (A, B, C) are energized at a time, while the third phase remains unpowered. This creates a magnetic field that rotates in 60^o steps, completing one revolution (360^o) in six steps. The ESC uses the Hall sensor code to select which phases to energize, ensuring the magnetic field leads the rotor by approximately 90^o for maximum torque production. Tables 1 and 2 show the commutation sequences for both clockwise (CW) and counterclockwise (CCW) rotation [17], respectively, mapping each Hall sensor state to the corresponding phase energization.

Table 1: Commutation Sequences for Clockwise Rotation.

Hall Sensor A	**Hall Sensor B**	**Hall Sensor C**	**Phase A**	**Phase B**	**Phase C**
1	0	0	$-V_{DCB}$	$+V_{DCB}$	NC
1	0	1	NC	$+V_{DCB}$	$-V_{DCB}$
0	0	1	$+V_{DCB}$	NC	$-V_{DCB}$
0	1	1	$+V_{DCB}$	$-V_{DCB}$	NC
0	1	0	NC	$-V_{DCB}$	$+V_{DCB}$
1	1	0	$-V_{DCB}$	NC	$+V_{DCB}$

Table 2: Commutation Sequences for Counterclockwise Rotation.

Hall Sensor A	**Hall Sensor B**	**Hall Sensor C**	**Phase A**	**Phase B**	**Phase C**
1	0	0	$+V_{DCB}$	$-V_{DCB}$	NC
1	1	0	$+V_{DCB}$	NC	$-V_{DCB}$
0	1	0	NC	$+V_{DCB}$	$-V_{DCB}$
0	1	1	$-V_{DCB}$	$+V_{DCB}$	NC
0	0	1	$-V_{DCB}$	NC	$+V_{DCB}$
1	0	1	NC	$-V_{DCB}$	$+V_{DCB}$

Tables 1 and 2 defines the phase energization for each Hall sensor state, enabling the ESC to control the direction of rotation. In the clockwise (CW) sequence, for example, when the Hall sensors output a state of 100, the ESC applies $-V_{DCB}$ to Phase A and $+V_{DCB}$ to Phase B, with Phase C unpowered (NC), causing current to flow from Phase B to Phase A and creating a magnetic field at a specific angle. As the rotor rotates and the Hall state changes to 101, the ESC switches to apply $+V_{DCB}$ to Phase B and $-V_{DCB}$ to Phase C, with Phase A unpowered (NC), shifting the magnetic field by 60^o. This sequence repeats every 60^o, rotating the magnetic field clockwise to drive the rotor in the same direction. For counterclockwise (CCW) rotation, the phase energization is reversed for the same Hall states; for instance, at Hall state 100, the ESC applies $+V_{DCB}$ to Phase A and $-V_{DCB}$ to Phase B, with Phase C unpowered (NC), reversing the current flow and thus the direction of the magnetic field. This process ensures smooth rotation by aligning the magnetic field with the rotor's position, allowing the motor to operate in either direction as needed [17]. Accurate placement of the Hall sensors is critical to ensure reliable position detection, as misalignment can lead to incorrect commutation and reduced motor efficiency [17].

Speed control

To control the speed of a BLDC motor, the ESC employs a closed-loop control system that adjusts the voltage applied to the stator windings using PWM signals. The speed control principle is based on comparing the desired speed ($\omega_{desired}$) with the actual speed (ω_{actual}) of the motor, which is determined using feedback from the Hall sensors. The Hall sensors provide rotor position information, and the time between their state transitions is used to calculate the actual speed (ω_{actual}). The difference between the desired and actual speeds, known as the speed error ($\omega_{error} = \omega_{desired} - \omega_{actual}$), is processed by a speed controller to adjust the PWM duty cycle, which regulates the average voltage applied to the motor windings, thereby controlling the motor's speed [17]. Figure 10 illustrates the closed-loop speed control principle for a BLDC motor.

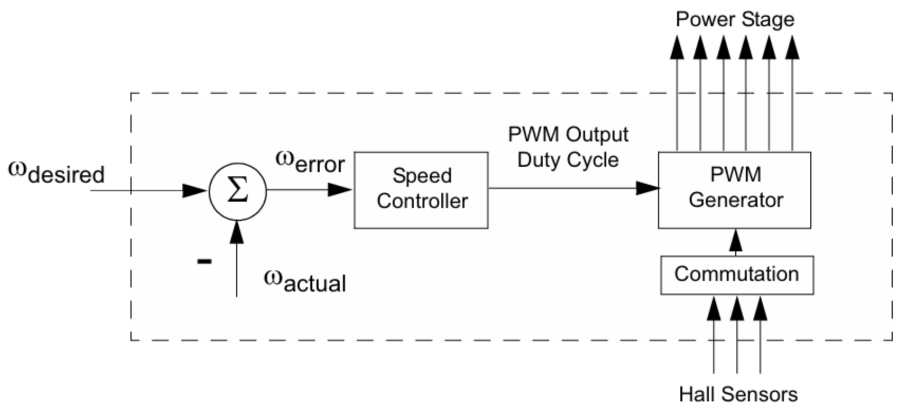


Figure 10: Closed-loop speed control principle for a BLDC motor [17].

As illustrated in Figure 10, the desired speed ($\omega_{desired}$) and actual speed (ω_{actual}) are compared to generate a speed error (ω_{error}), which is fed into the Speed Controller. The Speed Controller adjusts the PWM duty cycle based on the error, and the PWM Generator produces PWM signals that are combined with commutation signals from the Hall sensors. These signals are sent to the Power Stage, which drives the motor. The Hall sensors provide feedback on the rotor's position, enabling the calculation of the actual speed (ω_{actual}).

PWM is a technique that varies the width of pulses in a square wave signal to control the average power delivered to a load, such as a motor. By adjusting the duty cycle (the proportion of time the signal is high within a fixed period) the ESC can regulate the effective voltage supplied to the motor windings. For example, a 25% duty cycle results in a lower average voltage and slower speed, while a 75% duty cycle increases the average voltage, leading to a higher speed. PWM signals are typically generated at a frequency suitable for the application, often in the range of 1 kHz to 20 kHz, to ensure smooth motor operation and minimize noise [16]. In BLDC motor control, PWM is applied to the energized phases during each commutation step, allowing precise speed regulation while maintaining the commutation sequence determined by the Hall sensors [14][17]. Figure 11 illustrates a PWM signal with varying duty cycles and its effect on motor speed.

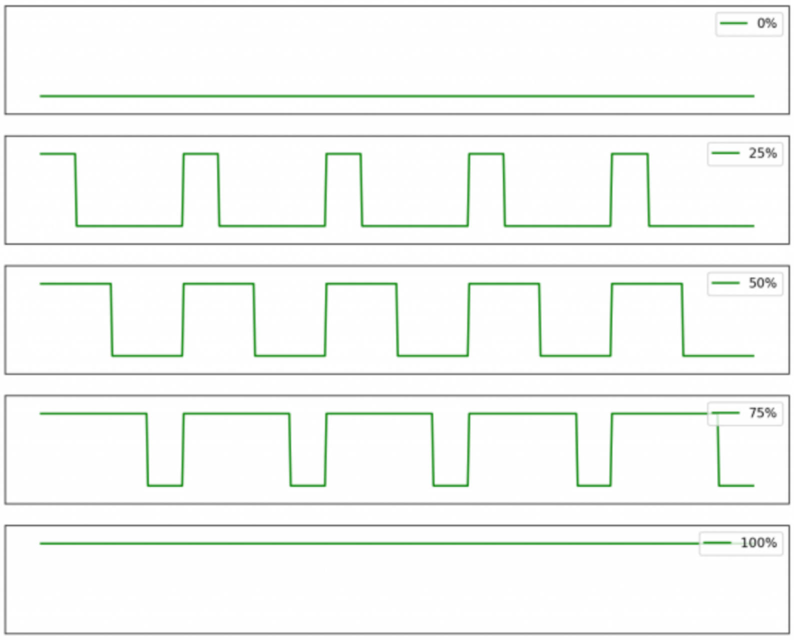


Figure 11: PWM Signal with Varying Duty Cycles [16].

Figure 11 illustrates five PWM waveforms, each representing a different duty cycle over a fixed time period, demonstrating how the duty cycle affects the signal's behavior and the resulting motor speed. The x-axis represents time, and the y-axis represents voltage (e.g., 0V to 5V). The first waveform, with a 0% duty cycle, remains at the low state (0V) throughout the entire period, indicating the signal is always off. The second waveform, with a 25% duty cycle, is in the high state (5V) for 25% of the period and in the low state for the remaining 75%, resulting in a lower average voltage. The third waveform, with a 50% duty cycle, spends equal time in the high and low states (half the period at 5V and half at 0V) forming an ideal square wave. The fourth waveform, with a 75% duty cycle, is in the high state for 75% of the period and the low state for 25%, delivering a higher average voltage. The fifth waveform, with a 100% duty cycle, remains in the high state (5V) for the entire period, indicating the signal is always on and providing the maximum average voltage. The duty cycle directly influences the motor's speed: a 0% duty cycle means the motor stops (0% of maximum speed), a 50% duty cycle results in the motor running at 50% of its maximum speed, and a 100% duty cycle drives the motor at its maximum speed. This relationship allows precise control of motor speed in robotic applications, where varying the duty cycle adjusts the power delivered to the motors, enabling smooth acceleration, deceleration, or stopping as needed.

3 Current State Analysis

3.1 Official Raspberry 5 Computer Board

The Raspberry Pi 5 is a powerful and compact single-board computer, equipped with a comprehensive set of I/O ports and GPIO pins suitable for numerous industrial and educational applications. Figure 12 illustrates the Raspberry Pi 5 board, highlighting all its I/O ports and key components.

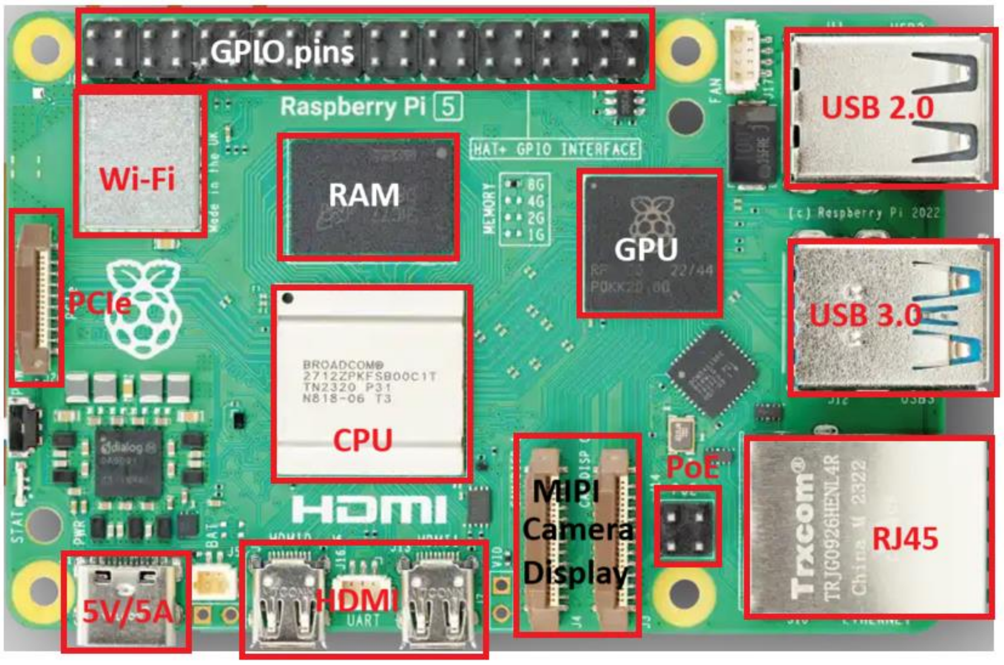


Figure 12: The Raspberry Pi 5 board, showcasing its ports and connectors [1]

As illustrated in Figure 12, the CPU of the Raspberry Pi 5 is a Broadcom BCM2712 quad-core 64-bit Arm Cortex-A76 processor, operating at a clock speed of 2.4GHz. This processor, supported by cryptography extensions, 512KB per-core L2 caches, and a 2MB shared L3 cache, provides 2-3 times the performance of the Raspberry Pi 4, enabling efficient handling of demanding tasks. The board features an 800MHz VideoCore VII Graphic Processing Unit (GPU), supporting dual 4Kp60 display output through two micro HDMI ports, which enhances its capability for multimedia and visualization applications [2]. The Raspberry Pi 5 deploys LPDDR4X-4267 SDRAM with options of 2GB, 4GB, 8GB, and 16GB, suitable for various budget applications.

Connectivity is a key strength, with two high-speed USB 3.0 ports and two USB 2.0 ports, alongside a Gigabit Ethernet port with Power over Ethernet (PoE) support via a four-pin connector. Dual-band 802.11ac Wi-Fi and Bluetooth 5.0 with Bluetooth Low Energy (BLE) further expand wireless options. A single-lane PCI Express 2.0 interface, accessible through an FFC connector, supports high-bandwidth peripherals such as M.2 SSDs. Main storage is facilitated by a microSD card slot with SDR104 mode, doubling peak performance compared to previous models. The two four-lane MIPI camera/display transceivers support up to two cameras or displays. Additional features include a 40-pin GPIO header, a USB-C power input requiring 5V/5A, a physical power button, and connectors for an RTC battery and a fan with PWM control.

Despite its advanced features, the Raspberry Pi 5 exhibits several limitations that hinder its suitability for specialized applications, particularly in multi-robot systems requiring compact, efficient control within constrained environments. The board’s physical dimensions, approximately 85mm x 56mm, remain unchanged from the Raspberry Pi 4, posing challenges for integration into space-restricted robotic chassis where additional components such as motors, cameras, and wiring must coexist. This fixed form factor and layout limit flexibility for custom configurations, often requiring external adapters or HATs that increase bulk and complexity.

Power management presents another constraint. The Raspberry Pi 5 requires a 5V/5A power supply via a USB-C port. While this may benefit some applications where high-power wall adapter is more suitable, it lacks an integrated solution for higher-voltage inputs (e.g., 24V) common in robotic systems, requiring external step-down converters that add size and inefficiency.

Connectivity, while robust, falls short for networked robotics. The single Gigabit Ethernet port restricts direct multi-device communication, such as the three-host network needed for two robots and a remote control, forcing reliance on external switches that increase the complexity. The PCIe 2.0 interface, HDMI, and USB ports are often unnecessary in many robotic applications. Using an SD card as the main storage is not a versatile solution in some situations, such as when operating in mobile environments like cars or robots.

The 40-pin GPIO header, while versatile and providing more I/O for control, increases the board size, and only some of the pins may be truly necessary for specialized applications. The current placement of the two MIPI interfaces, accessed via FFC connectors, is not optimal for camera connectivity.

These limitations underscore the need for a more adaptable and efficient hardware solution, particularly in robotic applications with space, power, and networking constraints. Leveraging the CM5's modular design, the proposed solution optimizes I/O port placement, integrates a 5V/5A step-down converter for streamlined power management, and incorporates a three-port Fast Ethernet switch to facilitate multi-device communication without external adapters.

3.2 Ethernet Communication in Robotics

Ethernet communication in robotics has become a cornerstone for enabling real-time, reliable data exchange in multi-robot systems, particularly in applications requiring coordination, such as collaborative tasks or remote monitoring. In the context of this project, the goal is to establish a three-host network comprising two robots and a remote-control system, where the robots communicate with each other and the remote control to share video streams and control commands. The Raspberry Pi 5, a popular single-board computer for robotics projects due to its improved processing power and connectivity options, is the selected platform for this setup. However, implementing Ethernet-based communication in this configuration creates several challenges, especially with the Raspberry Pi 5.

Ethernet offers several benefits for robotic communication, making it a preferred choice over wireless alternatives like Wi-Fi or Bluetooth in certain scenarios. It provides high-speed, low-latency communication, which is critical for real-time applications such as video streaming and precise motor control. Ethernet also ensures a stable and reliable connection, unlike Wi-Fi, which can suffer from interference or signal degradation in environments with many devices or physical obstructions. Additionally, Ethernet supports higher bandwidth, allowing for the transmission of large data packets, such as high-definition video feeds, which is essential for the project’s requirement of streaming video from the robots’ cameras to the remote control.

While Ethernet is advantageous, setting up a three-host network with two robots and a remote control using the Raspberry Pi 5 presents specific challenges due to its hardware limitations.

The Raspberry Pi 5 is equipped with a single Ethernet port, supporting a maximum throughput of 1 Gbps. However, in a three-host network setup, direct Ethernet connections between the two robots and the remote control are impossible with the Raspberry Pi 5, highlighting its limitation in such a system.

To address the constraint of having only one Ethernet port, an attempt was made to add a second port using the **PCIe to Gigabit ETH USB 3.2 HAT+** for the Raspberry Pi 5. This HAT provides an additional Gigabit Ethernet port (recognized as eth1) alongside the built-in port (eth0), enabling direct connections and potentially eliminating the need for an external switch.

However, despite efforts to configure both ports to operate on the same network, a suitable solution could not be found within the project’s limited timeframe. The built-in Ethernet port and the HAT Ethernet port were treated as separate network interfaces, making it difficult to integrate them into a single LAN where all devices could communicate seamlessly. Configuring the network to bridge the interfaces or assign them to the same subnet proved complex, likely due to the intricacies of network management in Raspberry Pi OS and potential compatibility issues between the HAT’s chipset and the Pi 5’s network stack.

Due to these challenges, the approach of using the Pi HAT was abandoned in favor of another solution. A three-port switch, designed as a separate board, connects the two robots and the remote control on the same LAN, ensuring all hosts share the same subnet.

This solution simplifies network configuration, as the switch handles packet routing at the hardware level, reducing the Raspberry Pi 5’s processing burden. Each Raspberry Pi 5 on the robots uses its built-in Ethernet port to connect to the switch, while the remote control (e.g., a laptop) connects to the third port.

While this setup resolves the networking issue, it introduces additional hardware, adding system complexity, power requirements (the switch typically requires a 5V/1A power supply), and physical footprint; all of which are potential drawbacks in a mobile robotic system where space and power are limited.

3.3 Key Issues identified

The evaluation of the Raspberry Pi 5 in an Ethernet-based multi-robot system has revealed several challenges that impact its practical implementation. These issues stem from hardware limitations, network constraints, and system design complexities, all of which affect its suitability for real-time robotic applications. The following key issues were identified:

Fixed Hardware Design Constraints

The Raspberry Pi 5 comes equipped with a wide range of I/O ports to support diverse applications, including 2 x USB 2.0, 2 x USB 3.0, 2 x micro HDMI, 1 x Gigabit Ethernet port, 1 x PCIe connector, 2 x CAM/DSI connectors, and a 40-pin GPIO header. While this comprehensive design is sufficient for many general-purpose applications, it presents limitations in specialized projects where specific port configurations are required.

In space-constrained applications, such as robotics, the fixed placement and presence of redundant ports can lead to suboptimal hardware utilization and integration challenges. Components that are unnecessary for the project take up valuable space, while the inflexibility of port positioning can complicate hardware layout and cable management. These constraints make it difficult to fully optimize the Raspberry Pi 5 for custom applications requiring a more compact or tailored design.

Communication Limitations

The Raspberry Pi 5 features a single Gigabit Ethernet port, which restricts direct multi-device communication in a three-host network setup involving two robots and a remote control. As detailed in Section 3.2, an attempt to add a PCIe to Gigabit Ethernet HAT was unsuccessful due to network stack limitations and integration challenges with Raspberry Pi OS. This forced reliance on an external Ethernet switch, which introduces hardware dependencies, additional points of failure (e.g., switch malfunctions, cable disconnections), and increased network complexity.

Power Management Challenges

The 5V/5A power requirement of the Raspberry Pi 5 poses a significant limitation for battery-powered robotic systems. Most mobile robots operate at higher voltages (12V or 24V), necessitating a separate 5V power supply or step-down converters, which add weight, reduce efficiency, and increase system complexity. Additionally, the Ethernet switch further raises power demands, impacting battery runtime and overall system stability. These constraints make the Pi 5 less suitable for extended autonomous operation in mobile robots.

4 Methods and Material

This section outlines the methods and materials used to design and develop the custom hardware solution for the multi-robot system based on the Raspberry Pi CM5 module. The hardware design process addresses the limitations identified in the Current State Analysis (Section 3) by developing a tailored I/O board and integrating a custom three-port Ethernet switch board to support a three-host network connecting two robots and a remote control. The following subsections describe the hardware design approach, including analysis of reference designs and relevant datasheets and the tools employed in this project.

4.1 Hardware Design Methods

The hardware design process for this project involved analyzing two key reference designs: the official Raspberry Pi CM5 I/O Board, which guided the development of a custom CM5 I/O board, and the Microchip KSZ8794 Evaluation Board, which informed the creation of a three-port Ethernet switch board tailored to the specific requirements of the multi-robot system. This analysis ensured the successful development of both custom boards with a focus on efficiency and reliability. The design process was supported by various tools, including EasyEDA for schematic capture and PCB layout, a digital multimeter (DMM) for electrical testing, and an oscilloscope for signal analysis.

4.1.1 Analysis of the CM5 I/O Board Reference Design and CM5 Datasheet

The hardware design began with the official Raspberry Pi CM5 I/O Board reference design, provided by Raspberry Pi. This board acts as a motherboard for the CM5 and provides peripheral I/O connections, serving as a blueprint for accessing the module’s features. The CM5, announced in late 2024, is a compact system-on-module featuring the same Broadcom BCM2712 processor configuration as the Raspberry Pi 5, as described in Section 3. However, unlike the Pi 5, the CM5 does not include any built-in I/O ports or connectors; instead, its features are accessed through two 100-pin board-to-board connectors. Figure 13 illustrates the top and bottom side of the official Raspberry Pi CM5.

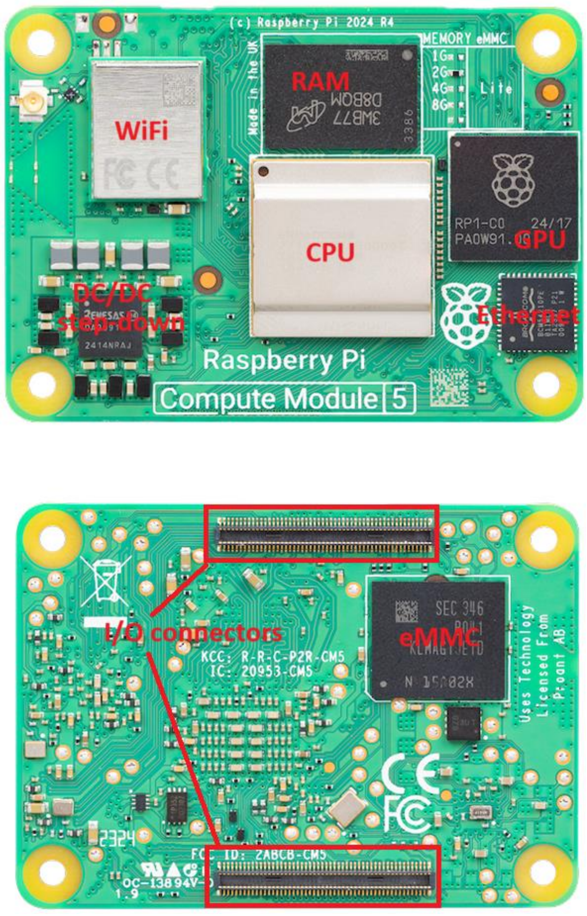


Figure 13: The official Raspberry Pi CM5 board [18].

As shown in Figure 13, the top side of the CM5 contains key components, including the CPU, GPU, RAM, WiFi module, Ethernet PHY, and the onboard DC/DC step-down circuitry for power regulation. These integrated components make the CM5 a self-contained compute unit. The bottom side of the module reveals the eMMC storage and two 100-pin high-density board-to-board connectors, which serve as the sole interface for accessing the CM5’s functionality. These connectors provide access to power, USB, HDMI, MIPI CSI/DSI, Ethernet, GPIO, and other critical I/O signals. The modular architecture allows for flexible hardware integration, enabling custom motherboards tailored to specific application needs. This design approach is particularly beneficial for embedded and robotics applications, where space, power, and connectivity requirements vary significantly.

To facilitate development and prototyping, Raspberry Pi provides an official CM5 I/O Board, which demonstrates the full range of the module’s capabilities. The official CM5 I/O Board is designed to host the CM5 and exposes a wide range of interfaces. Figure 14 depicts the official Raspberry Pi CM5 I/O Board with its I/O ports.

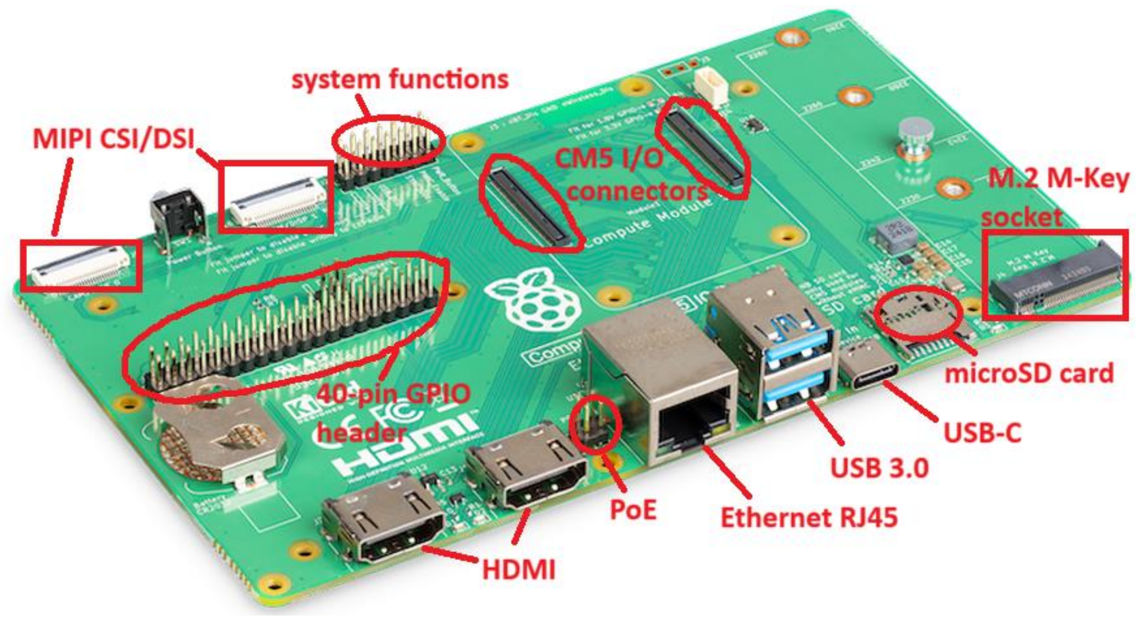


Figure 14: The official Raspberry Pi Compute Module 5 I/O Board[19].

As illustrated in Figure 14, the CM5 I/O board includes two full-size HDMI ports, a Gigabit Ethernet port, two USB 3.0 ports, an M.2 M-key PCIe socket, dual four-lane MIPI interfaces for cameras or displays, a microSD card slot, and a 40-pin GPIO header. The board also includes a USB-C connector for 5V/5A power input and a four-pin PoE header for Power over Ethernet (PoE) support.

The analysis focused on the detailed signal connections between the CM5 and its I/O ports, specifically, which pins from the two 100-pin connectors are used for power supply, ground, USB, HDMI, camera/display interfaces, Ethernet, and GPIO signals. The CM5 datasheet further provided detailed insights into the module’s pinout, signal purposes, and design recommendations, such as impedance guidelines for differential signal pairs (e.g., MIPI, HDMI, Ethernet) to preserve signal integrity for high-speed data lines.

While the official CM5 I/O Board is versatile, it was determined to be overly generic and included many unnecessary features for this project. For instance, the two full-size HDMI ports and dual USB 3.0 ports occupy considerable space and are redundant for the system’s intended robotic use. The PCIe and microSD card sockets were also considered unnecessary, as the robot utilizes the CM5’s built-in eMMC flash for data storage and boot devices. The USB-C connector is similarly unused, as the system is powered by a dedicated 5V/5A step-down converter.

The 40-pin GPIO header, though flexible, takes up significant space and includes many pins that are not needed in the project. Only a specific subset of GPIOs is required for motor control. Additionally, the onboard Gigabit Ethernet port requires an RJ45 connector, which increases the board’s footprint and may result in unstable connections in a mobile robotic environment.

4.1.2 Analysis of the KSZ8794 Evaluation Board Design and KSZ8794 Datasheet

To support reliable multi-host communication within the multi-robot system, a three-port Fast Ethernet (FE) switch board was required. The design of this switch board was informed by a detailed analysis of the Microchip KSZ8794 Evaluation Board and its associated datasheet. The KSZ8794 is an integrated Layer 2 managed Ethernet switch IC that supports up to three 10/100 Mbps MAC/PHY ports and one configurable 10/100/1000 Mbps RGMII/MII/RMII interface [20]. It also offers two operational modes: unmanaged and managed. These features make it an ideal candidate for implementing a compact, power-efficient three-port Ethernet switch tailored for embedded system applications.

The evaluation board provided a practical reference for hardware integration, including the recommended power supply configurations, signal routing, and layout recommendation. Special attention was given to the reference layout of the Ethernet PHYs and the magnetics interface, which are critical for maintaining signal integrity and minimizing EMI, especially in mobile robotic platforms.

The KSZ8794 datasheet was also thoroughly reviewed to understand the functional block diagram, pinout description, strap-in pins options, and electrical characteristics. This documentation also provided essential guidance for designing power rails (3.3V I/O and 1.2V core supply).

Together, the evaluation board and datasheet analysis formed the foundation for the custom three-port switch board design. The resulting hardware design aims to deliver reliable Ethernet communication between two robot units and a remote control station, while maintaining a compact form factor and low power consumption, consistent with the overall goals of the multi-robot system architecture.

4.1.3 Tools Used

The hardware design process utilized several tools to ensure accuracy, functionality, and reliability:

EasyEDA: EasyEDA, a cloud-based EDA (Electronic Design Automation) tool, was used for schematic capture and PCB layout design. The schematic for the custom CM5 I/O board was created by importing the official CM5 I/O reference design files (designed with KiCad EDA tool) and modifying them to reflect the changes. The PCB layout was designed to minimize the board’s footprint, with careful routing of high-speed signals (e.g., Ethernet MII, MIPI lanes) to reduce crosstalk and EMI. EasyEDA’s design rule check (DRC) and 3D visualization features helped verify the board’s dimensions and component placement, ensuring compatibility with the robotic chassis.

Digital Multimeter (DMM): A Fluke 77 DMM was used during prototyping to verify power supply voltages (e.g., 5V/5A from the step-down converter, 3.3V and

1.2V for the KSZ8794) and check for short circuits or open connections on the PCB. The DMM was also used to measure the current draw of the I/O board under load, ensuring it stayed within the project's power budget (e.g., <25W total for the CM5, I/O board, and Ethernet switch board).

Load Tester: An electronic load tester was used to apply a 5A load to the output of the TPS5450 converter, evaluating the performance and stability of the integrated step-down circuit.

Oscilloscope: A OWON SDS1102 dual channel oscilloscope was employed to verify the PWM signals generated by the CM5 for motor control.

Thermometer: A TENMA 72-7712 thermometer was used to monitor the temperature of the TPS5450 chip during the step-down capacity test.

4.2 Software Development Methods

The software developed for this project was designed as a Python-based embedded system that supports control and communication between two robots and a remote control unit over a three-host Ethernet network.

To meet the real-time demands of robotic control and monitoring, the software architecture employed different concurrency strategies on each side. The robot-side software utilized a real-time task scheduling approach (pyRTOS), while the remote-control side relied on a multithreading technique. The pyRTOS method deployed on the robot-side offered significant advantages over the multithreading approach used on the remote-control side, as it enabled smoother and more precise control of motor speed, including acceleration and deceleration. These strategies enabled concurrent handling of motor control, user input, and communication tasks, ensuring responsive system behavior and effective coordination across devices.

Python was selected as the programming language due to its simplicity, readability, strong support within the Raspberry Pi ecosystem, and the availability of rich libraries suited for embedded, networking, and hardware interfacing tasks. The development environment was based on Visual Studio Code (VS Code), which provided a powerful and flexible platform for editing and debugging.

Multiple Python libraries were used to streamline development:

- **pyRTOS** for real-time task management on the robot side
- **gpiozero** and **rpi_hardware_pwm** for GPIO and hardware PWM control
- **socket** and **struct** for Ethernet-based communication and data combination
- **pygame** for reading inputs from the PS4 controller connected to the remote control.

The communication protocol was implemented over Ethernet using WebSocket, selected for its reliability and ability to maintain low-latency, full-duplex communication between the devices. The application's architecture ensured consistent data exchange for command and feedback messages within millisecond-scale intervals.

5 Proposed Solution and Implementation

5.1 Custom CM5 I/O Board Design

Based on the earlier analysis, a custom CM5 I/O board was designed to optimize the system for this project, prioritizing a compact form factor, integration of a 24V DC power supply, and inclusion of only essential connectivity. The redesign process involved removing unnecessary I/O ports, adding required components, and repositioning interfaces to improve signal integrity and cable management, making the board more suitable for the robotic application.

Removing Unnecessary I/O Ports

Several ports from the official CM5 I/O design were removed to reduce the board's footprint and complexity. The two full-size HDMI ports were replaced with a single micro HDMI port, and only one USB 3.0 port was retained for initial system configuration. The PCIe socket, USB-C port, microSD card slot, and 4-pin PoE header were also removed. The Ethernet connector was replaced with an 8-pin header, allowing a cable to be directly soldered to the three-port Ethernet switch board. Finally, the 40-pin GPIO header was replaced with a smaller six-pin header, retaining only the essential pins needed for motor control.

Adding Required Components

To address the power management challenges identified in Section 3.3, an integrated 5V/5A step-down circuit was added to the custom I/O board. This converter, based on the Texas Instruments TPS5450 buck converter, accepts a 24V DC input (common in robotic battery systems) and efficiently steps it down to 5V/5A. It powers the CM5, the three-port Ethernet switch, and other components, eliminating the need for an external converter. The TPS5450 was selected for its high efficiency (up to 87%) and wide availability, minimizing power loss and heat generation compared to linear regulators.

Repositioning I/O Ports

The goal of this redesign was to create a compact custom CM5 I/O board while keeping signal traces as short as possible. Figure 15, a 3D model rendered using EasyEDA, shows the final custom CM5 I/O board used in this project.

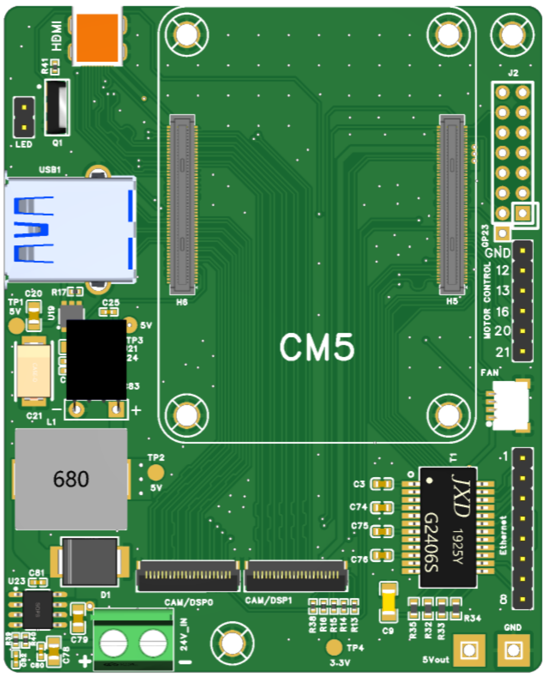


Figure 15: Final custom CM5 I/O board.

As shown in the Figure 15, the dual four-lane MIPI interfaces were retained for camera support but repositioned to the lower edge of the board, allowing shorter FFC cables to connect to robot-mounted cameras, thereby reducing signal interference and simplifying assembly. The single USB 3.0 port, LED control circuit, and micro HDMI port were placed near the top-left corner of the board to provide short signal paths to the CM5 connectors. All power circuits were laid out along the left edge of the board to minimize electromagnetic interference (EMI). Additional features, including the Ethernet header, motor control GPIOs, fan connector, and other system function pins, were placed along the right edge to facilitate direct and clean connections to motor drivers and the Ethernet switch.

5.2 Custom three-port Fast Ethernet switch board Design

Based on the analysis of the KSZ8794 Evaluation Board and datasheet, a compact and low-power three-port Ethernet switch board was developed to enable reliable three-host communication within the multi-robot system. Figure

16, a 3D model rendered using EasyEDA, shows the final custom three-port Ethernet Switch board used in this project.

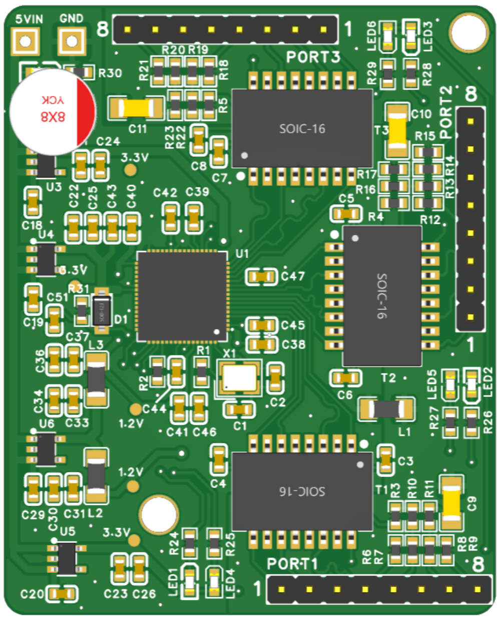


Figure 16: Final custom Ethernet Switch board.

As illustrated in Figure 16, all components were placed in close proximity to create a compact board layout. Rather than adopting the step-down regulators used in the evaluation board, the design employed the HE9703 series power converter due to its availability and simplified integration. Only three 10/100 Mbps MAC/PHY ports of the KSZ8794 were utilized, as the RGMII/MII/RMII interface and SPI control interface were excluded to operate the switch in unmanaged mode. To conserve board space and ensure secure Ethernet connections in a mobile robotic environment, standard RJ45 connectors were replaced with three 8-pin headers, allowing Ethernet cables to be directly soldered onto the board.

5.3 Embedded Application Development

The embedded software for this project was developed in Python and deployed across two main platforms: the robot units and the remote control. Each side featured a dedicated architecture optimized for its role in the three-host network system. The robot-side software was designed for real-time motor control and speed feedback, while the remote control-side software handled user input and overall coordination. This section presents the actual implementation of both components, using architectural and algorithm diagrams to explain their structures and workflows.

5.3.1 Robot-Side Software Architecture and Implementation

Each robot was responsible for receiving movement commands, controlling brushless motors accordingly, and sending RPM feedback to the remote control. Figure 17 illustrates the software architecture deployed on the robot side.

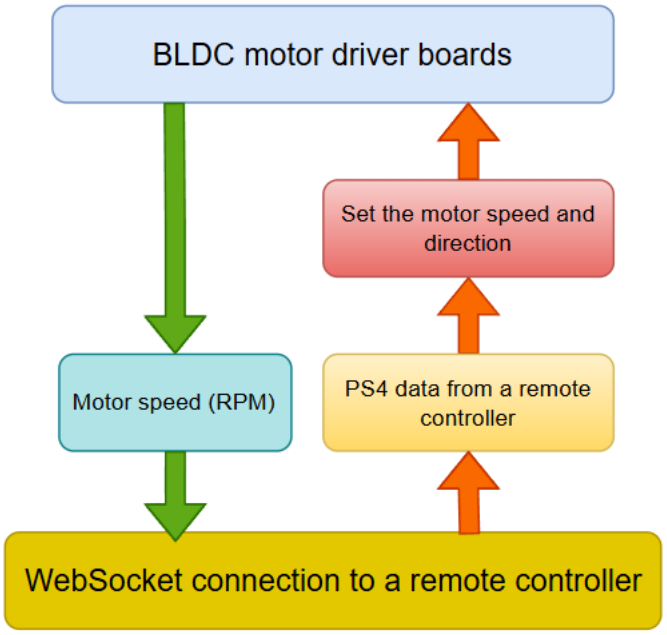

BLDC motor driver boards
Set the motor speed and direction
Motor speed (RPM)
PS4 data from a remote controller
WebSocket connection to a remote controller

Figure 17: Robot Software Architecture.

As shown in Figure 17, the robot-side software architecture diagram demostrates the key modules and their interactions. The system received PS4 controller data from the remote control via a WebSocket connection. This data included joystick inputs and directional commands, which were unpacked and interpreted by the robot’s onboard software. Based on this input, the system set the motor speed and direction using PWM signals sent to the BLDC motor driver boards.

In return, the motor driver boards provided real-time speed feedback in RPM, which was read by the robot through a GPIO-connected frequency generator

(FG) pin. The measured speed was then packed and sent back to the remote control over the same WebSocket connection. This closed-loop communication allowed the robot to be controlled and monitored in real time, ensuring responsive and coordinated movement across the multi-robot system.

Figure 18 presents the software algorithm deployed on the robot side using a Python-based pyRTOS approach.

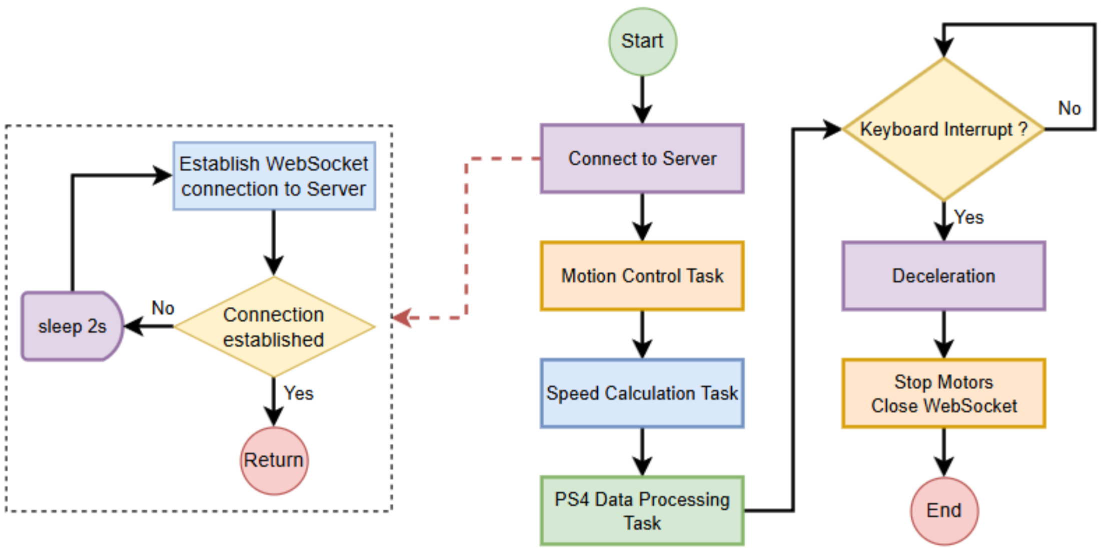


Figure 18: Software Deployed at the Robot Side.

As illustrated in Figure 18, the program began by attempting to establish a WebSocket connection to the server using a specified IP address and TCP port. If the connection was successful, the system proceeded to initialize three concurrent tasks: the Motion Control Task, Speed Calculation Task, and PS4 Data Processing Task, managed through the pyRTOS framework.

The program continuously checked for a keyboard interrupt. If detected, it initiated a deceleration sequence, safely stopped the motors, closed the WebSocket connection, and then terminated the application.

In case the WebSocket connection failed, the system entered a 2-second sleep interval before retrying the connection, ensuring robust reconnection logic for real-time remote control.

While running, the Motion Control, Speed Calculation, and PS4 Data Processing tasks were executed independently, thanks to the pyRTOS scheduling mechanism, allowing the program to operate in real time.

Motion Control Task

The Motion Control Task was responsible for managing the robot's movement based on data received from the PS4 controller. Figure 19 illustrates the software algorithm implemented for this task.

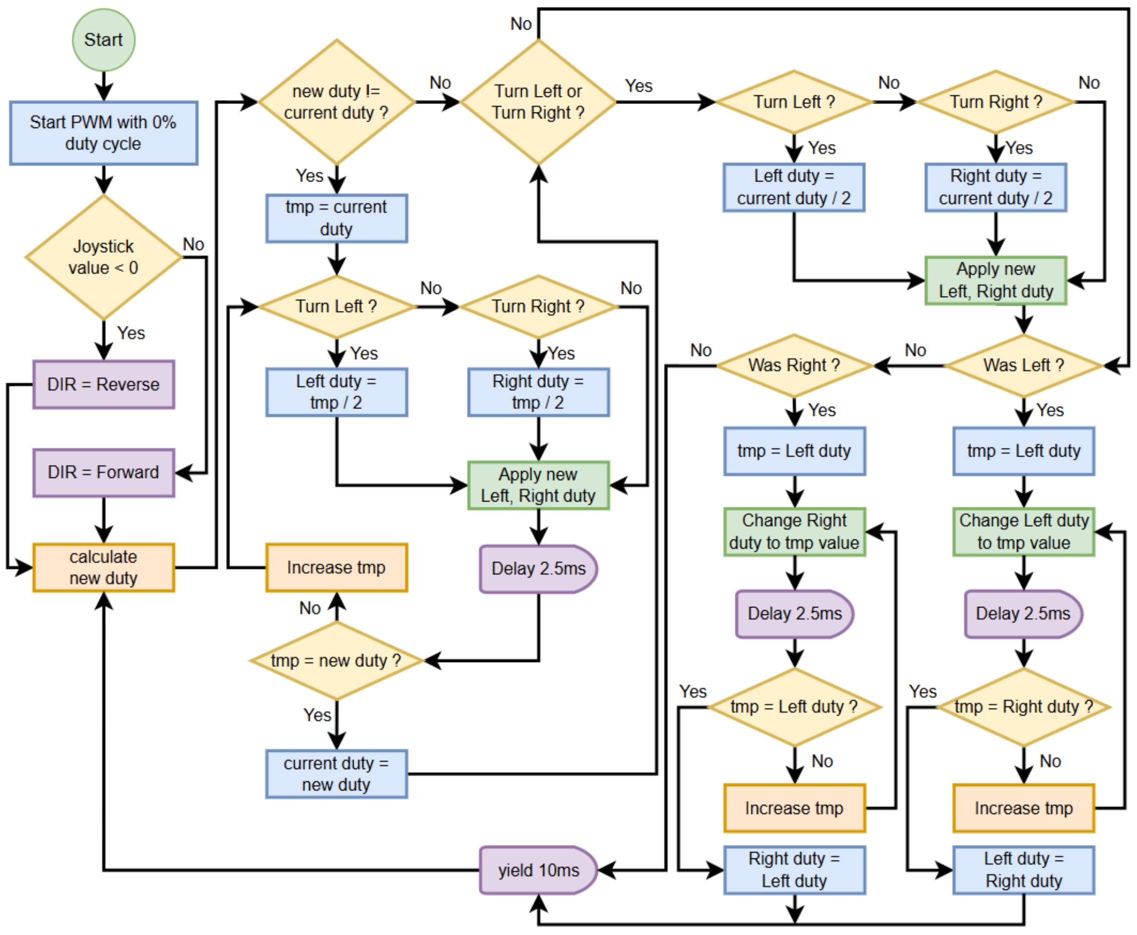


Figure 19: Motion Control Task Software Algorithm.

As shown in Figure 19, the flowchart outlines the algorithm used in the Motion Control Task, which controls the robot's motion by adjusting the speed and direction of its motors according to input from the PS4 controller. The process begins by initializing the PWM signals with a 0% duty cycle, ensuring that the robot remains stationary until a command is received.

The algorithm first evaluates the joystick value to determine the direction of motion. If the value is negative (indicating a backward command) the DIR pins are set to move the robot in reverse; otherwise, they are set for forward movement. Next, the program calculates a new duty cycle based on the absolute value of the joystick input, which determines the motor speed.

The algorithm then checks whether the new duty cycle differs from the current one. If they are different, the process enters a gradual adjustment loop to accelerate or decelerate the robot accordingly. A temporary variable (tmp) is initialized with the current duty cycle and is then incrementally adjusted until it matches the new duty cycle. During each step, the left and right motor speeds are updated based on this temporary value, followed by a brief delay (2.5ms) to smooth the transition.

During this adjustment process, the algorithm also checks the "Turn Left" and "Turn Right" conditions to determine whether the robot needs to turn. If a left turn is commanded, the left motor’s duty cycle is set to half the temporary value; if a right turn is commanded, the right motor’s duty cycle is halved. These new values are then applied to the motors.

Once the temporary value reaches the new duty cycle, the current duty cycle is updated accordingly.

If the new and current duty cycles are equal, the algorithm bypasses the acceleration phase and directly checks for turning commands. If a turn is detected, the corresponding motor’s duty cycle is halved and applied immediately.

Regardless of whether a new turn command is issued, the algorithm also checks if the robot was previously turning. If so, it gradually adjusts the motor speeds back to equal values using a similar incremental process.

Finally, the algorithm pauses for 10 milliseconds (yield 10ms) to allow other tasks to execute in the multitasking environment before looping back to check for new inputs from the PS4 controller. This cycle repeats continuously, ensuring responsive and smooth motor control based on real-time user input.

Speed Calculation Task

The speed of the motor, measured in RPM, was recorded and transmitted to the Remote Control via a WebSocket connection. Figure 20 depicts the software algorithm implemented to capture, calculate, and send the motor's speed to the Remote Control in real time.

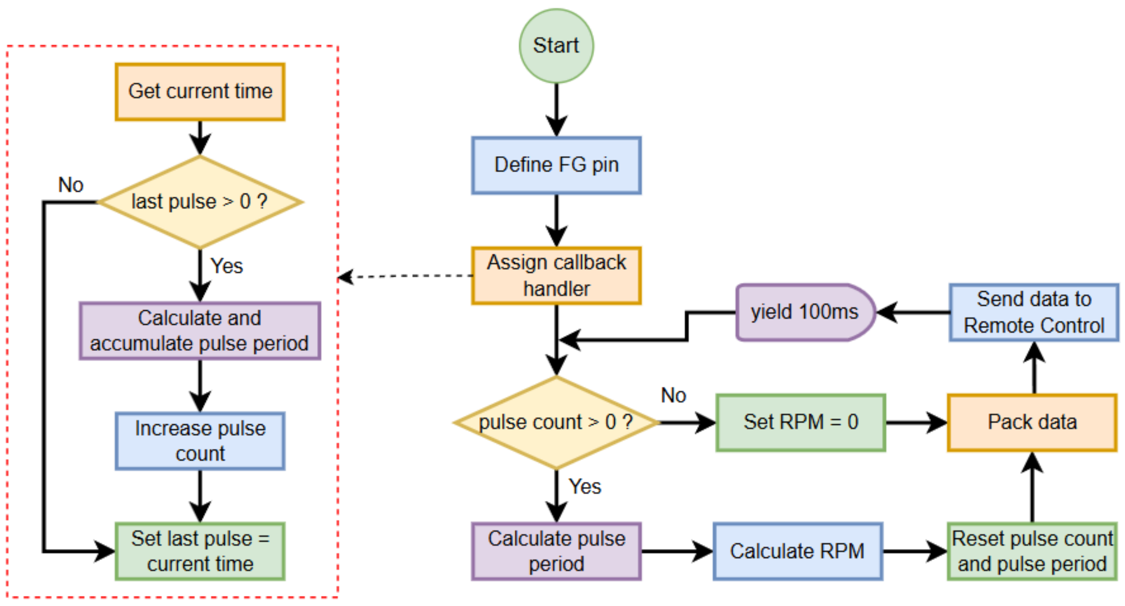


Figure 20: Speed Calculation Task Software Algorithm

As illustrated in Figure 20, the algorithm begins by defining the FG (frequency generator) pin and assigning a callback handler that responds to each pulse generated by the motor's speed sensor. Every time a pulse is detected, the callback routine is triggered.

Within the callback (highlighted by the red dashed box), the system retrieves the current timestamp. If a previous pulse has already been recorded (last pulse >

0), it calculates and accumulates the pulse period based on the time difference between the current and previous pulses. The pulse count is then incremented, and the current timestamp is stored as the last pulse value.

Meanwhile, the main loop checks every 100 milliseconds whether any pulses have been counted. If the pulse count is greater than zero, it calculates the average pulse period and converts this into an RPM value. If no pulses were counted during the interval, the RPM is set to zero, indicating that the motor was stationary.

The resulting RPM value, along with other relevant data, is then packed and sent to the Remote Control. After transmission, the pulse count and accumulated pulse period are reset to prepare for the next measurement cycle.

This loop continues periodically, providing an accurate and up-to-date measurement of motor speed for real-time monitoring and control.

PS4 Data Processing Task

The PS4 Data Processing Task was responsible for receiving and interpreting control data transmitted from the PS4 controller. This data, sent from the Remote Control, included joystick values and directional commands, which were used to manage the robot’s motion behavior. Figure 21 illustrates the software algorithm designed to handle this incoming data stream.

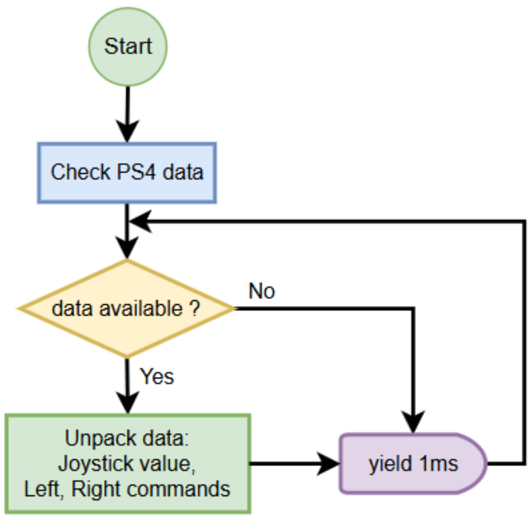


Figure 21: PS4 Data Processing Task Software Algorithm

As shown in Figure 21, the algorithm begins by continuously checking for incoming data from the PS4 controller. This is done by polling the communication interface for any available data packets.

If no data is available, the algorithm yields for 1 millisecond and resumes polling. If data is available, the algorithm proceeds to unpack the information, which includes the joystick value as well as directional commands for turning left or right.

After unpacking, the loop restarts to continuously monitor the PS4 controller for new inputs. This ensures that the robot responds in near real-time to user commands issued via the PS4 controller.

5.3.2 Remote Control-Side Software Architecture and Implementation

The remote control was responsible for interfacing with the PS4 controller to obtain control commands, sending these commands to the robots, processing the received motor speed values (in RPM), converting them to linear velocity (in m/s), and displaying the results on a graphical user interface (GUI). Figure 22 illustrates the software architecture of the remote control-side application, which was executed on a laptop functioning as the server within the three-host Ethernet network.

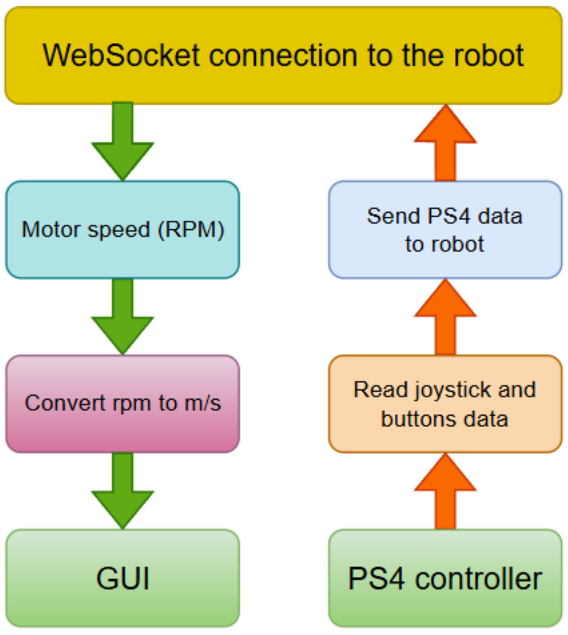


Figure 22: Remote Control-Side Software Architecture.

As illustrated in Figure 22, the diagram represents the software architecture implemented on the remote control side, highlighting the key components and data flows involved in controlling the robots and displaying feedback. The architecture is divided into two primary data flows. The first flow (green arrows) focuses on receiving and processing motor speed data from the robots, converting the values into linear velocity in meters per second (m/s), and displaying them for user monitoring. The second flow (orange arrows) manages the control input, wherein the PS4 controller's inputs are captured and transmitted to the robots. The communication between the remote control and the robots was implemented using the WebSocket protocol.

The software execution began with the initialization of the server, as illustrated in Figure 23.

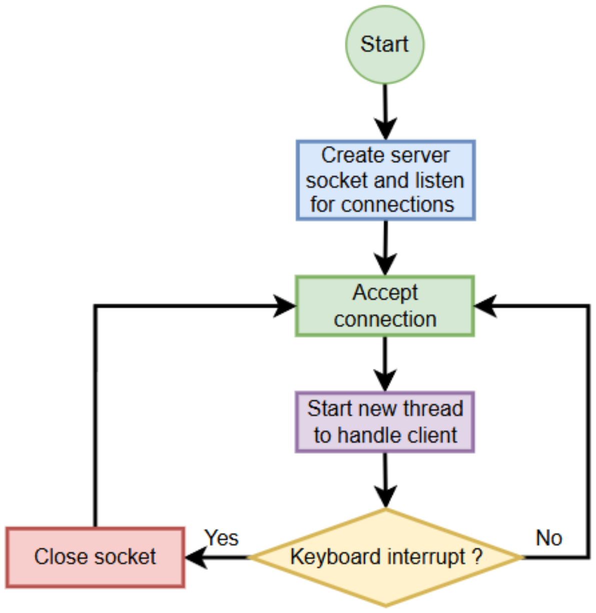

Start
Create server socket and listen for connections
Accept connection
Start new thread to handle client
Keyboard interrupt ?
Yes
No
Close socket

Figure 23: Server Startup flowchart.

As shown in Figure 23, the process starts with the creation of a server socket that listens for incoming connections. Once a connection request is received, a new thread is spawned to handle communication with the connected client. This allows the server to concurrently manage multiple clients. The server continues to accept new connections in a loop until a keyboard interrupt is detected, at which point the server gracefully shuts down by closing the socket.

When a client connected to the server, a new thread was created to handle the communication process, as depicted in Figure 24.

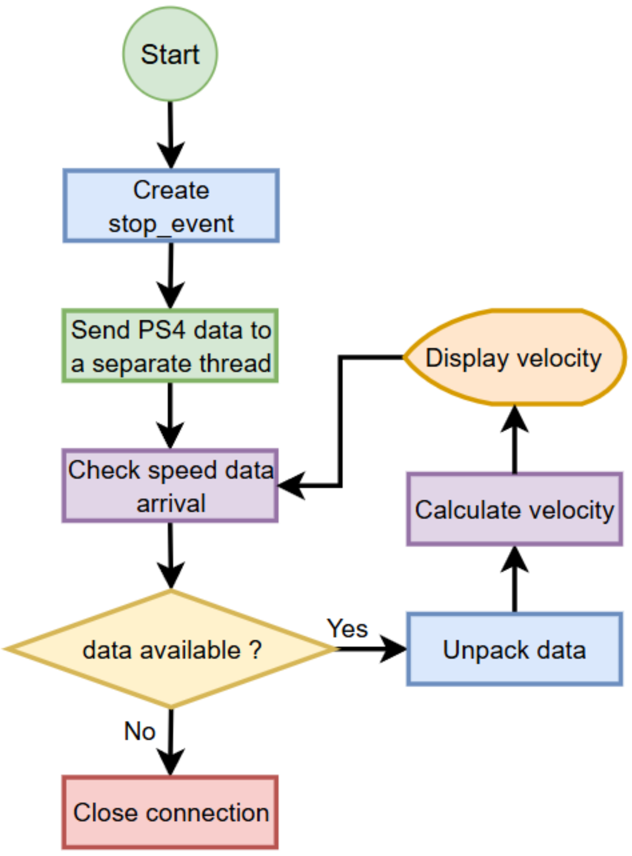


Figure 24: Client Handling Thread Flowchart.

As illustrated in Figure 24, the thread begins by creating a stop_event, which is used to manage the termination of the communication loop. It then sends PS4 controller data to a separate thread for processing before sending it to the robot. The main loop continuously checks for the arrival of speed data from the robot. When data becomes available, the thread unpacks the received information, calculates the corresponding linear velocity, and displays the result on the user interface. If no data is available, the connection is closed and the thread terminates.

Before sending PS4 data to the robot, a separate thread was created to continuously transmit joystick input data to the robot, as depicted in Figure 25.

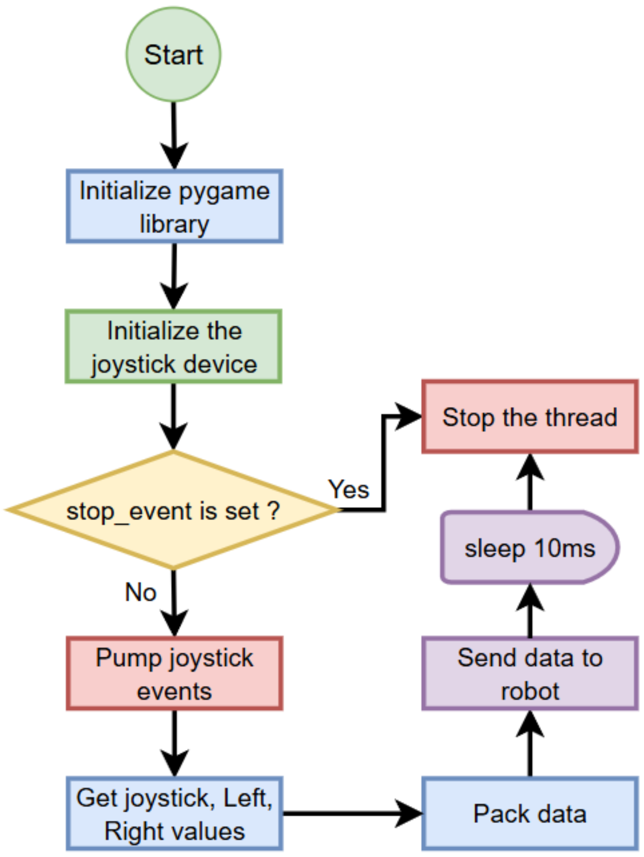


Figure 25: Joystick Data Transmission Thread Flowchart.

As illustrated in Figure 25, the thread begins by initializing the Pygame library and its joystick module, followed by connecting to the first available PS4 controller. It then enters a loop that runs until a stop signal is received. In each iteration, the joystick's vertical axis and two button states are read, packed into a binary format, and sent over the network connection to the robot. A short delay is introduced between iterations to regulate the transmission rate. If an error occurs or the stop signal is triggered, the thread exits the loop and terminates gracefully.

5.4 Testing and Validation

This section presents the results of the testing and validation procedures conducted to ensure that the custom CM5 I/O board, the custom three-port Ethernet switch board, and the overall multi-robot system met the project's requirements for real-time control, efficient communication, and robust operation within a three-host network comprising two robots and a remote control. The tests were divided into hardware tests, which evaluated the functionality and power efficiency of the CM5 I/O board and the Ethernet switch board, and software tests, which assessed the system’s performance in terms of command execution accuracy and response latency to PS4 controller inputs.

5.4.1 Hardware Test

The hardware tests focused on validating the functionality of the custom CM5 I/O board and the three-port Ethernet switch board, as well as ensuring their power consumption remained within the specified budget of the TPS5450 buck converter (25 W total).

CM5 I/O Board Functional Tests

Functional testing of the CM5 I/O board was conducted to verify the operation of its modified I/O ports and the integrated 5V/5A step-down power circuit.

The **USB 3.0 port** was tested by connecting a Bluetooth receiver for a wireless mouse and keyboard to confirm normal operation. Both devices operated normally, confirming the port’s ability to deliver stable power and reliable data transmission for peripheral devices.

The **micro HDMI port** was connected to a monitor via an HDMI cable and successfully displayed video output, validating its functionality for potential use in debugging or external display applications.

The **Ethernet connector**, soldered to a CAT5e cable and connected to a router, negotiated a 1 Gbps link and provided stable internet connectivity, confirming reliable network performance.

The **dual MIPI camera interfaces** were tested by connecting two cameras and streaming video simultaneously to a remote laptop, followed by visual inspection of the results. Both cameras delivered high-quality video streams; however, the camera connected to the CAM0 socket exhibited flickering approximately every three seconds. This issue was consistently reproduced using different cameras and another CM5 I/O board, indicating a hardware-related fault in the custom board design.

The **six-pin GPIO header** was tested for PWM output on GPIO pins 12 and 13, which were used to drive actual motors. The test confirmed proper motor control functionality, though minor signal deviations were recorded: approximately 0.6% error in duty cycle and 0.07% error in frequency. Other GPIO pins, such as DIR and FG, functioned correctly with the motor drivers.

Finally, the **TPS5450 step-down power circuit** was tested by applying a 24V DC input and connecting the output to a 5A load using a load tester. The circuit shut down its output after running for approximately one minute due to the activation of its overheat protection (thermal shutdown) feature. The test was then continued with a 3A load, where the circuit remained stable, and the chip’s temperature was measured at 54ºC. These results indicate that the power circuit did not meet the original 5A (25W) load expectation and is reliably stable only up to 3A (15W) under the current design.

Three-port Ethernet Switch Functional Tests

The three-port Fast Ethernet switch board, based on the Microchip KSZ8794 (Section 4.1), was tested to validate its ability to facilitate reliable communication within the three-host network. A series of functional tests were conducted to assess connectivity, data throughput, and long-term stability.

Connectivity Test: The network was configured with the remote control and two robot units connected through the custom switch board. A continuous ping test was performed between all pairs of devices to confirm mutual connectivity. The test was executed over an extended period (one hour), during which approximately 3600 packets were sent per device pair to monitor packet loss, latency variation, and network stability. The results showed that one device pair exhibited no packet loss, with a minimum round-trip time of 0 ms, a maximum of 5 ms, and an average of 0 ms. The second device pair experienced minimal packet loss, with only two packets lost, a minimum round-trip time of 0 ms, a maximum of 6 ms, and an average of 0 ms. These results indicate that the custom switch board provides stable, low-latency communication with negligible packet loss, meeting the performance expectations for the intended robotic applications.

The **bandwidth and throughput test** was performed using the iperf3 tool. In this setup, the robots acted as clients and the remote control served as the server. Over 10 test iterations, the average unidirectional bandwidth sustained on each port was approximately 95 Mbps on the sender side and 93 Mbps on the receiver side. These values are near the theoretical maximum of 100 Mbps for Fast Ethernet, confirming that the switch is capable of handling high-throughput data transmission effectively.

Power Consumption Test

The power consumption test evaluated the energy efficiency of the CM5 I/O board and the three-port Ethernet switch under normal operating conditions, including motor control, communication tasks, and video streaming from two cameras. A digital multimeter was used to measure the current drawn by the custom CM5 I/O board and Ethernet switch separately. The total current draw was approximately 1.12A (about 5.6W) for the CM5 I/O board and about 0.1A (approximately 0.5W) for the Ethernet switch. These results confirm that the system operated within the power budget (25W), even though the actual capacity of the TPS5450, as tested above, was about 15W.

5.4.2 Software Test

The software tests evaluated the system-level performance of the multi-robot system, with a focus on accurate execution of commands and response latency to PS4 controller inputs.

The test involved observing the robots’ behavior in response to PS4 joystick and button inputs. Specifically, joystick movements were expected to translate into forward or backward motion, while button presses, such as the square or circle buttons, were intended to cause the robots to move left or right. The observed motor behavior aligned well with the expected outputs, confirming that the software interpreted and executed control inputs correctly.

In addition to verifying functional correctness, the test demonstrated the software’s responsiveness. The system responded to inputs with minimal latency, effectively achieving near real-time performance. This responsiveness was attributed to the use of task scheduling and multithreading, which allowed the system to handle concurrent tasks efficiently and maintain smooth, coordinated control across multiple subsystems.

6 Results and Discussion

This section evaluates the testing outcomes of the custom CM5 I/O board, the three-port Ethernet switch board, and the embedded application, as detailed in Section 5.4, against the project's goals of real-time control, efficient communication, and robust operation in a three-host network with two robots and a remote control. Test screenshots are included in the Appendix for additional details.

The hardware tests confirmed the functionality of the CM5 I/O board's modified ports: the USB 3.0 port successfully supported a Bluetooth receiver, the micro HDMI port displayed video output correctly, and the Ethernet connection achieved a stable 1 Gbps link. The six-pin GPIO header produced PWM signals with minor deviations (approximately 0.6% duty cycle error and 0.07% frequency error), which proved sufficient for motor control during testing. However, the dual MIPI camera interfaces revealed a hardware-related fault: the camera connected to the CAM0 socket exhibited flickering approximately every three seconds, even after testing with different cameras and an alternative CM5 I/O board. This suggests a potential signal integrity issue due to the compact PCB layout, affecting the reliability of dual-camera streaming.

The TPS5450 step-down power circuit, originally designed to handle a 5A (25W) load, was unable to sustain such performance. It triggered thermal shutdown after approximately one minute under a 5A load, stabilizing only at a 3A (15W) load with a chip temperature of 54^{o}C. Despite this shortfall, the system's actual power demand (5.6W for the CM5 I/O board and 0.5 W for the Ethernet switch), remained well within the TPS5450's stable output capacity.

The custom three-port Ethernet switch board, based on the Microchip KSZ8794 chip, demonstrated excellent performance. During one hour of continuous ping testing (approximately 3600 packets per pair), one device pair experienced no packet loss, while another had only two lost packets. The average was 0 ms, indicating highly stable and low-latency communication. Bandwidth testing using iperf3 revealed an average unidirectional throughput of 95 Mbps (sender side) and 93 Mbps (receiver side), which approaches the theoretical maximum for Fast Ethernet. These results confirm that the custom switch board meets the communication requirements for real-time robotic control.

Software testing showed that robot movements accurately responded to PS4 controller inputs, with minimal latency observed. The multi-threaded and real-time task scheduling architecture ensured smooth motor control, effective speed feedback, and seamless communication between the robots and the remote control unit.

Overall, the testing results validate the effectiveness of the custom hardware and software solutions. Minor limitations, such as the CAM0 camera flickering and the power circuit’s thermal limit at high load, were identified, suggesting areas for future improvement.

7 Summary and Conclusions

This thesis presented the design, development, and validation of a custom CM5 I/O board, a three-port Ethernet switch board, and a Python-based embedded application for multi-robot control. The project addressed the limitations of standard Raspberry Pi 5 hardware, offering a compact, energy-efficient, and network-capable solution tailored for robotic applications.

The custom CM5 I/O board successfully integrated essential interfaces, a 5V/5A step-down circuit, and repositioned ports for improved compactness and usability in robotic systems. The three-port Ethernet switch board enabled stable, low-latency communication among the robots and the remote control device, crucial for coordinated multi-robot operations.

Testing results confirmed that the system achieved the intended performance targets, including real-time motor control, reliable Ethernet communication, and acceptable power consumption within system constraints. While the dual-camera interface exhibited flickering issues and the TPS5450 buck converter underperformed at maximum load, these limitations did not impact the core functionality of the system.

In conclusion, the project demonstrates the viability of using customized embedded hardware and software solutions to enhance robotic control efficiency. Future work could focus on improving the power management design to support higher loads, enhancing the signal integrity of MIPI camera interfaces, and implementing Gigabit Ethernet support for multi-camera streaming and higher bandwidth requirements.

References

1 Raspberry Pi Foundation. Raspberry Pi 5 [Internet]. Available from: https://www.raspberrypi.com/products/raspberry-pi-5/

2 Raspberry Pi Foundation. Raspberry Pi 5 Product Brief [Internet]. Available from: https://datasheets.raspberrypi.com/rpi5/raspberry-pi-5-product-brief.pdf

3 Texas Instruments. TPS5450 5A Step-Down Converter Datasheet [Internet]. Available from: https://www.ti.com/document-viewer/tps5450/datasheet

4 ROHM Semiconductor. Buck Converter Efficiency Factors [Internet]. Available from: https://fscdn.rohm.com/en/products/databook/applinote/ic/power/switching_regulator/buck_converter_efficiency_app-e.pdf

5 Texas Instruments. SLVA630A: Basic Calculation of a Buck Converter's Power Stage [Internet]. Available from: https://www.ti.com/lit/an/slva630a/slva630a.pdf

6 Monolithic Power Systems. Buck Converter Topology Design Guide [Internet]. Available from: https://www.monolithicpower.com/en/buck-converter-topology-design-guide

7 He X, Zhang L, Liu W, Fan Y, Zhao S. Enhancing Industrial Communication with Ethernet/Internet Protocol: A Study and Analysis of Real-Time Cooperative Robot Communication and Automation via Transmission Control Protocol/Internet Protocol [Internet]. Sensors (Basel). 2023;23(22):9301. Available from: https://pmc.ncbi.nlm.nih.gov/articles/PMC10611002/

8 Bakhshi MR, Zamani MS, Talebi H. Real-time Communication between Robot PLC and PC over Ethernet-based Protocols [Internet]. arXiv preprint arXiv:1902.01924. 2019. Available from: https://arxiv.org/pdf/1902.01924

9 IETF. RFC 6455: The WebSocket Protocol [Internet]. 2011. Available from: https://www.rfc-editor.org/rfc/rfc6455.html

10 The Curve IoT. What Are WebSockets and Why They Are Essential for IoT [Internet]. Available from: https://thecurve.io/what-are-websockets-and-why-they-are-essential-for-iot/

11 Khan Z, Jain V. Analysis of WebSockets as the New Age Protocol for Remote Robot Tele-operation [Internet]. Procedia Comput Sci. 2015;76:19-25. Available from: https://www.sciencedirect.com/science/article/pii/S1474667015343688

12 Renesas Electronics. Brushless DC Motor Basics (Part 1): Overview [Internet]. Available from: https://www.renesas.com/en/support/engineer-school/brushless-dc-motor-01-overview

13 Microchip Technology. Understanding Brushless DC Motor Fundamentals [Internet]. Available from: https://ww1.microchip.com/downloads/aemDocuments/documents/OTH/ApplicationNotes/ApplicationNotes/00885a.pdf

14 Texas Instruments. Understanding Sensorless BLDC Motor Control and Its Application [Internet]. Available from: https://www.ti.com/lit/an/sprabz4/sprabz4.pdf

15 Kwan JM, Choi JW. A Review of Sensorless Control Techniques for BLDC Motors [Internet]. Sensors (Basel). 2011;11(6):5758-5777. Available from: https://pmc.ncbi.nlm.nih.gov/articles/PMC3231115/

16 Seeed Studio. Basic Electronics: Pulse Width Modulation (PWM) and Arduino Applications [Internet]. 2020 Jun 16. Available from: https://www.seeedstudio.com/blog/2020/06/16/basic-electronics-pulse-width-modulationpwm-and-arduino-applications/

17 NXP Semiconductors. AN1916: BLDC Motor Control with Hall Sensors [Internet]. Available from: https://www.nxp.com/docs/en/application-note/AN1916.pdf

18 Raspberry Pi Foundation. Compute Module 5 (CM5) Datasheet [Internet]. Available from: https://datasheets.raspberrypi.com/cm5/cm5-datasheet.pdf

19 Raspberry Pi Foundation. Compute Module 5 I/O Board Datasheet [Internet]. Available from: https://datasheets.raspberrypi.com/cm5/cm5io-datasheet.pdf

20 Microchip Technology. KSZ8794CNX Ethernet Switch Datasheet [Internet]. Available from: https://ww1.microchip.com/downloads/aemDocuments/documents/UNG/ProductDocuments/DataSheets/KSZ8794CNX-Data-Sheet-DS00002134.pdf

21 HE9703 Series DC/DC Converter Datasheet [Internet]. Available from: https://atta.szlcsc.com/upload/public/pdf/source/20200818/C723783_47C0FB0690B3745218579CD1DF0C296C.pdf

Test Results

1. PWM Signal Measurement

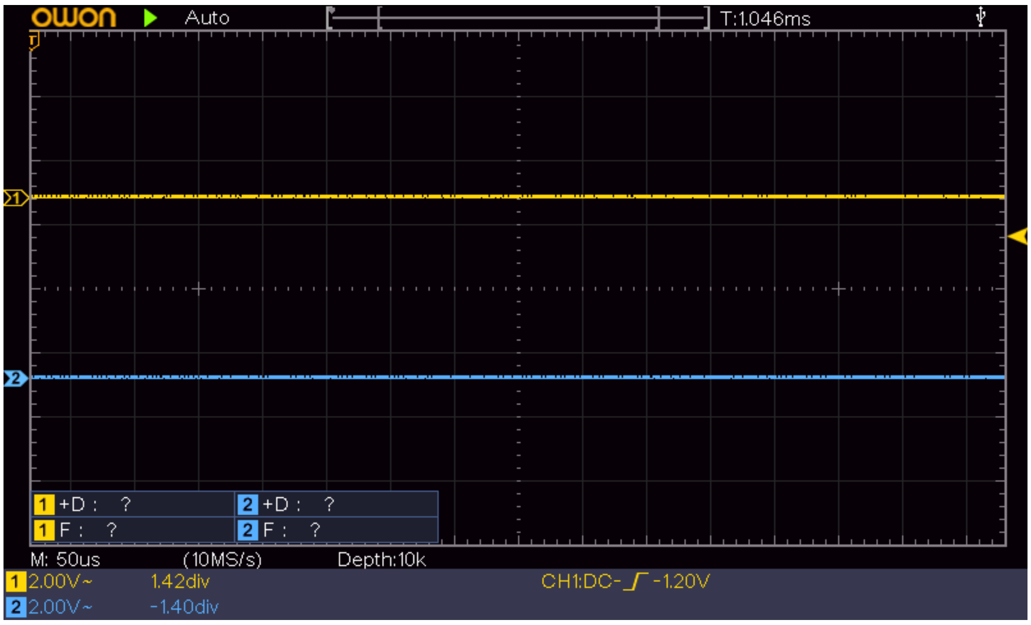


Figure 1: PWM Signal at 0% duty cycle

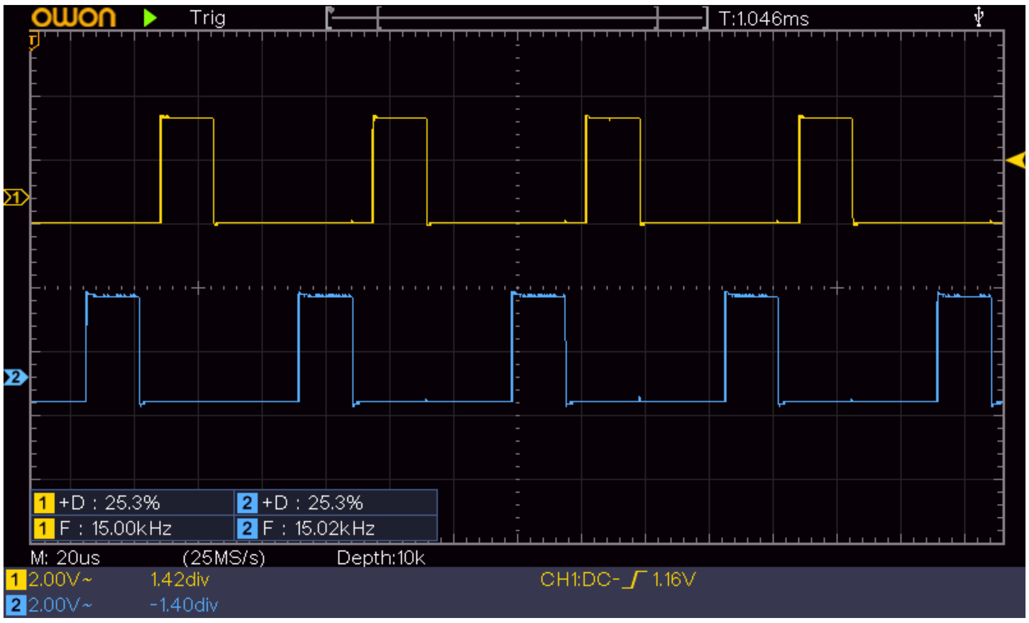


Figure 2: PWM Signal at 25% duty cycle

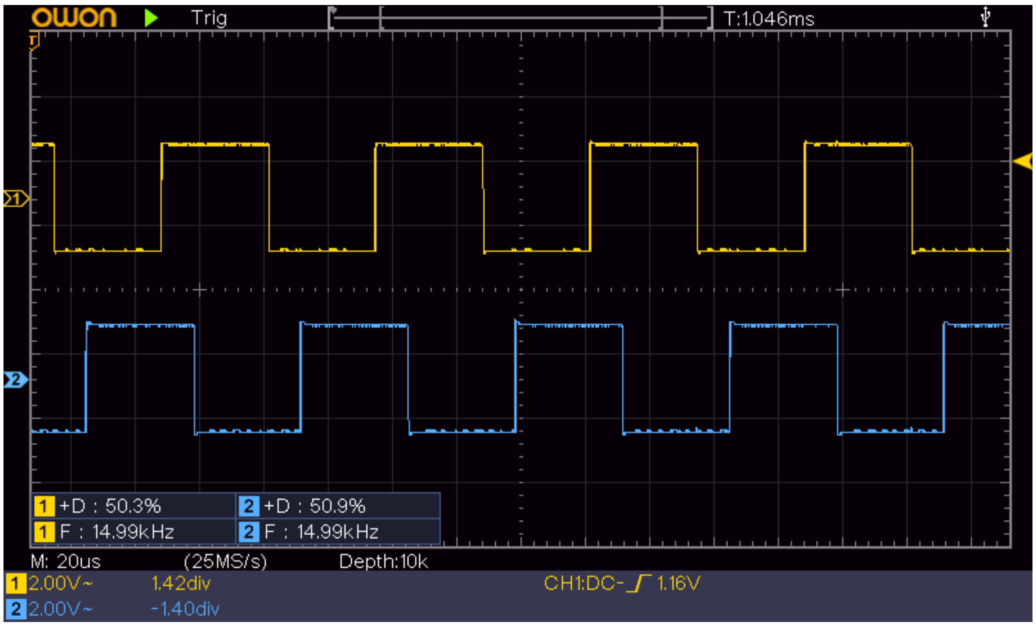


Figure 3: PWM Signal at 50% duty cycle

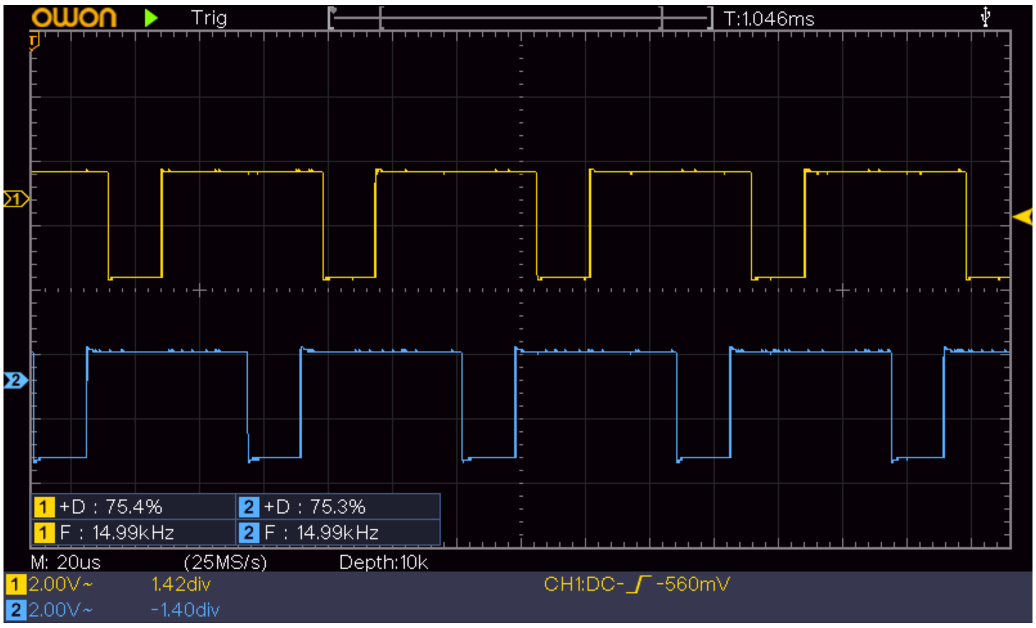


Figure 4: PWM Signal at 75% duty cycle

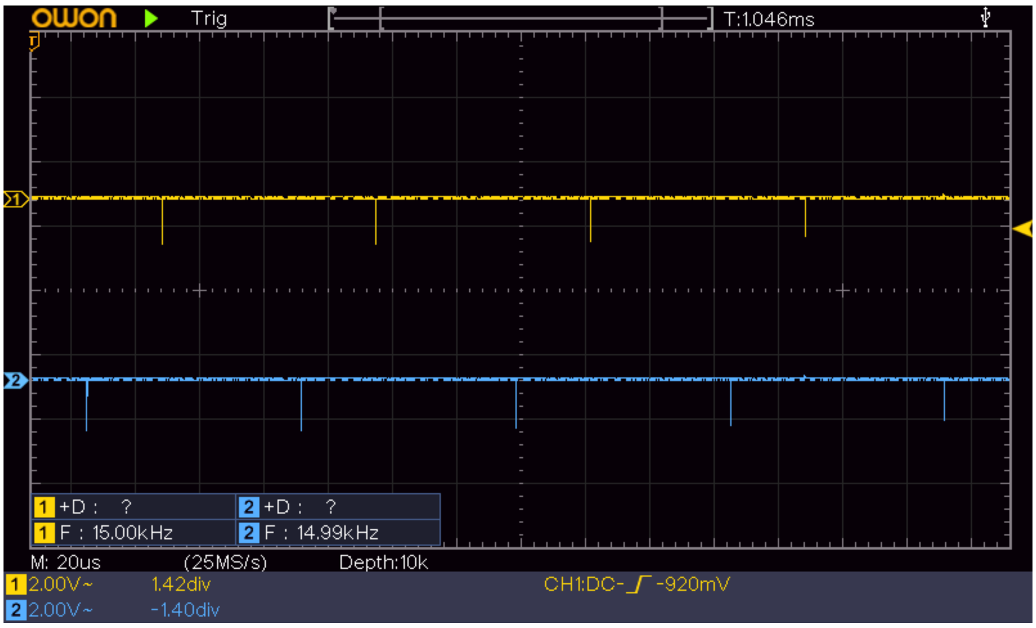


Figure 4: PWM Signal at 100% duty cycle

2. TPS5450 Step-Down Capacity Test

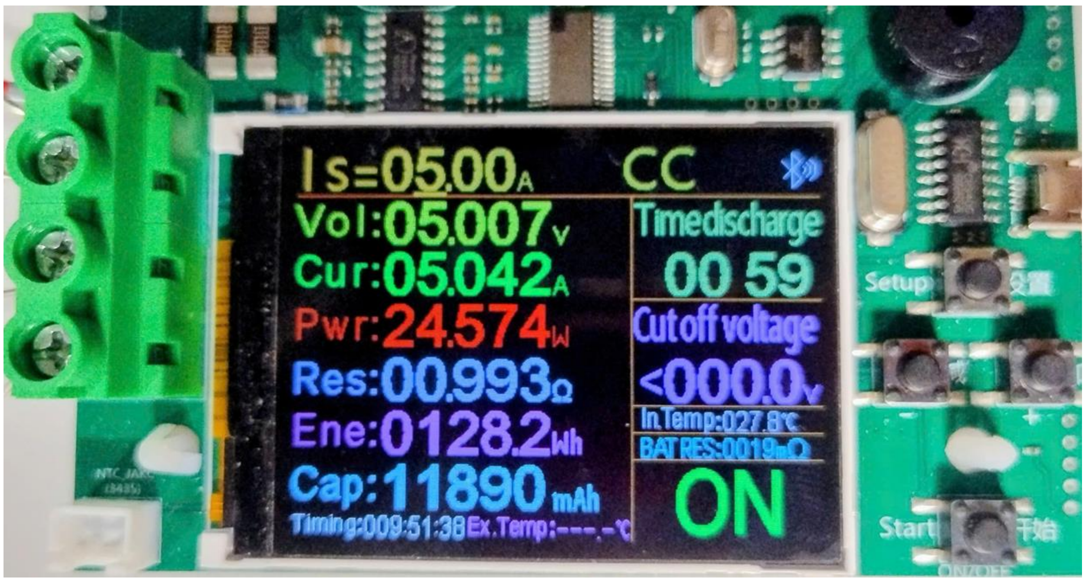
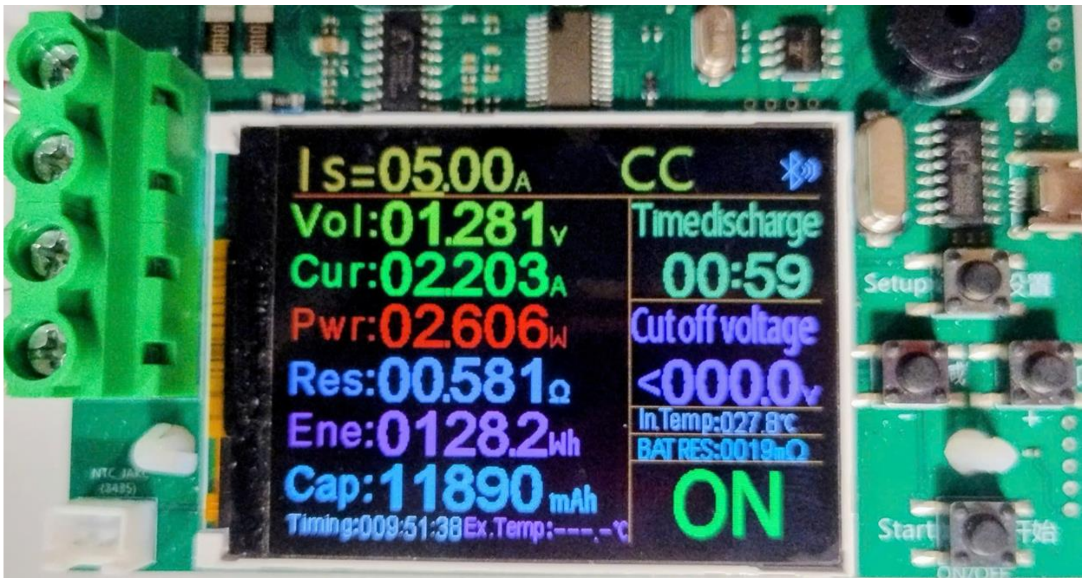
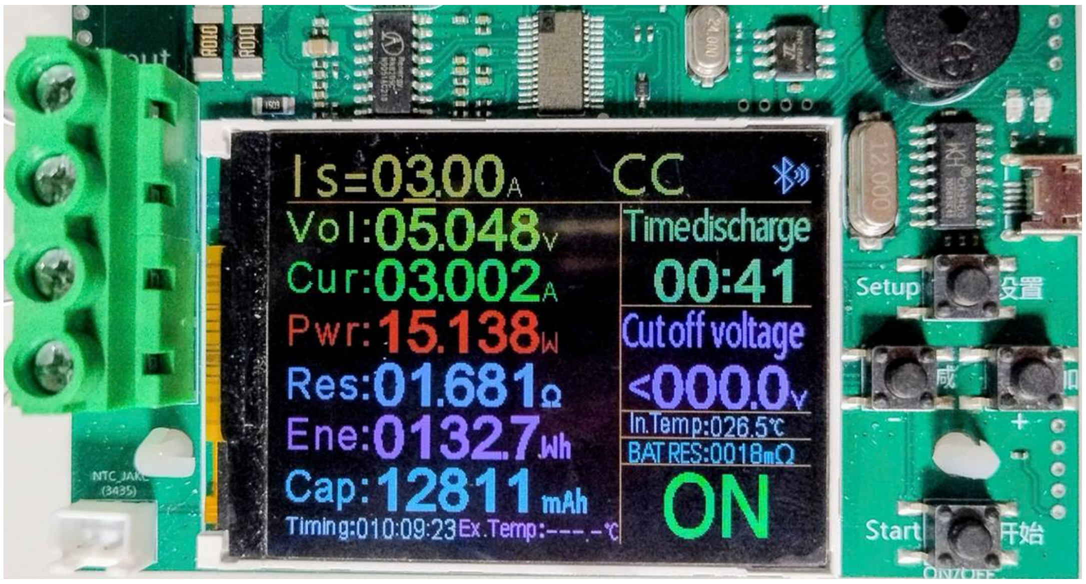


Figure 5: 5A load test at start



```
Is=05.00A        CC
Vol:01.281V      Timedischarge
Cur:02.203A      00:59
Pwr:02.606W      Cutoff voltage
Res:00.581Ω      <000.0V
Ene:0128.2Wh     In.Temp:027.8℃
                 BAT RES:0019mΩ
Cap:11890mAh     ON
Timing:009:51:38 Ex.Temp:---.-℃
```

Figure 6: 5A load test after one minute



```
Is=03.00A        CC
Vol:05.048V      Timedischarge
Cur:03.002A      00:41
Pwr:15.138W      Cutoff voltage
Res:01.681Ω      <000.0V
Ene:0132.7Wh     In.Temp:026.5℃
                 BAT RES:0018mΩ
Cap:12811mAh     ON
Timing:010:09:23 Ex.Temp:---.-℃
```

Figure 7: 3A load test

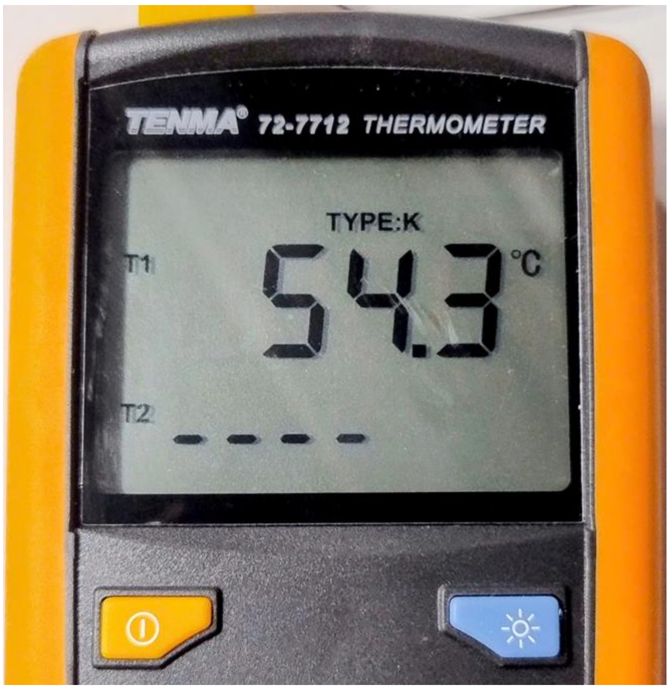


Figure 8: Temperature of the TPS5450 chip at 3A load test

3. CM5 I/O Board Current Draw Test

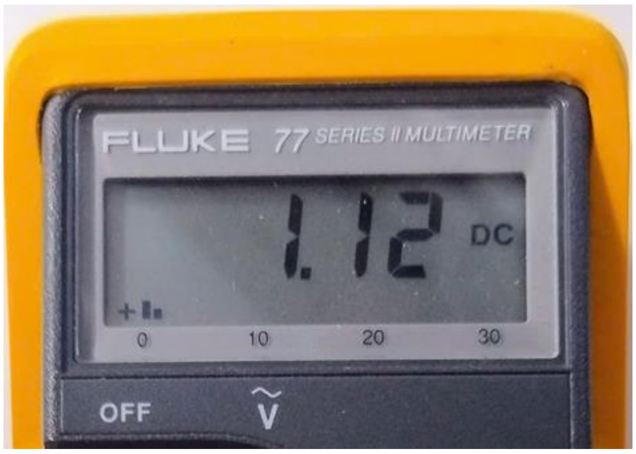


Figure 9: Current draw of the custom CM5 I/O board

4. Ethernet Switch Board Current Draw Test

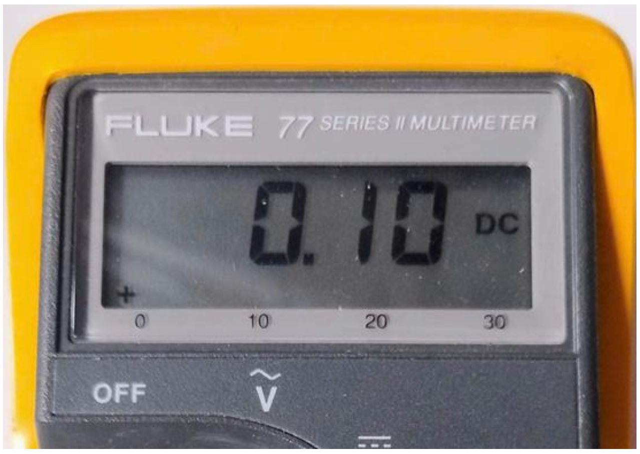
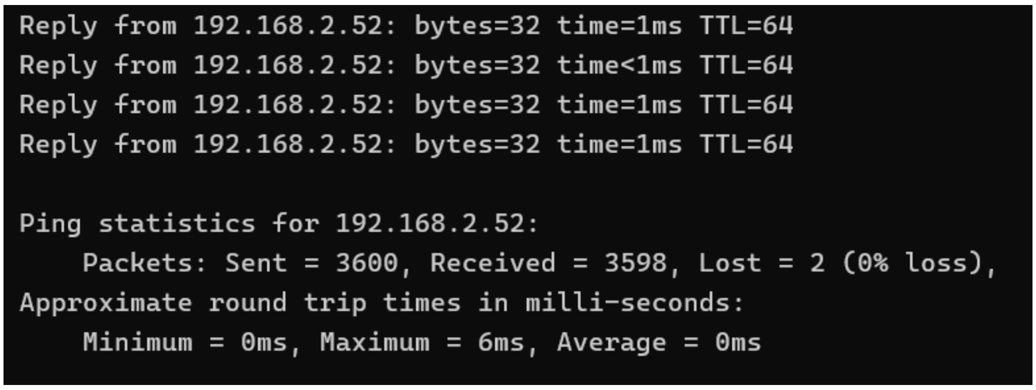


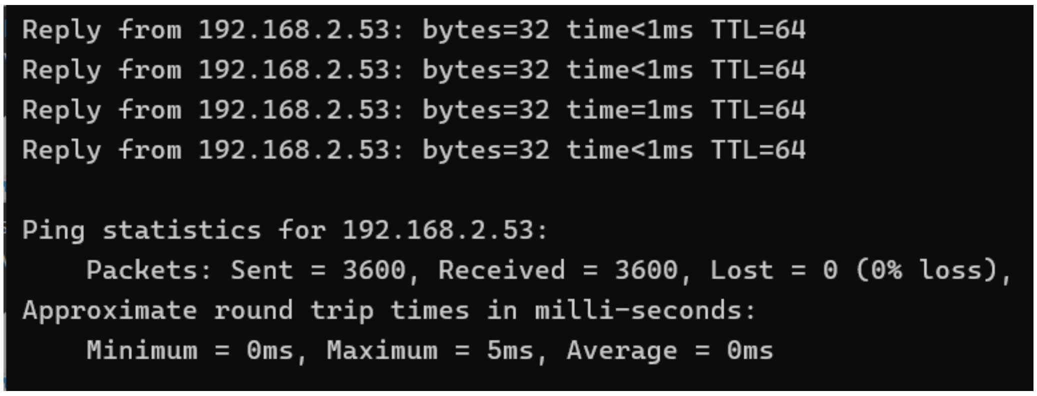
Figure 10: Current draw of the custom Ethernet Switch board

5. Ping Test Results



```
Reply from 192.168.2.52: bytes=32 time=1ms TTL=64
Reply from 192.168.2.52: bytes=32 time<1ms TTL=64
Reply from 192.168.2.52: bytes=32 time=1ms TTL=64
Reply from 192.168.2.52: bytes=32 time=1ms TTL=64

Ping statistics for 192.168.2.52:
    Packets: Sent = 3600, Received = 3598, Lost = 2 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 6ms, Average = 0ms
```

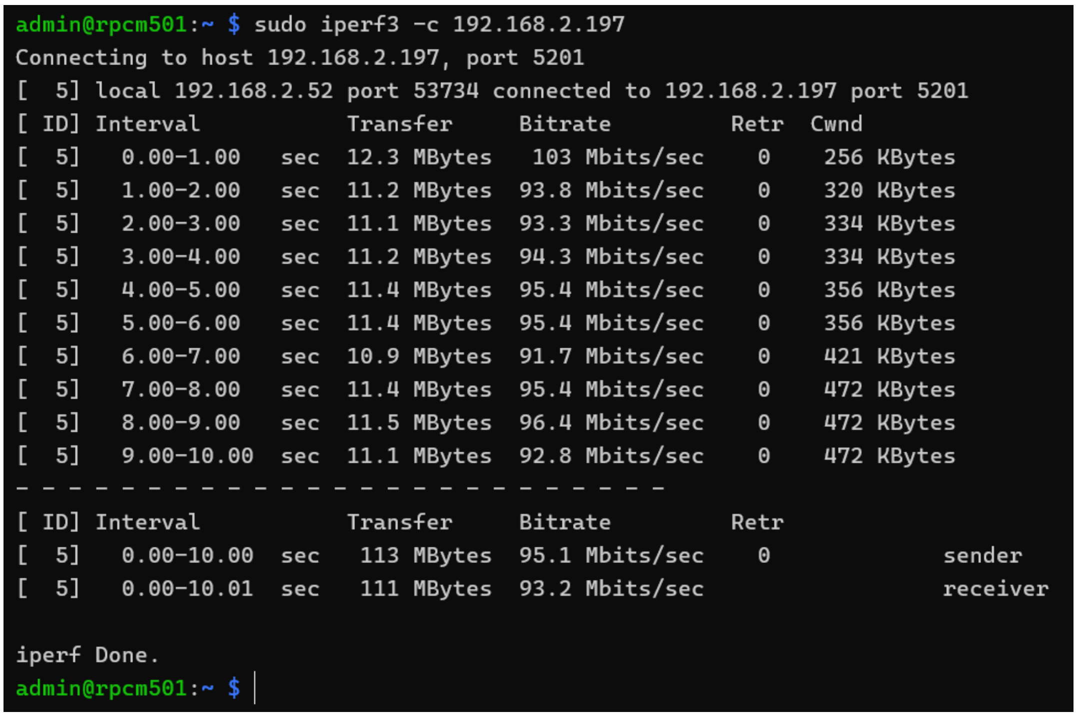
Figure 11: Ping test result from laptop to robot #1



```
Reply from 192.168.2.53: bytes=32 time<1ms TTL=64
Reply from 192.168.2.53: bytes=32 time<1ms TTL=64
Reply from 192.168.2.53: bytes=32 time=1ms TTL=64
Reply from 192.168.2.53: bytes=32 time<1ms TTL=64

Ping statistics for 192.168.2.53:
    Packets: Sent = 3600, Received = 3600, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 5ms, Average = 0ms
```

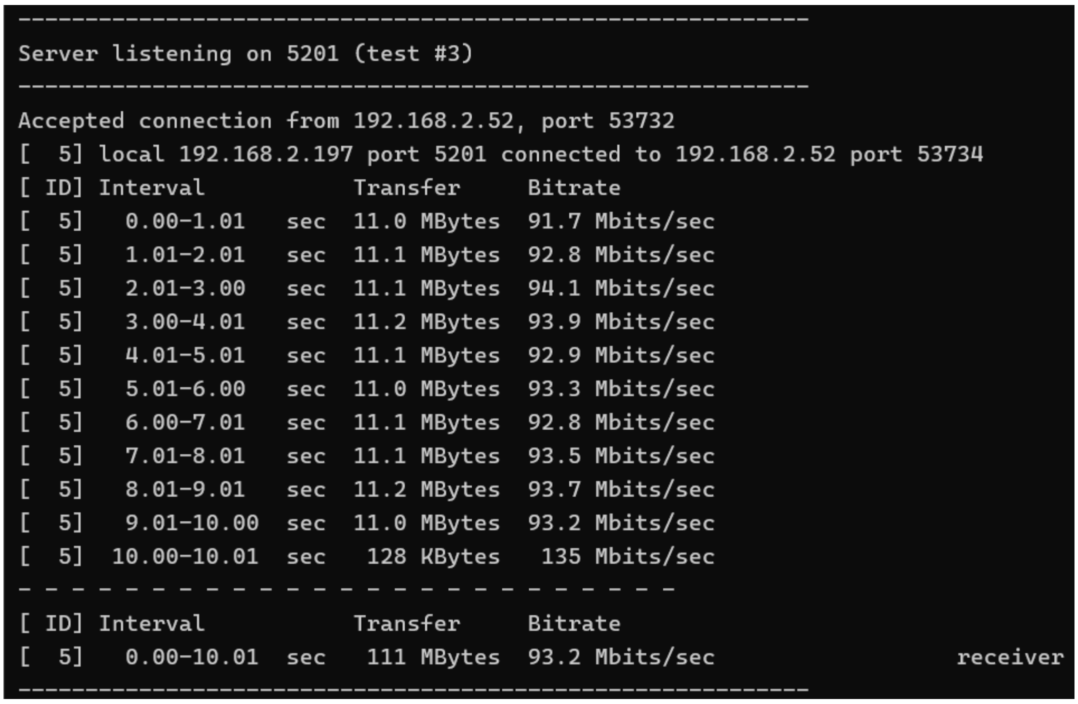
Figure 12: Ping test result from laptop to robot #2

6. iPerf3 Bandwidth Test Results



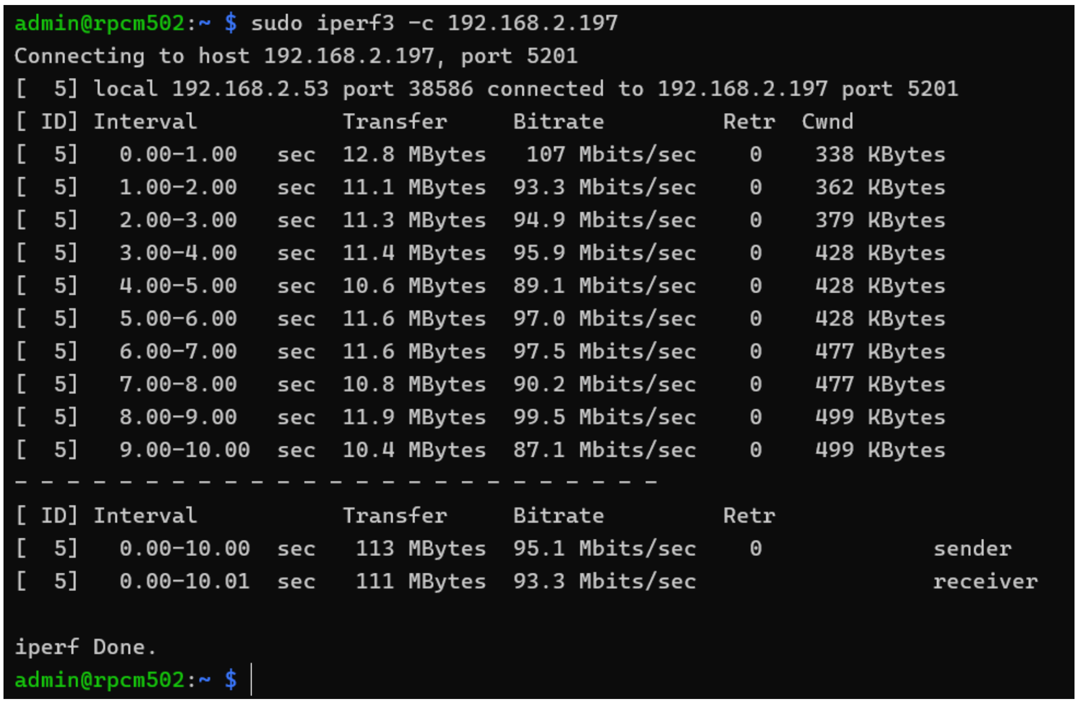
```
admin@rpcm501:~ $ sudo iperf3 -c 192.168.2.197
Connecting to host 192.168.2.197, port 5201
[  5] local 192.168.2.52 port 53734 connected to 192.168.2.197 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[  5]   0.00-1.00   sec  12.3 MBytes   103 Mbits/sec    0    256 KBytes
[  5]   1.00-2.00   sec  11.2 MBytes  93.8 Mbits/sec    0    320 KBytes
[  5]   2.00-3.00   sec  11.1 MBytes  93.3 Mbits/sec    0    334 KBytes
[  5]   3.00-4.00   sec  11.2 MBytes  94.3 Mbits/sec    0    334 KBytes
[  5]   4.00-5.00   sec  11.4 MBytes  95.4 Mbits/sec    0    356 KBytes
[  5]   5.00-6.00   sec  11.4 MBytes  95.4 Mbits/sec    0    356 KBytes
[  5]   6.00-7.00   sec  10.9 MBytes  91.7 Mbits/sec    0    421 KBytes
[  5]   7.00-8.00   sec  11.4 MBytes  95.4 Mbits/sec    0    472 KBytes
[  5]   8.00-9.00   sec  11.5 MBytes  96.4 Mbits/sec    0    472 KBytes
[  5]   9.00-10.00  sec  11.1 MBytes  92.8 Mbits/sec    0    472 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[  5]   0.00-10.00  sec   113 MBytes  95.1 Mbits/sec    0             sender
[  5]   0.00-10.01  sec   111 MBytes  93.2 Mbits/sec                  receiver

iperf Done.
admin@rpcm501:~ $
```

Figure 13: iPerf3 test result on robot #1 side to server



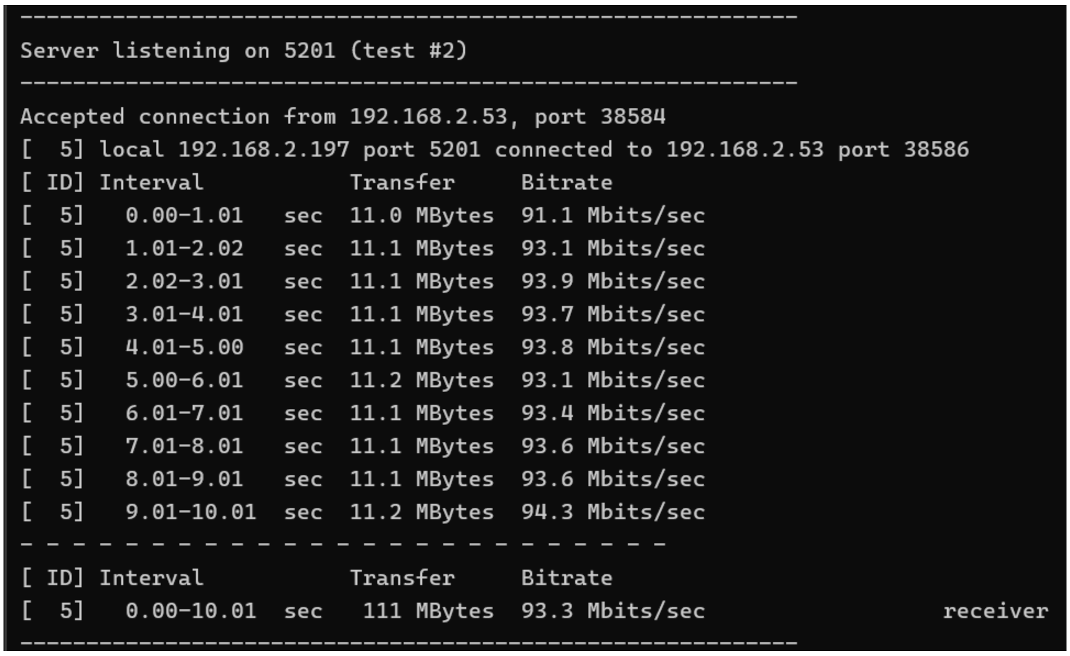
```
-----------------------------------------------------------
Server listening on 5201 (test #3)
-----------------------------------------------------------
Accepted connection from 192.168.2.52, port 53732
[  5] local 192.168.2.197 port 5201 connected to 192.168.2.52 port 53734
[ ID] Interval           Transfer     Bitrate
[  5]   0.00-1.01   sec  11.0 MBytes  91.7 Mbits/sec
[  5]   1.01-2.01   sec  11.1 MBytes  92.8 Mbits/sec
[  5]   2.01-3.00   sec  11.1 MBytes  94.1 Mbits/sec
[  5]   3.00-4.01   sec  11.2 MBytes  93.9 Mbits/sec
[  5]   4.01-5.01   sec  11.1 MBytes  92.9 Mbits/sec
[  5]   5.01-6.00   sec  11.0 MBytes  93.3 Mbits/sec
[  5]   6.00-7.01   sec  11.1 MBytes  92.8 Mbits/sec
[  5]   7.01-8.01   sec  11.1 MBytes  93.5 Mbits/sec
[  5]   8.01-9.01   sec  11.2 MBytes  93.7 Mbits/sec
[  5]   9.01-10.00  sec  11.0 MBytes  93.2 Mbits/sec
[  5]  10.00-10.01  sec   128 KBytes   135 Mbits/sec
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate
[  5]   0.00-10.01  sec   111 MBytes  93.2 Mbits/sec                  receiver
-----------------------------------------------------------
```

Figure 14: iPerf3 test result received on server side from robot #1



```
admin@rpcm502:~ $ sudo iperf3 -c 192.168.2.197
Connecting to host 192.168.2.197, port 5201
[  5] local 192.168.2.53 port 38586 connected to 192.168.2.197 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[  5]   0.00-1.00   sec  12.8 MBytes   107 Mbits/sec    0    338 KBytes
[  5]   1.00-2.00   sec  11.1 MBytes  93.3 Mbits/sec    0    362 KBytes
[  5]   2.00-3.00   sec  11.3 MBytes  94.9 Mbits/sec    0    379 KBytes
[  5]   3.00-4.00   sec  11.4 MBytes  95.9 Mbits/sec    0    428 KBytes
[  5]   4.00-5.00   sec  10.6 MBytes  89.1 Mbits/sec    0    428 KBytes
[  5]   5.00-6.00   sec  11.6 MBytes  97.0 Mbits/sec    0    428 KBytes
[  5]   6.00-7.00   sec  11.6 MBytes  97.5 Mbits/sec    0    477 KBytes
[  5]   7.00-8.00   sec  10.8 MBytes  90.2 Mbits/sec    0    477 KBytes
[  5]   8.00-9.00   sec  11.9 MBytes  99.5 Mbits/sec    0    499 KBytes
[  5]   9.00-10.00  sec  10.4 MBytes  87.1 Mbits/sec    0    499 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[  5]   0.00-10.00  sec   113 MBytes  95.1 Mbits/sec    0             sender
[  5]   0.00-10.01  sec   111 MBytes  93.3 Mbits/sec                  receiver

iperf Done.
admin@rpcm502:~ $
```

Figure 15: iPerf3 test result on robot #2 side to server



```
-----------------------------------------------------------
Server listening on 5201 (test #2)
-----------------------------------------------------------
Accepted connection from 192.168.2.53, port 38584
[  5] local 192.168.2.197 port 5201 connected to 192.168.2.53 port 38586
[ ID] Interval           Transfer     Bitrate
[  5]   0.00-1.01   sec  11.0 MBytes  91.1 Mbits/sec
[  5]   1.01-2.02   sec  11.1 MBytes  93.1 Mbits/sec
[  5]   2.02-3.01   sec  11.1 MBytes  93.9 Mbits/sec
[  5]   3.01-4.01   sec  11.1 MBytes  93.7 Mbits/sec
[  5]   4.01-5.00   sec  11.1 MBytes  93.8 Mbits/sec
[  5]   5.00-6.01   sec  11.2 MBytes  93.1 Mbits/sec
[  5]   6.01-7.01   sec  11.1 MBytes  93.4 Mbits/sec
[  5]   7.01-8.01   sec  11.1 MBytes  93.6 Mbits/sec
[  5]   8.01-9.01   sec  11.1 MBytes  93.6 Mbits/sec
[  5]   9.01-10.01  sec  11.2 MBytes  94.3 Mbits/sec
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate
[  5]   0.00-10.01  sec   111 MBytes  93.3 Mbits/sec                  receiver
-----------------------------------------------------------
```

Figure 16: iPerf3 test result received on server side from robot #2