Faculty of Mechanical Engineering and Mechatronics
at Karlsruhe University of Applied Sciences

DESIGN OF A MODULAR MECHATRONIC GADGET SYSTEM FOR MANUAL ASSISTANCE (ESUPERHEROES)

Master-Thesis

submitted in partial fulfillment of the requirements for the degree of
Master of Engineering (M.Eng.)

presented by

Meron Woldeselassie

born at 12.06.1997 in Germany, Offenburg
Program of Study: Mechatronics (EU4M)
Matr.-Nr.: UO317723

Tutor: Prof. Dr.-Ing. IGNACIO ALVAREZ GARCIA
Korreferent: Prof. Dr.-Ing. RAMON RUBIO GARCIA

Gijón, the July 24, 2026

Project Description

This Master's thesis focuses on the design and development of a modular system of interchangeable mechatronic gadgets that can be attached to a wearable glove. The proposed system is inspired by the *Superhéroes* project developed at the MediaLab of the University of Oviedo.

The system consists of two main components:

1. **Glove (Master Unit):** The glove integrates the reusable common electronics, including:
    - Power supply (battery and power management),
    - Master microcontroller,
    - Internal communication via a CAN bus for command exchange and telemetry,
    - User interface and intention detection based on sensors located on the glove or forearm (e.g., push buttons, IMU-based gesture recognition, and optionally EMG).
2. **Gadgets (Interchangeable Functional Modules):** Each gadget is an autonomous mechatronic module comprising:
    - Actuators (servo motors or geared DC/BLDC motors depending on the application),
    - Local electronics (microcontroller with CAN transceiver or integrated CAN controller) for low-level control,
    - Application-specific sensors (e.g., force or pressure sensors in a gripper to limit the gripping force),
    - A housing and mechanical structure designed for additive manufacturing.

The glove and the gadgets are connected through a quick-swap mechanical coupling mechanism combined with a pogo-pin electrical connector, providing both power and CAN communication (CANH/CANL). This interface enables rapid module replacement without requiring any modification of the main electronics.

The proposed concept investigates a modular, low-cost, and customizable prosthetic platform in which a single glove can perform different functions by simply exchanging the attached gadget. As part of this work, at least one gripper-type gadget will be designed and implemented. The project covers mechanical design, electronic hardware development, embedded control, and experimental validation. The final outcome will be a fully documented functional prototype together with an analysis of its technical feasibility, manufacturing cost, usability, basic safety aspects, and possible future developments within the *Superhéroes* project.

General Objective

The main objective of this work is to develop a modular system of mechatronic gadgets attachable to a glove for functional hand assistance. The system is based on the principles of low-cost design, additive manufacturing, and the reuse of common electronic hardware. Communication between the glove and the interchangeable modules is implemented using a CAN bus.

Specific Objectives

1. Review the current state of the art in hand assistance devices, modular prostheses, exo-gloves, and interchangeable mechatronic systems. Analyze the *Superhéroes* project as the conceptual and technical foundation, identifying its requirements, limitations, and opportunities for improvement.

2. Design the master glove integrating the main electronics, including the microcontroller, power supply, CAN communication, and user interface.

3. Develop a mechanical and electrical quick-swap interface that enables the safe and reliable connection of interchangeable gadgets using pogo-pin contacts.

4. Design, manufacture using additive manufacturing, and implement the control system for at least two functional gadgets:

    - **Gadget 1 – Pressure-Controlled Gripper:** A motorized gripper integrating an electric actuator and a pressure sensor to limit the applied gripping force. The module includes local control and communicates with the master glove through the CAN bus. The gripping motion is automatically stopped when a predefined pressure threshold is reached.

    - **Gadget 2 – Rotary Assistance Module:** An interchangeable module providing electrically driven rotational motion by means of a geared motor and local electronics. The

module receives position, velocity, or torque commands from the master glove via the CAN bus, demonstrating the versatility of the proposed modular architecture.

5. Perform functional experiments to evaluate the performance of the developed gadgets, the reliability of the coupling mechanism, power consumption, and basic ergonomics.

6. Analyze the manufacturing cost, customization possibilities, and scalability of the proposed platform.

7. Prepare the final Master's thesis together with demonstration material (functional prototype, video, or test bench) for the thesis defense.

Project Schedule

February 2026: Literature review on hand assistance devices, modular prosthetic systems, and analysis of the *Superhéroes* project.

March 2026: Conceptual design of the modular platform, development of the master glove, definition of the quick-swap interface, and selection of electronic components and actuators.

April 2026: Mechanical design and additive manufacturing of the gadgets. Development of the embedded control system and preliminary testing of each module.

May 2026: Complete integration of the glove and interchangeable gadgets. Functional testing together with mechanical and electronic optimization.

June 2026: Final system validation, analysis of the experimental results, preparation of the thesis document, and defense preparation.

Declaration

I hereby declare that I have completed this Master's thesis independently and have used no sources or aids other than those explicitly indicated.

All passages taken either literally or in substance from other sources have been clearly identified by appropriate references. This also applies to drawings, sketches, illustrations, and all information obtained from Internet sources.

.....	.....
Location, Date	Signature

Acknowledgements

First and foremost, I would like to express my sincere gratitude to Prof. Dr. Ramón Rubio García for giving me the opportunity to carry out this master's thesis at the MediaLab of the University of Oviedo. I am deeply grateful for his continuous support, guidance, and trust throughout this project. His enthusiasm for research, innovative thinking, and willingness to share his knowledge have been a constant source of motivation and have significantly contributed to the successful completion of this work.

I would also like to thank the entire MediaLab team for their warm welcome and for creating such an inspiring and collaborative working environment. Working alongside researchers from different backgrounds and being part of an international research group has been an invaluable experience, both professionally and personally. The many discussions, shared ideas, and daily collaboration made my time in Gijón truly unforgettable.

A special acknowledgment goes to the eSuperheroes project, whose vision of providing affordable and task-specific assistive devices inspired this thesis from the very beginning. I am sincerely grateful for the opportunity to contribute to this project and to further develop its concept towards a modular mechatronic platform. It is a privilege to be able to build upon the work of the project and contribute to what I hope will become the next step in its evolution. I truly hope that the results of this work will support future developments and, ultimately, help improve the quality of life of many people who rely on assistive technologies.

Furthermore, I would like to express my sincere appreciation to my academic supervisor Prof. Dr. Ignacio Alvarez Garcia for the valuable feedback, encouragement, and support throughout this thesis.

Finally, I would like to thank my family and friends for their unconditional support, patience, and encouragement throughout my studies. Their confidence in me has always motivated me to pursue my goals, and this achievement would not have been possible without them.

Abstract

Upper-limb prostheses have advanced considerably in recent years, providing improved dexterity, control strategies, and user functionality. However, current research primarily focuses on multi-functional prosthetic hands, while comparatively little attention has been given to task-specific modular prosthetic systems. Existing modular solutions are often limited to mechanically interchangeable devices and generally lack standardized electrical interfaces for integrating powered mechatronic modules.

This thesis presents the design and development of a low-cost modular mechatronic platform for interchangeable prosthetic gadgets, inspired by the eSuperheroes project developed at the MediaLab of the University of Oviedo. The proposed system extends the original concept by introducing a standardized platform that combines a quick-change mechanical coupling with an integrated electrical interface for power supply and CAN-based communication. A master–slave architecture is implemented in which a central control unit coordinates interchangeable gadget modules, each equipped with dedicated electronics for local sensing, actuation, and embedded control.

The developed platform includes the design of the electronic hardware, the mechanical coupling system, the embedded software architecture, and two interchangeable proof-of-concept gadgets demonstrating different functional capabilities. The system was designed following a low-cost philosophy based on additive manufacturing and commercially available electronic components while maintaining scalability for future extensions.

The proposed architecture establishes a common mechanical, electrical, and software interface that enables future mechatronic gadgets to be integrated without redesigning the underlying platform. By extending the vision of the eSuperheroes project beyond mechanically interchangeable devices, this work provides a foundation for an open, scalable, and affordable ecosystem of task-specific prosthetic assistance devices.

Resumen

Los avances en las prótesis de miembro superior han mejorado significativamente la destreza, las estrategias de control y la funcionalidad para el usuario. Sin embargo, la investigación actual se centra principalmente en manos protésicas multifuncionales, mientras que las soluciones modulares específicas para tareas concretas han recibido mucha menos atención. Además, los sistemas modulares existentes suelen limitarse a dispositivos intercambiables únicamente desde el punto de vista mecánico y, por lo general, carecen de interfaces eléctricas estandarizadas para integrar módulos mecatrónicos accionados.

Este trabajo presenta el diseño y desarrollo de una plataforma mecatrónica modular y de bajo coste para gadgets protésicos intercambiables, inspirada en el proyecto eSuperheroes desarrollado en el MediaLab de la Universidad de Oviedo. La plataforma propuesta amplía el concepto original mediante la incorporación de una interfaz estandarizada que combina un mecanismo mecánico de cambio rápido con una interfaz eléctrica integrada para la alimentación y la comunicación mediante CAN. Se implementa una arquitectura maestro-esclavo en la que una unidad central coordina módulos intercambiables, cada uno equipado con su propia electrónica para el control local, la adquisición de sensores y el accionamiento de actuadores.

La plataforma desarrollada incluye el diseño del hardware electrónico, el sistema de acoplamiento mecánico, la arquitectura del software embebido y dos gadgets intercambiables que demuestran diferentes funcionalidades. El sistema se ha diseñado siguiendo una filosofía de bajo coste basada en fabricación aditiva y componentes electrónicos comerciales, manteniendo al mismo tiempo la escalabilidad necesaria para futuras ampliaciones.

La arquitectura propuesta establece una interfaz mecánica, eléctrica y de software común que permite integrar futuros gadgets mecatrónicos sin necesidad de rediseñar la plataforma base. Al ampliar la visión del proyecto eSuperheroes más allá de los dispositivos intercambiables exclusivamente mecánicos, este trabajo sienta las bases para un ecosistema abierto, escalable y asequible de dispositivos protésicos modulares específicos para distintas tareas.

Contents

List of Figures

List of Tables

Nomenclature

Symbols

F_G	Gripping force
g	Gravitational acceleration
I	Electric current
m	Object mass
M	Torque
P	Electrical power
t	Operating time
U	Voltage
C_{req}	Required battery capacity
DC	CAN bus duty cycle
D_{active}	Active duty cycle
D_{standby}	Standby duty cycle
E_{bat}	Required battery energy
I_{dom}	Dominant-state current
I_{rec}	Recessive-state current
I_{TJA1050}	Average current consumption of the TJA1050 CAN transceiver
L_{emg}	Length of the EMG CAN frame
L_{poti}	Length of the potentiometer CAN frame
P_{active}	Active-state power consumption
P_{avg}	Average power consumption
P_{bat}	Battery-side power consumption
P_{load}	Load power consumption
P_{standby}	Standby-state power consumption
t_{active}	Total CAN transmission time
t_{emg}	EMG frame transmission time
t_{poti}	Potentiometer frame transmission time
T	Transmission period
U_{nom}	Nominal battery voltage
η	DC-DC converter efficiency

Abbreviations

ADC	Analog-to-Digital Converter
CAN	Controller Area Network
CRC	Cyclic Redundancy Check
DLC	Data Length Code

EMG	Electromyography
EOF	End of Frame
FDM	Fused Deposition Modeling
GPIO	General Purpose Input/Output
IDE	Identifier Extension
IFS	Interframe Space
PCB	Printed Circuit Board
PLA	Polylactic Acid
PWM	Pulse Width Modulation
RAM	Random Access Memory
RPM	Revolutions per Minute
sEMG	Surface Electromyography
TPU	Thermoplastic Polyurethane
TWAI	Two-Wire Automotive Interface
USB	Universal Serial Bus
Wi-Fi	Wireless Fidelity
XML	Extensible Markup Language

1 Introduction

Upper-limb impairments can considerably reduce a person's ability to perform activities of daily living and therefore have a significant impact on independence and quality of life. Prosthetic and assistive technologies have continuously evolved over the past decades with the aim of restoring lost functionality through advances in mechanics, electronics, sensing, and embedded control. At the same time, the increasing availability of additive manufacturing and low-cost embedded hardware has enabled the development of more affordable and customizable assistive devices.

Modern prosthetic systems are expected not only to restore basic functionality but also to support users in a wide variety of everyday activities. Since different tasks often impose different mechanical and functional requirements, the development of adaptable assistive systems has become an increasingly important research topic. This has led to growing interest in modular design approaches, which separate common platform functionality from application-specific components and thereby facilitate customization, maintenance, and future system extensions.

The objective of this thesis is to develop a modular mechatronic platform for interchangeable assistive modules. The proposed platform combines standardized mechanical, electrical, and software interfaces to provide a common infrastructure for different application-specific modules. Instead of focusing on the development of a single multifunctional device, this work establishes a reusable platform that enables the integration of different mechatronic modules while following a unified system architecture.

To demonstrate the proposed concept, two proof-of-concept modules with different mechanical functionalities were developed and integrated into the platform. In addition to the module development, the presented work includes the design of the electronic hardware, the embedded software architecture, the standardized coupling interface, and the system integration. Finally, experimental validation is performed to verify the functionality of the developed prototype and to evaluate the feasibility of the proposed modular concept.

The remainder of this thesis is organized as follows. Chapter 2 reviews the state of the art in modular prosthetic systems, mechanical coupling mechanisms, and modular mechatronic architectures before identifying the research gap addressed by this work. Chapter 3 introduces the overall system concept and the corresponding design objectives. Chapters 4 to 8 describe the development of the electronic hardware, the mechanical coupling system, the main unit, the interchangeable modules, and the embedded software architecture. Chapter 9 presents an overview of the prototype manufacturing costs, while Chapter 10 provides the experimental validation and eval-

uation of the developed system. Finally, the thesis concludes with a summary of the obtained results and an outlook on possible future developments.

2 State of the Art

2.1 Modular Prosthetic Systems

Recent advances in mechatronics, embedded systems, and sensor technologies have significantly improved the functionality of upper-limb prostheses. Modern prosthetic systems increasingly provide multiple degrees of freedom, intuitive control interfaces, and enhanced dexterity to support activities of daily living. Current research primarily focuses on improving control performance, sensory feedback, modularity, and user acceptance [15].

Modern upper-limb prostheses can generally be divided into commercial systems designed for everyday use and research platforms that investigate novel mechanical designs, control strategies, and system architectures. While commercial devices prioritize reliability and usability, research platforms often explore new concepts intended to improve functionality and adaptability [15].

Figure 2.1: Modular Prosthetic Limb (MPL) – Research platform [14]

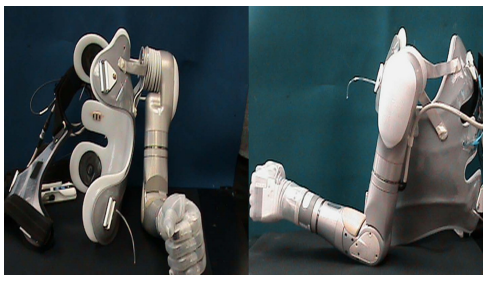


Figure 2.2: DEKA Arm – Commercial prosthetic system [21]

One of the most advanced research platforms is the Modular Prosthetic Limb (MPL) developed by the Johns Hopkins University Applied Physics Laboratory[14]. The system employs a modular architecture with multiple powered degrees of freedom, enabling highly dexterous movements while providing a flexible platform for integrating advanced sensing and control technologies (Figure 2.1) .

Another representative system is the DEKA Arm, which was developed to restore a wide range of

functional movements required for activities of daily living [21]. By integrating multiple powered joints and several grasp patterns, the system demonstrates the continuous development towards more functional and intuitive upper-limb prostheses (Figure 2.2).

Besides improving anthropomorphic prosthetic hands, recent research increasingly investigates modular prosthetic concepts. Instead of using a single universal hand, interchangeable terminal devices allow the prosthesis to be adapted to specific tasks, thereby improving overall functionality and user performance (Figure 2.3) [7].

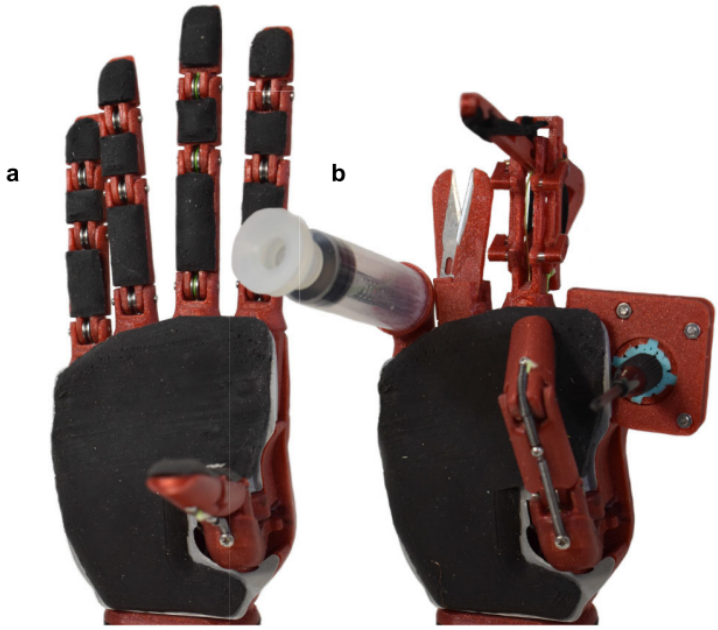


Figure 2.3: Beyond Humanoid Prosthetic Hands

A similar approach is presented by Bingham et al., who developed a modular prosthetic wrist featuring a quick-swap mechanism, continuous electrical connectivity, and interchangeable EMG-controlled attachments. Their work illustrates the growing interest in standardized mechanical and electrical interfaces that enable users to exchange powered end-effectors while maintaining both power and signal transmission (Figure 2.4)[6].

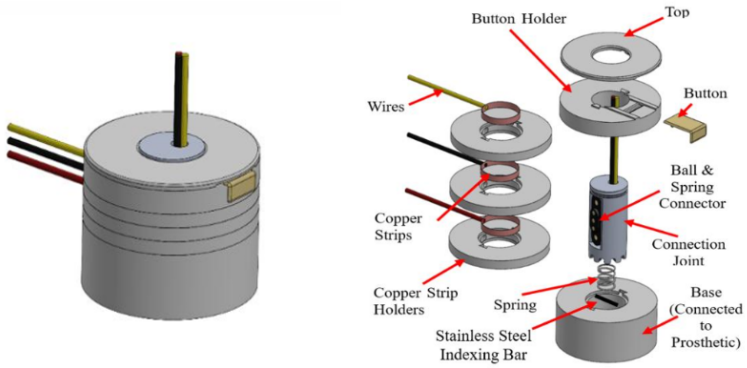


Figure 2.4: CAD model and exploded view of the modular prosthetic wrist proposed by Bingham et al

Overall, current developments indicate a clear trend towards modular prosthetic systems that improve adaptability, functionality, and user independence. While significant progress has been achieved in the development of advanced prosthetic devices, standardized modular platforms for interchangeable mechatronic gadgets remain an active field of research[15][6]. This forms the motivation for the work presented in this thesis.

2.2 Mechanical Coupling Mechanisms

The mechanical coupling mechanism is a fundamental component of every modular prosthetic system, as it enables interchangeable modules to be securely connected while allowing fast and intuitive replacement. Besides providing sufficient mechanical strength, the coupling must ensure reliable load transmission, repeatable positioning, and simple user operation. Consequently, the mechanical interface has a significant influence on the overall functionality and usability of modular prosthetic systems[3].

The increasing use of interchangeable prosthetic modules has led to growing interest in mechanical interfaces that enable rapid and reliable module exchange. As modular prosthetic concepts continue to evolve, the coupling mechanism becomes a key element for supporting task-specific terminal devices while maintaining a common mechanical interface. In addition to prosthetic applications, quick-change systems developed for engineering applications demonstrate how standardized interfaces can reduce exchange time while preserving accurate positioning and structural integrity[7][8].

The design of a coupling mechanism requires balancing several often conflicting requirements. Fast module replacement should be achieved without sacrificing mechanical robustness, positional accuracy, or ease of operation. Furthermore, the interface should support repeated attachment and removal while maintaining reliable performance throughout its service life. These considerations are particularly important for modular prosthetic systems, where users may frequently

exchange terminal devices depending on the intended activity[3][8].

Among the reviewed literature, the work of Bingham et al. is particularly relevant to this thesis, as it presents a modular prosthetic wrist based on a quick-swap mechanism, integrated pogo-pin electrical contacts, and additive manufacturing as the primary fabrication method. These design principles serve as the primary inspiration for the mechanical coupling system developed in this work and are adapted to the specific requirements of the proposed modular prosthetic platform. [6]

2.3 Modular Mechatronic Architectures

The increasing complexity of modern mechatronic systems has led to the development of modular system architectures, in which complex functionalities are distributed across multiple independent modules rather than being implemented within a single centralized unit. Although these concepts have been developed for a wide range of mechatronic applications beyond prosthetics, such as industrial automation and modular robotic systems, the underlying architectural principles are directly applicable to the development of modular prosthetic platforms. By decomposing a system into functional modules with standardized interfaces, modular architectures improve scalability, maintainability, and the reusability of both hardware and software components. Furthermore, independent module development simplifies system integration and enables new functionalities to be added without fundamentally redesigning the overall system [13].

One common implementation of modular mechatronic systems is the use of distributed control architectures. Schlette et al. present such an architecture for distributed robotic systems, where multiple functional modules communicate through standardized interfaces while performing local processing tasks. Similarly, Zhang et al. investigate distributed control concepts for modular robots, demonstrating how local controllers can execute low-level control functions while a higher-level controller coordinates the overall system. Although both approaches originate from modular robotics rather than prosthetics, the same architectural concepts can be applied to prosthetic systems consisting of interchangeable mechatronic modules. This distributed approach reduces the computational load of the central controller while improving flexibility and system scalability [22][26].

The separation of high-level system coordination from local device control provides several practical advantages. Individual modules can be developed, tested, and replaced independently, allowing new functionalities to be integrated without extensive modifications to the existing system. Furthermore, distributing processing tasks among multiple modules improves maintainability and facilitates the extension of the system as additional devices are introduced. These characteristics are particularly beneficial for modular prosthetic platforms, where future gadgets can be integrated without redesigning the complete electronic architecture [13][26].

2.4 The eSuperheroes Project

The work presented in this thesis builds upon the eSuperheroes project developed at MediaLab, which aims to improve the functionality and accessibility of upper-limb prostheses through modular and user-oriented design. The project introduced a family of interchangeable prosthetic gadgets that enable users to adapt their prosthesis to different daily activities by replacing the terminal device according to the required task. This modular approach represents an alternative to the use of a single universal prosthetic hand and has demonstrated the potential of task-specific prosthetic solutions (Figure 2.5)[11].

Figure 2.5: Examples of task-specific prosthetic gadgets developed within the eSuperheroes project for different daily activities

The interchangeable gadgets developed within the eSuperheroes project are primarily based on mechanical interfaces. While this concept successfully enables the rapid replacement of different terminal devices, the exchanged modules do not provide integrated electrical power, electronic control, or communication capabilities. As a result, the functionality of interchangeable gadgets is currently limited to passive mechanical devices or independently operated solutions (Figure 2.6) [11].

Figure 2.6: Examples of mechanically interchangeable prosthetic gadgets and coupling interfaces used in the eSuperheroes project

The objective of this thesis is therefore not to replace the existing eSuperheroes concept, but to extend it by introducing a standardized mechanical and electrical interface for electronically interchangeable mechatronic gadgets. By integrating power supply, communication, and local electronic control into the interchangeable modules, the proposed platform establishes the technological foundation for future powered prosthetic gadgets while preserving the modular philosophy of the original project.

2.5 Research Gap

The reviewed literature demonstrates significant progress in modular prosthetic systems, quick-change coupling mechanisms, and modular mechatronic architectures. Furthermore, the eSuperheroes project has shown that low-cost, task-specific prosthetic gadgets can improve accessibility and functionality by enabling users to interchange mechanical end-effectors according to their daily activities.

However, current prosthetic research is still largely focused on multifunctional prosthetic hands designed to perform a broad range of tasks. Comparatively little research has addressed task-specific modular prosthetic solutions, particularly platforms that combine interchangeable gadgets with standardized mechanical and electrical interfaces, integrated communication, and distributed electronic control.

This thesis extends the vision of the eSuperheroes project by introducing a modular mechatronic

platform for electronically interchangeable prosthetic gadgets. Building upon the existing low-cost and modular philosophy, the proposed system establishes a standardized mechanical, electrical, and software interface that enables future powered gadgets to be integrated without redesigning the underlying platform. In this way, the work represents the next step towards an open, scalable, and affordable ecosystem for modular prosthetic devices.

3 Overall System Concept

3.1 Design Objectives

The objective of this work is to develop a modular prosthetic platform that enables the rapid exchange of application-specific modules while maintaining standardized mechanical, electrical, and software interfaces. The proposed platform extends the concept of the eSuperheroes project by providing a common infrastructure for active mechatronic modules with integrated actuation, and communication.

While conventional anthropomorphic prosthetic hands are designed to perform a wide range of daily tasks using a single multifunctional end-effector, the concept proposed in this work follows a different approach by employing interchangeable modules that are optimized for specific applications. This modular approach allows the functionality of the prosthesis to be adapted to the user's current task by replacing the attached module rather than relying on a single universal device.

The overall system concept is derived from the analysis presented in Chapter 2. The reviewed literature highlights the benefits of modular prosthetic systems, standardized coupling mechanisms, and distributed mechatronic architectures. Building upon these findings, the developed platform combines a standardized coupling interface with reusable electronics and a common software architecture to simplify the development and integration of future prosthetic modules.

Table 3.1 summarizes the main design objectives that guided the development of the proposed platform.

Table 3.1: Design objectives of the proposed modular prosthetic platform.

Objective	**Description**
Modularity	Support interchangeable prosthetic modules using a standardized interface.
Standardized coupling	Provide a mechanical and electrical interface that enables repeatable module exchange.
Reusable electronics	Use a common secondary PCB for different prosthetic modules.
Distributed control	Implement local module control while communicating through a CAN bus.
Low-cost implementation	Utilize commercially available components and additive manufacturing to reduce production costs.
Scalability	Allow the integration of additional prosthetic modules without modifications to the platform architecture.
Demonstration platform	Validate the concept by implementing two interchangeable modules with different mechanical functions.

3.2 System Architecture

The overall system architecture of the proposed modular prosthetic platform is illustrated in Figure 3.1. The platform follows a hierarchical architecture consisting of a high-level master controller, a unified communication interface, and interchangeable application-specific modules. This organization separates system-wide functionality from module-specific functionality while providing a common framework for all modules.

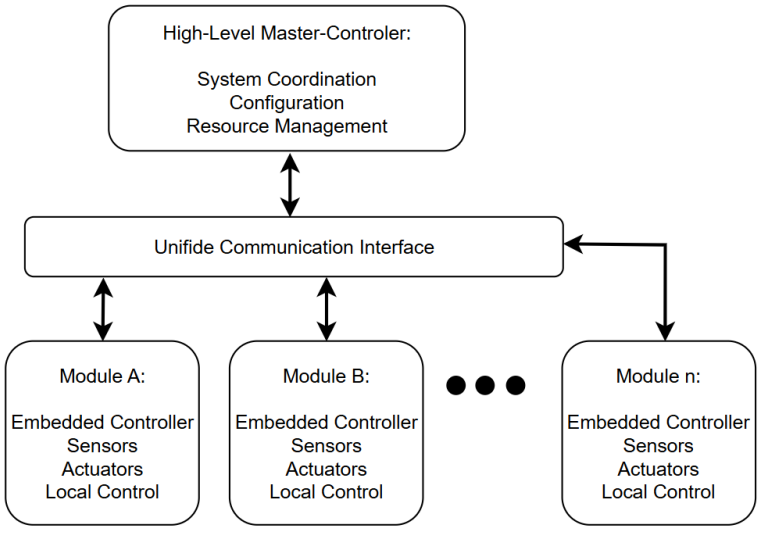


Figure 3.1: Conceptual architecture of a modular mechatronic system derived from the reviewed literature

The high-level master controller is responsible for the overall coordination of the platform, including system management, configuration, and the exchange of information with the connected modules. The unified communication interface provides a standardized connection between the master controller and the individual modules, allowing all modules to communicate through the same architecture regardless of their specific functionality.

Each module combines standardized electronics with application-specific sensors, actuators, and embedded control. This approach enables different mechanical modules to share a common hardware and software architecture while implementing different functions. Consequently, additional modules can be integrated by following the established platform architecture, ensuring compatibility with the overall system without requiring fundamental changes to the system concept.

4 Electronics

This chapter describes the selection of the required electronic components as well as the design and development electronics.

4.1 System Requirements

This chapter presents the fundamental concept of the developed electronic system and the corresponding system requirements derived from it. Based on these requirements, the selection of the electronic components and the design of the individual circuit sections are subsequently described.

The overall electronic system is divided into two functional units: a Master Unit and a Slave Unit. The Master Unit is integrated into the main electronics of the prosthetic system and is responsible for the central control of the entire system. It contains the microcontroller, the CAN communication interface, and the power supply for all connected modules. In addition, it acquires user input through various sensors, such as an EMG sensor or other input devices, and processes these signals into control information.

Communication between the Master Unit and the Slave Unit is established through a common connector that provides both the power supply and the CAN communication lines. This interface allows different functional modules to be connected and operated using the same electrical connection.

Figure 4.1 illustrates the basic communication architecture of the electronic system. The sensor data acquired by the Master Unit is transmitted to the connected gadget via the CAN bus, where it serves as the input for actuator control.

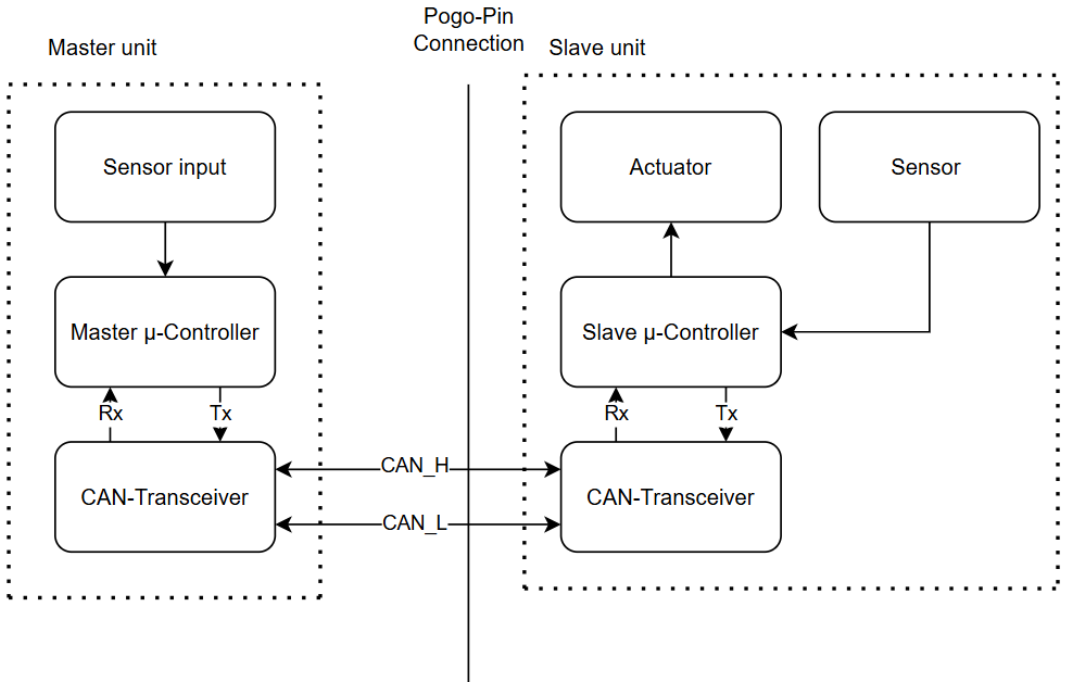


Figure 4.1: Electronic Communication Diagramm

The Slave Unit represents the fundamental electronic architecture of the interchangeable functional modules. Each gadget contains its own microcontroller, a CAN transceiver, and the sensors and actuators required for its specific application. The control information transmitted by the Master Unit is received via the CAN interface and processed locally by the respective gadget. Based on this information, each module controls its corresponding actuator independently while the Master Unit remains responsible only for the overall system coordination.

4.2 Selection of the Microcontroller

Based on the system requirements, the requirements for the central component of the electronics, namely the microcontroller, can be derived. The relevant requirements can be categorized into the areas of communication, peripherals, performance, and form factor.

Category	**Requirement**
Communication	CAN bus support (native or via external transceiver)
	Wi-Fi connectivity
Peripherals	At least 2 PWM-capable outputs (motor control)
	At least 2 ADC inputs (EMG sensor, potentiometer)
	At least 4 GPIO pins
Performance	At least 80 KB RAM (XML parsing using TinyXML2)
	At least 1 MB Flash memory (firmware and filesystem)
Form Factor	Compact board design
	Cost-effective and readily available

Table 4.1: Microcontroller requirements

Based on the defined requirements, four microcontrollers were identified as potential candidates and compared with one another. In the following, the individual candidates are presented and their characteristics are evaluated with respect to the defined requirements.

The Arduino Mega 2560 is a widely used microcontroller and is based on the ATmega2560 [2].

Parameter	Value
Clock Frequency	16 MHz
RAM / Flash	8 KB / 256 KB
GPIO Pins	54
ADC Channels	16
CAN Bus	No native support (external MCP2515 required)
Wi-Fi	Not available

Table 4.2: Technical specifications of the Arduino Mega 2560

This microcontroller is not suitable for the requirements of the present system, as neither CAN communication nor Wi-Fi are supported natively. In addition, the available processing power and memory capacity are insufficient for XML parsing.

The STM32F103C8T6 is a high-performance microcontroller based on the ARM Cortex-M3 architecture, which is widely used in industrial and professional embedded systems [12].

Parameter	Value
Clock Frequency	72 MHz
RAM / Flash	20 KB / 64 KB
GPIO Pins	37
ADC Channels	10
CAN Bus	Native support
Wi-Fi	Not available

Table 4.3: Technical specifications of the STM32F103C8T6

The STM32 satisfies the requirements regarding CAN communication and processing performance. However, it does not provide an integrated Wi-Fi module, making additional external hardware necessary for wireless communication.

The Raspberry Pi Pico W is a modern microcontroller based on the RP2040 chip and features an integrated Wi-Fi module [20].

Parameter	Value
Clock Frequency	133 MHz
RAM / Flash	264 KB / 2 MB
GPIO Pins	26
ADC Channels	3
CAN Bus	No native support (external MCP2515 required)
Wi-Fi	(802.11 b/g/n)

Table 4.4: Technical specifications of the Raspberry Pi Pico W

The Pico W provides integrated Wi-Fi functionality, sufficient processing performance, and a com-

pact form factor. However, it does not include a native CAN controller, and the limited number of available ADC channels restricts sensor integration.

The ESP32 is a compact microcontroller based on the ESP32 chip from Espressif, integrating Wi-Fi, Bluetooth, and a native CAN controller (TWAI) within a single system [10].

Parameter	**Value**
Clock Frequency	240 MHz
RAM / Flash	520 KB / 4 MB
GPIO Pins	32
ADC Channels	16
CAN Bus	Native support (TWAI controller)
Wi-Fi	(802.11 b/g/n)

Table 4.5: Technical specifications of the ESP32

The ESP32 is the only candidate among those evaluated that natively integrates both a CAN controller (TWAI) and Wi-Fi, while also providing sufficient RAM and Flash memory for XML parsing and an embedded filesystem. Compared to the STM32, no additional hardware is required for wireless communication. Compared to the Raspberry Pi Pico W, no external CAN controller is needed. The Arduino Mega 2560 meets neither the communication nor the performance requirements. For the physical implementation, the ESP32 D1 Mini development board was selected. Due to its compact form factor it is particularly well suited for integration into a prosthetic device where space is limited.

4.3 Selection of Sensor Inputs

EMG stands for electromyography. In general, a distinction is made between invasive and non-invasive electromyography. It has been shown that surface electromyography (sEMG) is a reliable method for recording muscle activity ([25] chapter 9.3.1.).

For this reason, this work focuses exclusively on surface electromyography (sEMG).

The human nervous system utilizes the movement of ions for information transmission. The activation of muscle fibers generates electrical potentials that propagate to the surface of the skin. In contrast, electronic measurement systems are based on the movement of electrons within metallic conductors [24].

To bridge this physical interface between biological and electronic systems, electrodes are employed. These act as transducers between the ionic processes within the body and the electrical signals that can be processed by the measurement electronics (Figure 4.2) [24].

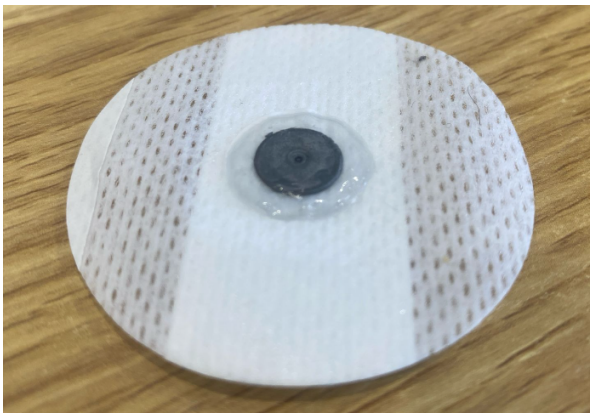


Figure 4.2: Electrode

A surface electrode consists of a metal electrode and a conductive gel that improves the electrical contact with the skin (Figure 4.2). When a muscle is activated, the movement of ions generates electrical potentials at the skin surface. These potentials are transmitted through the conductive gel to the metal electrode [24].

When multiple electrodes are placed at different positions on the skin, the electrical potentials at the respective measurement points can be recorded. A differential amplifier subsequently determines the voltage difference between the electrodes. This voltage difference constitutes the actual sEMG signal [24].

Various approaches exist for signal processing, including both analog and digital methods. However, the underlying principle remains the same in both cases. First, the signal is amplified using a differential amplifier. Subsequently, unwanted noise components are suppressed by appropriate filtering stages. The signal is then rectified and smoothed, allowing it to be readily acquired by an analog-to-digital converter (ADC)[16][5][19] [1] .

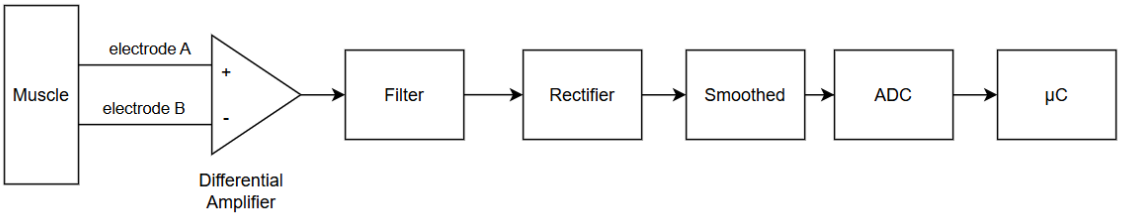


Figure 4.3: Processing Chain EMG-Signal

In the presented figure, the EMG signal is illustrated as a sinusoidal waveform for the purpose of simplification. This representation does not imply that the actual EMG signal exhibits a sinusoidal behavior. Rather, it serves to demonstrate that the signal contains both positive and negative voltage components, which necessitates rectification during signal processing (Figure 4.4).

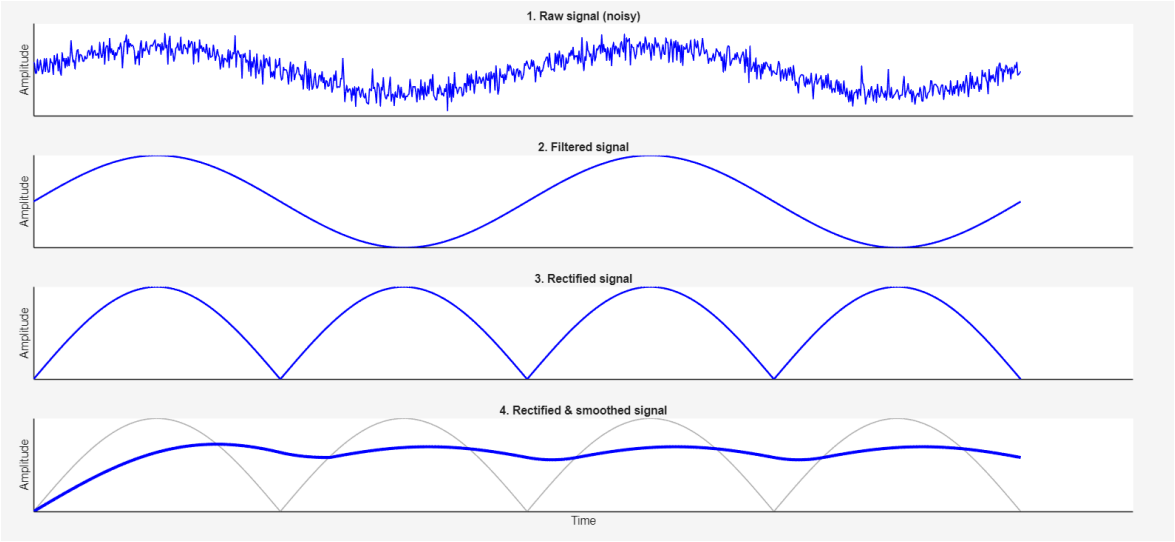


Figure 4.4: Processing of EMG-Signal

Selection of the EMG Sensor

Based on the literature review and market analysis, three key selection criteria were identified for evaluating and selecting a suitable surface electromyography (sEMG) sensor.

Cost-effectiveness (Low-Cost): The sensor module should be affordable for student laboratory projects and academic prototyping. Therefore, the target cost per sensor unit is within the lower price range.

Market availability: To ensure rapid procurement, the sensor should be readily available through established electronics distributors or common online retailers. This minimizes lead times associated with international sourcing.

Power supply architecture: From a technical perspective, sensor modules can be categorized into single-supply (asymmetric) and dual-supply (symmetric) systems. For straightforward system integration, a single-supply sensor is preferred, as it requires only a supply voltage and ground (GND). This reduces circuit complexity and enables direct interfacing with a microcontroller.

Based on these selection criteria, three widely used surface EMG sensor modules were compared: the EMG Sensor V3.0 by Advancer Technologies, the MyoWare 2.0, and the DFRobot SEN0240 (Tabel4.6).

Criterion	EMG-Sensor-V3.0	MyoWare 2.0	SEN0240
Approx. Price	~22 €	~43 €	~50 €
Market Availability	High	Limited	Limited
Power Supply Architecture	Dual-Supply (±3.5 V)	Single-Supply (3.3–5 V)	Single-Supply (3.3–5.5 V)
Additional Circuitry Required	Yes	No	No

Table 4.6: Comparison of the evaluated sEMG sensor modules

The EMG Sensor V3.0 by Advancer Technologies measures, filters, rectifies, and amplifies electrical muscle activity and provides an analog output signal that can be directly processed by the analog-to-digital converter (ADC) of a microcontroller. However, the module requires a dual-supply voltage of at least ±3.5 V. The MyoWare 2.0 is an improved version of the EMG Sensor V3.0 from the same manufacturer and features a simplified single-supply architecture operating from 3.3 V to 5 V. In addition, the module provides three different output signals: the raw signal (RAW), the rectified signal, and the smoothed envelope signal. This makes it particularly suitable for applications requiring more advanced signal processing. The DFRobot SEN0240 is another analog sEMG sensor module that operates from a single supply voltage between 3.3 V and 5.5 V. The module amplifies the muscle signal by a factor of 1000 and provides an analog output voltage ranging from 0 V to 3.0 V with a reference voltage of 1.5 V. With respect to the defined selection criteria, both the MyoWare 2.0 and the SEN0240 fully satisfy the technical requirements while offering the advantage of a simple single-supply architecture. However, at the time of development, both modules were only available from established distributors such as DigiKey with long delivery times. In addition, shipping costs were nearly equivalent to the purchase price of the sensor itself, making both options impractical within the limited timeframe of this project. In contrast, the EMG Sensor V3.0 was readily available in the laboratory and could therefore be integrated into the system without delay. Since the proposed control strategy relies exclusively on threshold-based activation, the signal quality provided by the EMG Sensor V3.0 is entirely sufficient for the intended application.

For future development of the project, it would be advisable to re-evaluate the selection of the sEMG sensor module. In particular, the MyoWare 2.0 or the SEN0240 could reduce circuit complexity due to their single-supply architecture and enable easier integration into a more compact PCB design.

Since the muscle sensor is currently used only to activate the gadget, an additional input was introduced to demonstrate system control. For this purpose, a potentiometer is used, which can represent different control behaviours depending on the connected gadget.

Figure 4.5: Potentiometer

4.4 Electronic System

After selecting the main electronic components, the next step is to design the power supply for the overall system.

Most components, including the microcontroller, the CAN transceiver, and the servo motor, require a supply voltage of 5 V. The power supply is dimensioned based on the maximum current consumption of each individual component.

No official datasheet could be found for the ESP32 D1 mini board used in this project to determine its current consumption. However, the ESP32 chip datasheet includes a section on power supply design. For the ESP32 chip, a supply voltage of 3.3 V with a maximum current of 500 mA is recommended.

Since the ESP32 D1 mini board is supplied with 5 V, the maximum input current of the board is estimated based on the power balance:

$$I_{ESP32} = \frac{500\,\text{mA} \cdot 3.3\,\text{V}}{5\,\text{V}} = 330\,\text{mA} \tag{4.1}$$

The average current consumption of the TJA1050 CAN transceiver is estimated based on the transmitted CAN messages and the resulting bus load. For this calculation, a worst-case assumption is

adopted in which the entire CAN frame is assumed to be transmitted in the dominant state.

Furthermore, it is assumed that both the potentiometer values and the sEMG sensor measurements are transmitted over the CAN bus every 10 ms.

For the current consumption in both the dominant and recessive states, the maximum values specified in the TJA1050 datasheet are used (Table: 4.7) [17].

Table 4.7: Parameters used for the current consumption calculation

Parameter	**Value**	**Source**
CAN Bit Rate	500 kBit/s	Firmware configuration
Transmission Interval	10 ms	Firmware configuration
Data Length `emgData`	2 Byte	Firmware configuration
Data Length `potiData`	4 Byte	Firmware configuration
Supply Current (Dominant State) (I_{dom})	75 mA	TJA1050 datasheet
Supply Current (Recessive State) (I_{rec})	10 mA	TJA1050 datasheet

Frame Length

A standard CAN frame with an 11-bit identifier consists of a fixed overhead of 47 bits in addition to the payload field. The overhead is composed of the protocol fields defined in ISO 11898-1 and is summarized in Table 4.8.

Table 4.8: Structure of a Standard CAN Frame (11-bit Identifier)

Field	**Bits**	**Description**
Start of Frame (SOF)	1	Start bit, always dominant
Identifier	11	11-bit message identifier
RTR	1	Remote Transmission Request
IDE	1	Identifier Extension bit
r0	1	Reserved bit
DLC	4	Data Length Code
CRC	15	Cyclic Redundancy Check
CRC Delimiter	1	Delimiter bit
ACK Slot	1	Acknowledgement slot
ACK Delimiter	1	Delimiter bit
EOF	7	End of Frame
IFS	3	Interframe Space
Total Overhead	**47**	

The total frame length of the two CAN messages is obtained by adding the fixed frame overhead to the corresponding payload length:

$$L_{\text{emg}} = 47 + 2 \times 8 = 63\,\text{Bit} \tag{4.2}$$

$$L_{\text{poti}} = 47 + 4 \times 8 = 79\,\text{Bit} \tag{4.3}$$

Frame Transmission Time

At a CAN bit rate of 500 kBit/s, the transmission time of each CAN frame is determined by its frame length and the bus bit rate:

$$t_{\text{emg}} = \frac{63}{500.000} = 0{,}126\,\text{ms} \tag{4.4}$$

$$t_{\text{poti}} = \frac{79}{500.000} = 0{,}158\,\text{ms} \tag{4.5}$$

$$t_{\text{active}} = t_{\text{emg}} + t_{\text{poti}} = 0{,}285\,\text{ms} \tag{4.6}$$

Duty-Cycle

Since both CAN messages are transmitted with a transmission interval of $T = 10\,\text{ms}$, the duty cycle is determined by the ratio of the total transmission time within one transmission interval to the corresponding period:

$$DC = \frac{t_{active}}{T} = \frac{0{,}285ms}{10ms} = 2{,}85\,\% \tag{4.7}$$

Average Current Consumption

The average current consumption of the CAN transceiver is calculated as the weighted average of the current consumption in the dominant and recessive states. The weighting is based on the previously calculated duty cycle:

$$I_{\text{TJA1050}} = I_{\text{dom}} \cdot D_{\text{total}} + I_{\text{rec}} \cdot (1 - D_{\text{total}}) \tag{4.8}$$

$$I_{\text{TJA1050}} = 75\,\text{mA} \times 0.0284 + 10\,\text{mA} \times 0.9716 \tag{4.9}$$

$$I_{\mathrm{TJA1050}} = 2.13\,\mathrm{mA} + 9.72\,\mathrm{mA} \approx \mathbf{11.85}\,\mathrm{mA} \tag{4.10}$$

Under the adopted worst-case assumption, the TJA1050 exhibits an average current consumption of approximately 12 mA per CAN transceiver during normal operation at a transmission rate of 100 Hz.

Actuators

The gripper servo motor is also powered by the 5 V supply. According to the datasheet, the servo can draw a peak current of up to 2.1 A. This value is used as the worst-case current for sizing the power supply.

Design of the 5 V DC-DC Converter

At this stage, all components supplied by the 5 V DC-DC converter have been identified. Based on the previously determined maximum current consumption of each load, the required output current of the converter can now be determined.

Table 4.9: Calculated current consumption for the 5 V supply

Component	**Peak current at 5 V**
2× ESP32 D1 Mini	660 mA
2× TJA1050 CAN transceivers	12 mA
25 kg Servo	2100 mA
Total	**2.772 A**

In parallel with the theoretical calculations, initial experimental investigations were carried out.

In this experiment, the current consumption of all electronic components supplied by the 5 V power supply is measured. The objective is to validate the calculated values experimentally and to determine the actual current consumption of the system during operation 4.6.

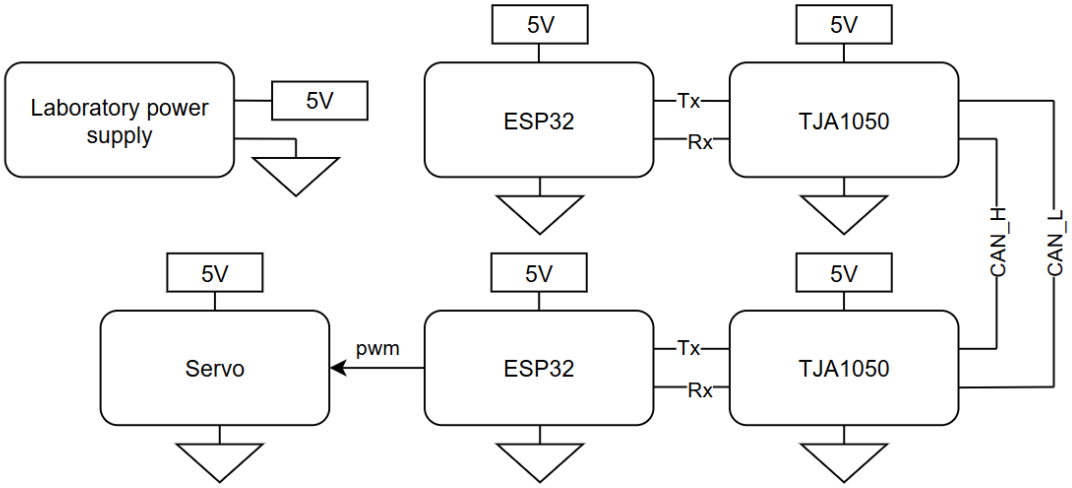


Figure 4.6: Experimental Setup

The measurement results show a significantly lower current consumption than predicted by the theoretical worst-case calculation. While the calculated maximum current consumption is 2.772 A, a maximum current of approximately 1.3 A was measured during operation with the servo motor active (Table: 4.10).

This discrepancy can be explained by the fact that the theoretical calculation is based on the maximum specified current consumption of all components and assumes that these peak currents occur simultaneously. In the experimental setup, the servo motor did not reach the peak current of 2.1 A specified in the datasheet. Consequently, the overall current consumption of the system was lower than predicted by the theoretical worst-case scenario. The measurements therefore confirm that the calculation provides a conservative basis for sizing the power supply.

Table 4.10: Measured current consumption of the 5 V system

Operating Condition	**Voltage**	**Current**	**Power**
Servo motor idle	5 V	245 mA	1.23 W
Servo motor under load	5 V	1.3 A	6,5 W

Based on the theoretical analysis and the experimental measurements, a DC-DC converter capable of delivering at least 10 W (5 V / 2 A) is sufficient to supply the developed system.

For the prototype implementation, the XL6009 DC-DC converter module was selected. The module was available in the laboratory and was integrated into the power supply to replace the laboratory power supply during system testing. The experimental evaluation confirmed stable operation under the expected load conditions. Since the converter satisfied the electrical requirements of the system, it was selected for the subsequent prototype implementation.

Design of the 12 V Power Supply

The 12 V power supply is used exclusively for the DC gear motor of the corresponding gadget. Therefore, the DC-DC converter can be dimensioned directly based on the electrical characteristics of the motor.

A JGA25-371 DC gear motor is used in the prototype. According to the datasheet, the motor has a maximum stall current of 1 A at a supply voltage of 12 V. To verify this specification, the motor was supplied with 12 V while the rotor was mechanically prevented from rotating. Under these conditions, a current of approximately 1 A was measured, confirming the datasheet value.

Since the stall current represents the maximum expected load, it was used to dimension the 12 V power supply. The required output power is therefore

$$P = U \cdot I = 12\,\mathrm{V} \times 1\,\mathrm{A} = 12\,\mathrm{W} \tag{4.11}$$

Consequently, the selected DC-DC converter must be capable of providing at least 12 W of output power.

For the prototype, the XL6009 DC-DC converter module was also selected for the 12 V power supply, as its electrical specifications satisfy the required output power of the JGA25-371 DC gear motor.

Negative Voltage

Zum Schluss fehlt noch die negative Spannung für den Muskelsensor. Wie zuvor schon besprochen, benötigt der ausgewählte Muskelsensor eine Positive und negative Spannungsversorgung. Um nun keine weitere Batterie nutzen zu müssen muss die Spannung mit einer weiteren Schaltung invertiert werden. Um die negative Spannungsversorgung zu erzeugen wird der IC MAX1044 verwendet. Das ist eine sogenannte Ladungspumpe.

The final power rail required by the system is the negative supply voltage for the selected sEMG sensor. As described previously in the sensor selection section, the selected sensor requires both a positive and a negative supply voltage for proper operation.

To avoid the need for an additional battery or a dedicated negative voltage source, the required negative supply voltage is generated from the existing positive supply. For this purpose, the MAX1044 integrated circuit is used. The MAX1044 is a switched-capacitor voltage inverter (charge pump) that generates a negative output voltage from a positive input voltage.

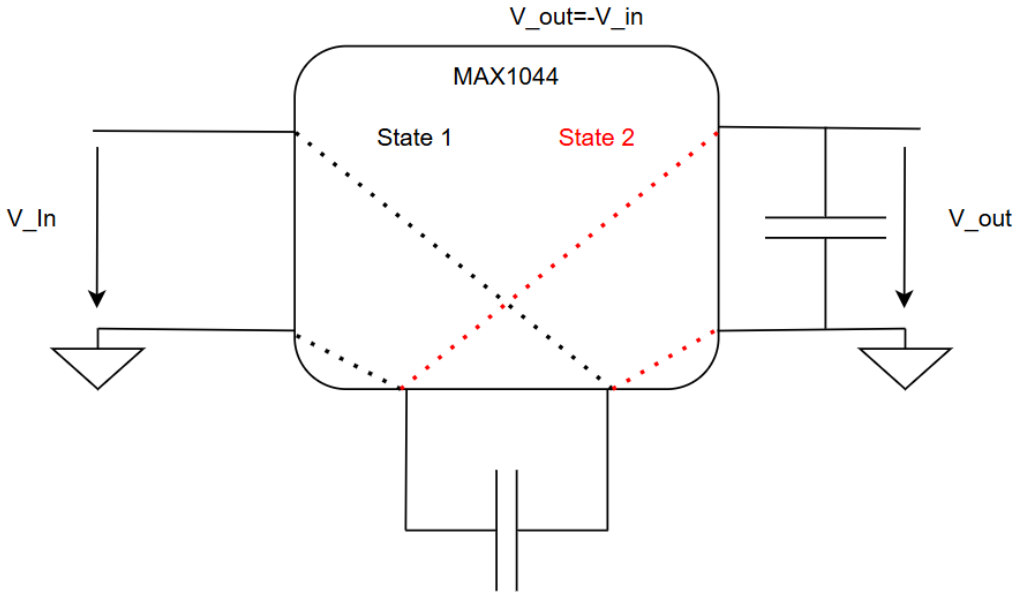


Figure 4.7: Operating principle of the MAX1044 charge pump

The charge pump operates by periodically switching between two operating phases. During the first phase, the flying capacitor is charged to the positive input voltage. In the second phase, the capacitor is disconnected from the input and connected to the output with reversed polarity, thereby generating a negative output voltage (Figure 4.7).

This switching process operates at a frequency of approximately 5 kHz. An additional output capacitor smooths the generated voltage and enables a continuous output current of approximately 10 to 20 mA.

As part of the project, experimental investigations were carried out to determine suitable capacitance values for the two capacitors required by the MAX1044. According to the datasheet, both the flying capacitor and the output capacitor should have a capacitance of 10 µF.

During testing, however, significant voltage drops were observed on the negative supply rail when the sEMG sensor was connected as the load. This indicated that the capacitance recommended in the datasheet was insufficient for the present application. Therefore, different capacitor values were evaluated experimentally to ensure a stable negative supply voltage (Table 4.11).

Table 4.11: Measured output voltage for different capacitor values

Capacitance	U_{in} **[V]**	U_{out} **[V]**	$\Delta U = \|\|U_{in}\| - \|U_{out}\|\|$ **[V]**
10 µF	5.0	-1.2	3.8
22 µF	5.0	-4.6	0.4
100 µF	5.0	-4.8	0.2
220 µF	5.0	-4.8	0.2

When the sEMG sensor was connected as the load, the recommended 10 µF capacitors specified in the datasheet proved insufficient for the present application. Increasing the capacitance to 22 µF significantly improved the output voltage. Beyond 100 µF, however, only a marginal improvement in the output voltage was observed.

During testing, it was also observed that the output voltage was not consistently stable when very large capacitance values were used. A possible explanation is the increased charging time of the capacitors in combination with the fixed switching frequency of 5 kHz of the MAX1044. Therefore, two 100 µF capacitors were selected for the final circuit design, as they provide a suitable compromise between output voltage stability and converter performance.

4.5 Electronics Architecture

With all required supply voltages available, the complete main electronics circuit can now be designed.

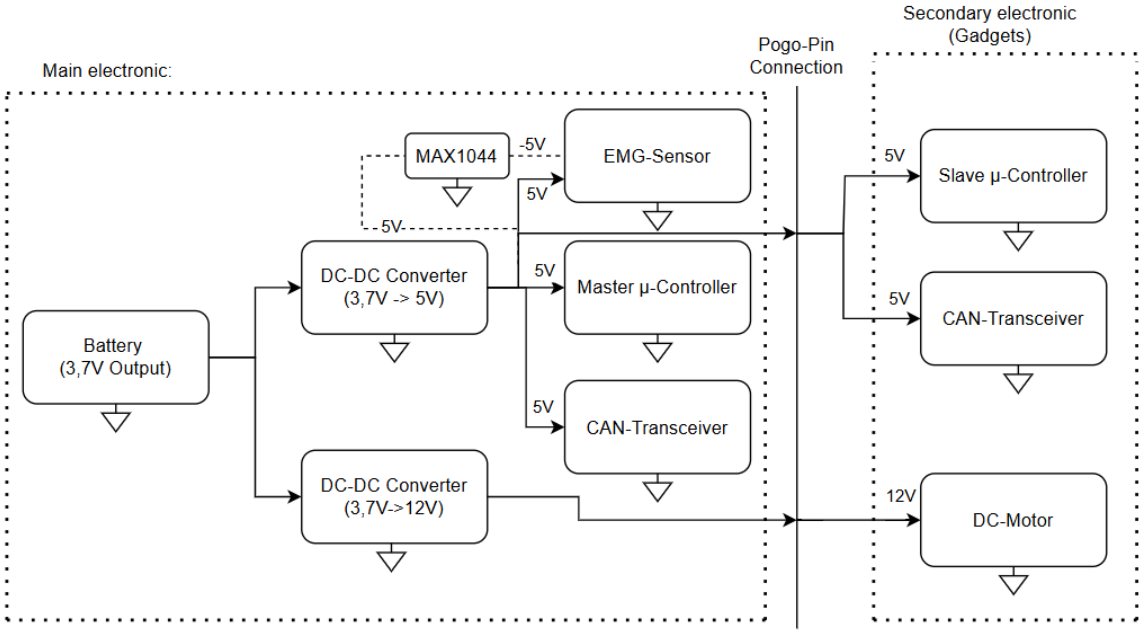


Figure 4.8: Power supply architecture of the developed system

Figure 4.8 illustrates the overall power supply architecture of the developed electronic system. The system is powered by a rechargeable battery, which is intended to provide sufficient capacity for a full day of operation without recharging.

Two DC-DC converters are used to generate the required supply voltages. The 5 V DC-DC converter supplies the majority of the electronic circuitry, as most components operate from a 5 V supply. In addition, a 12 V DC-DC converter is required to power the DC gear motor of the first

gadget (Rotator).

During the system design process, the introduction of an additional 3.3 V supply rail for future extensions, such as additional sensors, was also considered. However, since the ESP32 D1 mini provides a regulated 3.3 V output capable of supplying external modules with currents of up to 150 mA, which is sufficient for most sensors, a dedicated 3.3 V power rail was omitted. This simplifies the power supply architecture and reduces the overall circuit complexity.

During the implementation of the negative supply voltage, experimental testing revealed that the MAX1044 charge pump could not be operated reliably when supplied by the 5 V DC-DC converter. Since the MAX1044 generates the negative output voltage by periodically charging and discharging an external capacitor, it presents a high inrush current during startup. From the perspective of the DC-DC converter, this behavior is comparable to connecting a discharged capacitor directly to its output. As a result, startup currents of up to 3 A were measured, exceeding the operating limits of the MAX1044 and causing permanent damage to the device.

To eliminate this issue, the power supply architecture was modified. Instead of being supplied by the 5 V DC-DC converter, the MAX1044 is connected directly to the battery. This configuration significantly reduces the startup current stress and ensures reliable operation of the negative voltage generation circuit.

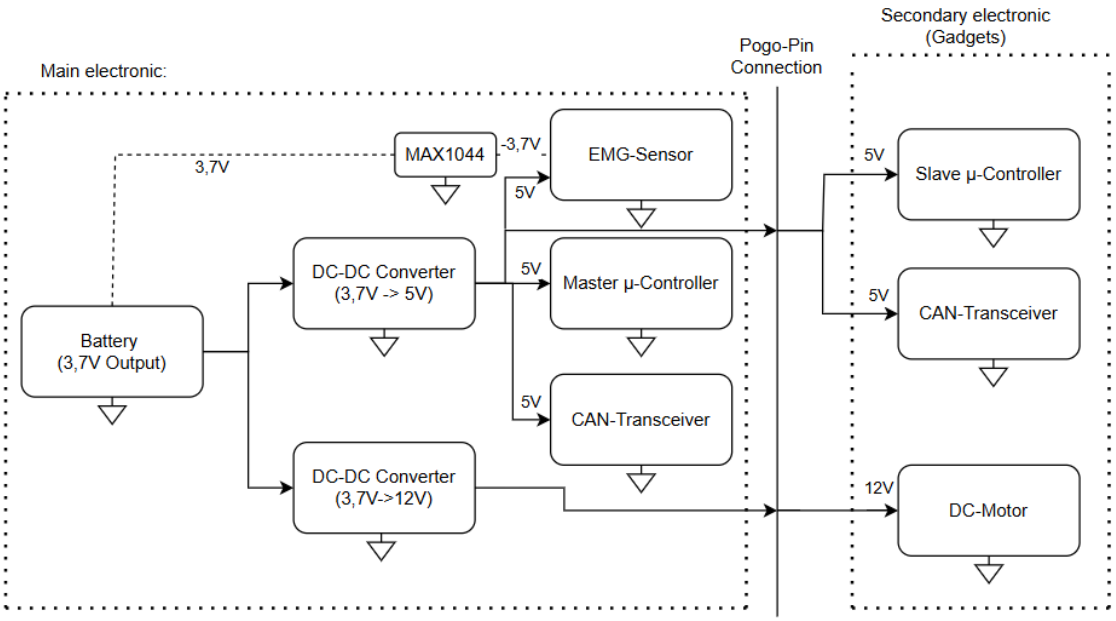


Figure 4.9: Final Power supply architecture of the developed system

4.6 Hardware Implementation

As described in the system concept, the electronics are divided into two functional units: the main electronics and the secondary electronics.

The main electronics include the complete power supply of the system, including the DC-DC converters and the circuit for generating the negative supply voltage required by the sEMG sensor. In addition, this PCB contains the sensors used to control the gadgets, as well as the master microcontroller and its associated CAN transceiver.

The connection between the main electronics and the interchangeable gadgets is established via a six-pin pogo-pin interface. This interface provides the 5 V and 12 V supply rails, the CAN communication lines, and a common ground connection. The pin assignment of the pogo-pin interface is summarized in Table 4.12.

Table 4.12: Pin assignment of the main PCB terminal connector

Pin	Signal	Description
1	GND	Ground
2	12 V	12 V power supply
3	CAN_L	CAN bus low (D-)
4	5 V	5 V power supply
5	NC	Not connected
6	CAN_H	CAN bus high (D+)

Figure 4.10 shows the implemented 3D model of the main PCB. In addition to the ESP32 microcontroller and the CAN transceiver, the board integrates the complete power supply, including the DC-DC converters, the negative voltage generation circuit for the sEMG sensor, and the user input devices. Furthermore, the power and communication connector provides the interface to the interchangeable gadgets.

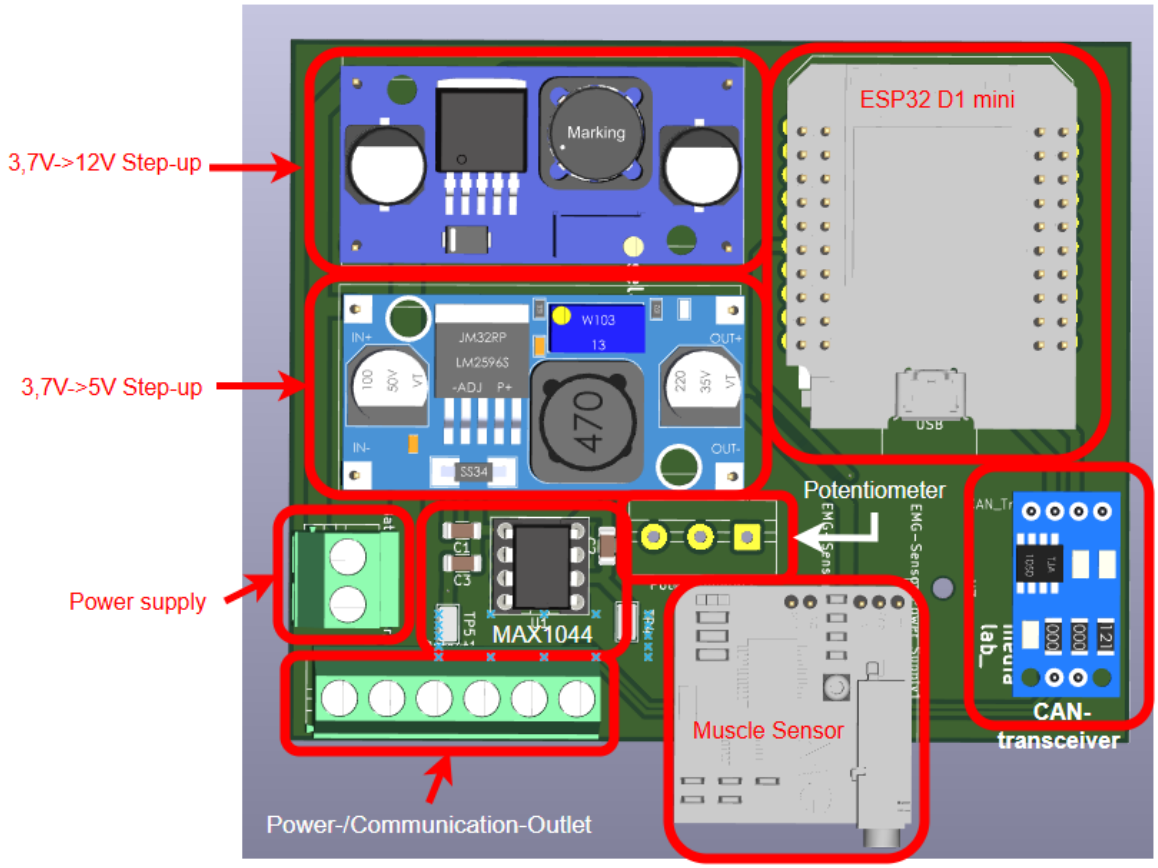


Figure 4.10: Master PCB 3D-Model

The secondary electronics consist of an ESP32 microcontroller and a CAN transceiver, forming a standardized control PCB for all gadgets. Pin headers provide the power supply, the CAN communication interface to the main electronics, and the connections for sensors and actuators. As a result, different gadgets can be implemented on the same hardware platform, with only the required sensors and actuators varying between applications.

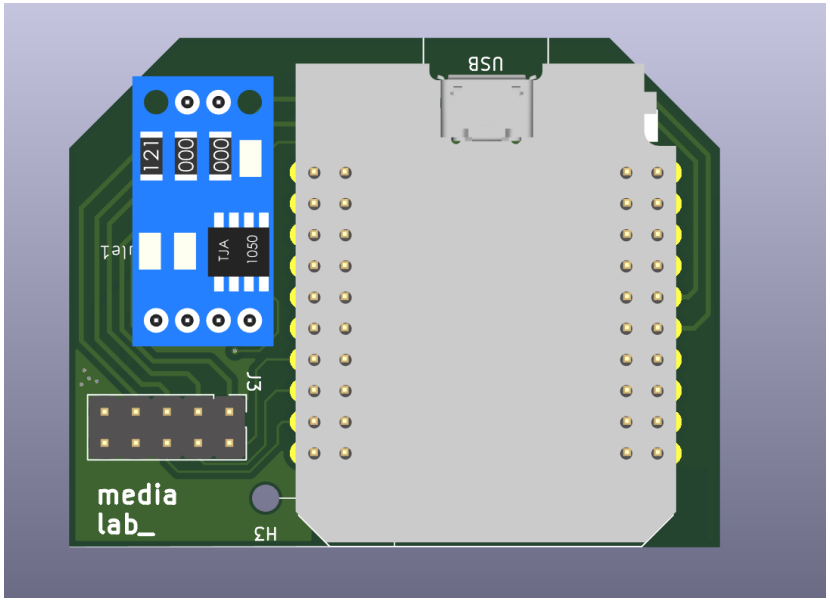


Figure 4.11: Slave PCB 3D-Model

The pin assignment of the standardized secondary electronics is summarized in Table 4.13. In addition to the 5 V and 3.3 V supply rails and the CAN interface, several ESP32 GPIO pins are routed to the pin header. This allows sensors and actuators to be connected flexibly to the PCB. The selected pin assignment provides sufficient input and output signals for the gadgets developed in this work while also supporting future extensions.

Table 4.13: Pin assignment of the secondary electronics connector

Pin	Signal	Description
1	GPIO34	Analog input (ADC1)
2	GPIO32	Digital I/O / PWM
3	GPIO33	Analog input (ADC1)
4	GPIO25	Digital I/O / DAC
5	CAN_H	CAN bus high
6	GPIO27	Digital I/O
7	CAN_L	CAN bus low
8	GND	Ground
9	3.3 V	Sensor supply
10	5 V	Power supply

Although the overall electronics architecture and the PCB interfaces have now been defined, two important hardware design aspects remain to be addressed. First, the electrical design of the pogo-pin interface must ensure reliable power transmission and CAN communication between the main electronics and the interchangeable gadgets. Second, the rechargeable battery must be dimensioned to provide sufficient energy for a full day of operation while meeting the power requirements of the complete system.

Integration of the Secondary PCB into the Gadgets

The standardized secondary PCB is used as the common control and communication platform for both gadgets. While the interface to the main electronics and the core hardware remain identical, only the connected peripheral components differ depending on the functionality of the respective module.

For Gadget 1, the Hall encoder signals are acquired through two GPIO inputs of the ESP32. The motor driver is controlled via two digital output pins, while the DC motor is supplied separately through the 12 V power rail. Communication with the main electronics is established through the standardized CAN interface.

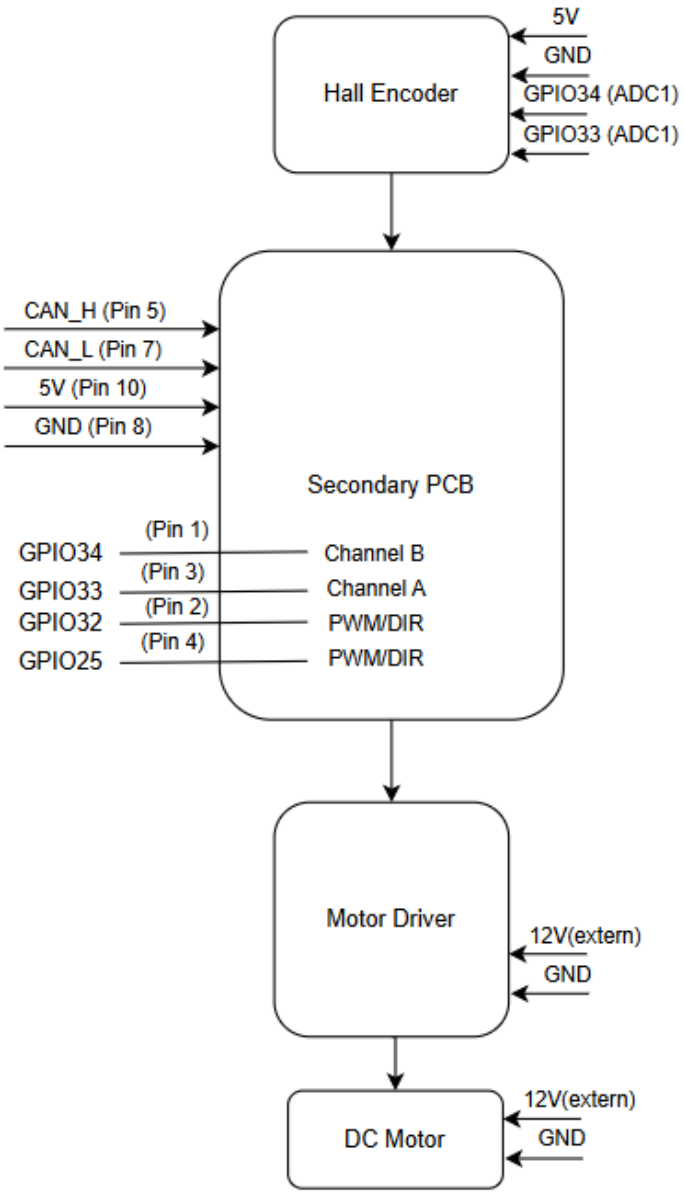


Figure 4.12: Electrical interconnection of the secondary PCB used in Gadget 1

For Gadget 2, the secondary PCB is used to control a servo motor. The PWM control signal is generated directly by the ESP32, while the servo is supplied from the 5 V power rail. In addition, one GPIO input is reserved for the connection of future sensor extensions. This allows additional functionality to be integrated without requiring modifications to the standardized hardware platform.

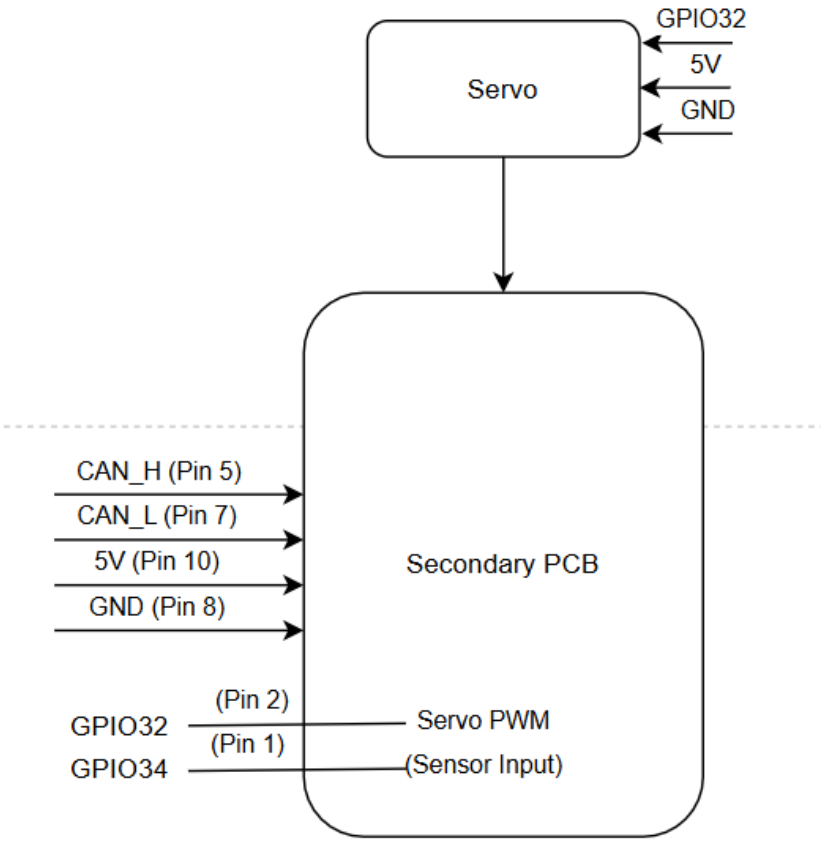


Figure 4.13: Electrical interconnection of the secondary PCB used in Gadget 2

The two figures demonstrate that different functional modules can be realized by adapting only the external peripheral connections of the standardized secondary PCB, while the underlying hardware platform remains unchanged.

4.7 Electrical Design of the Pogo-Pin Interface

The previously completed power supply design provides an estimate of the maximum current that must be transmitted through the pogo-pin interface. For the 5 V supply rail, a maximum current of 2 A was determined, while the 12 V supply rail is designed for a maximum current of 1 A. Consequently, the pogo pins should be rated to carry at least 2 A.

In practice, however, it cannot be assumed that every pogo pin provides an ideal electrical contact under all operating conditions. Mechanical tolerances, contamination, or contact wear can increase the contact resistance and reduce the maximum current that can be carried by individual contacts. To compensate for these effects, two design approaches are generally possible: using larger pogo pins with a higher current rating or connecting multiple pogo pins in parallel.

For the prototype presented in this work, the second approach was selected. Pogo pins with a higher current rating are generally larger and would have increased the space required by the mechanical design. By using multiple standard pogo pins in parallel, a compact design can be maintained while also providing redundant electrical contacts. If a single contact fails or exhibits an increased contact resistance, the electrical connection remains intact.

To account for potential contact losses, a safety factor of 2 was applied. In addition, one extra pogo pin was included so that the required safety factor is maintained even if one pogo pin fails. As a result, three pogo pins connected in parallel are used for each supply voltage.

Based on the previously defined interface, each pogo-pin strip must provide four electrical connections: one supply voltage (either 5 V or 12 V), ground (GND), and the two CAN communication lines, CAN_H and CAN_L. Consequently, each pogo-pin strip must contain at least four contacts.

Since three pogo-pin strips are used in parallel for each supply voltage, a total of six identical pogo-pin strips are required. Three strips carry the 5 V supply together with GND, CAN_H, and CAN_L, while the remaining three strips provide the 12 V supply together with the same ground and CAN communication lines. Using identical pogo-pin strips simplifies the mechanical design and improves the modularity of the interchangeable gadgets.

In addition to the number of contacts, the pin assignment was arranged with the mating sequence in mind. The supply voltage (5 V or 12 V) is placed at the uppermost contact, while ground (GND) is located at the lowermost contact, maximizing the separation between the two power rails. The CAN communication lines are positioned between the power contacts.

This arrangement reduces the risk of unintended electrical contact during the mating process. In particular, it minimizes the possibility of the supply voltage or ground briefly contacting the CAN lines, which could otherwise damage the connected electronics. The selected pin assignment therefore improves the robustness and operational reliability of the detachable interface.

Table 4.14: Pin assignment of a single pogo-pin connector

Contact	**Signal**	**Description**
1	VCC (5 V / 12 V)	Supply voltage
2	CAN_H	CAN bus high
3	CAN_L	CAN bus low
4	GND	Ground

4.8 Battery Sizing

This section addresses the battery sizing of the developed system. Since the scope of this work is limited to the development of a functional prototype, the battery is sized based on simplified assumptions. The objective is to establish a sizing methodology that can later be adapted using realistic usage data during future product development.

The system can generally be operated in three different states: Active, Standby, and Off. However, only the Active and Standby states are considered for battery sizing, since the Off state is associated only with the self-discharge of the lithium-ion battery. As the battery is dimensioned to achieve the required operating time between charging cycles, the effect of self-discharge is considered negligible over this period and is therefore neglected.

The required battery capacity strongly depends on the expected operating profile of the system. Therefore, a usage scenario must first be defined, describing the proportion of time spent in active operation and standby operation. For a production-ready system, this operating profile should be derived from long-term measurements or user studies to accurately represent real operating conditions.

For the purpose of this work, a simplified usage scenario consisting of 20% active operation and 80% standby operation is assumed. Although this assumption is intended only to demonstrate the sizing methodology, the calculation presented in the following section can be directly applied to future usage scenarios obtained from experimental measurements.

The battery sizing procedure is based on the average power consumption of the system. Since the system alternates between the Active and Standby operating states, the average power consumption can be expressed as the weighted average of both states:

$$P_{\text{avg}} = P_{\text{active}} \cdot D_{\text{active}} + P_{\text{standby}} \cdot D_{\text{standby}} \tag{4.12}$$

with

$$D_{\text{active}} + D_{\text{standby}} = 1. \tag{4.13}$$

Here, P_{active} and P_{standby} denote the power consumption during the corresponding operating states, while D_{active} and D_{standby} represent their respective fractions of the total operating cycle.

Once the average power consumption has been determined, the required energy for the desired operating time can be calculated as

$$E_{\text{bat}} = P_{\text{avg}} \cdot t \tag{4.14}$$

where (t) is the required operating time between two charging cycles.

Finally, the required battery capacity can be determined from the required battery energy and the nominal battery voltage:

$$C_{\text{req}} = \frac{E_{\text{bat}}}{U_{\text{nom}}} \tag{4.15}$$

where U_{nom} is the nominal voltage of the selected lithium-ion battery.

Before applying the battery sizing procedure, the power consumption values used for the active and standby states must be referred to the battery side. Since the measured power values were obtained at the output of the voltage converters, the converter efficiency has to be considered when estimating the power drawn from the battery.

Therefore, the battery-side power consumption is calculated as

$$P_{\text{bat}} = \frac{P_{\text{load}}}{\eta} \tag{4.16}$$

where P_{load} is the power consumed by the load, P_{bat} is the corresponding power drawn from the battery, and η is the assumed converter efficiency.

For the example calculation, the gripper module (Gadget 2) is used as the reference gadget. This gadget represents the 5 V load case, since it includes the servo motor as well as the remaining electronics supplied by the 5 V power rail. The measured power consumption values from the 5 V system are used as the basis for the calculation.

Since these values were measured at the output side of the voltage converter, the converter efficiency is considered before applying the battery sizing procedure. An efficiency of 85% is assumed for the DC-DC converter. The parameters used for the example calculation are summarized in Table 4.15.

Table 4.15: Parameters used for the battery sizing example

Parameter	Value
Active duty cycle	20%
Standby duty cycle	80%
Required operating time	8 h
Active power	6.5 W
Standby power	1.23 W
Converter efficiency	85%
Battery nominal voltage	3.7 V

Using the parameters listed in Table 4.15, the battery-side power consumption is first determined by considering the converter efficiency.

$$P_{\text{active}} = \frac{6.5,\text{W}}{0.85} = 7.65,\text{W} \tag{4.17}$$

$$P_{\text{standby}} = \frac{1.23,\text{W}}{0.85} = 1.45,\text{W} \tag{4.18}$$

The average power consumption is then calculated as

$$P_{\text{avg}} = 7.65,\text{W} \cdot 0.2 + 1.45,\text{W} \cdot 0.8 = 2.69,\text{W} \tag{4.19}$$

For the required operating time of 8 hours, the required battery energy is

$$E_{\text{bat}} = 2.69,\text{W} \cdot 8,\text{h} = 21.5,\text{Wh} \tag{4.20}$$

Finally, the required battery capacity is

$$C_{\text{req}} = \frac{21.5,\text{Wh}}{3.7,\text{V}} = 5.81,\text{Ah} \approx 5810,\text{mAh} \tag{4.21}$$

The example calculation serves as a guideline for the battery selection of the functional prototype. Since the calculated minimum required capacity is approximately 5810 mAh, a commercially available 7500 mAh lithium-ion battery was selected. The additional capacity provides a safety margin to compensate for variations in operating conditions, battery aging, and uncertainties in the assumed operating profile.

In addition to the battery itself, a charging circuit is required to enable convenient recharging of the system. For the prototype, a TP4056 lithium-ion charging module was integrated into the power supply. The module implements the charging process for lithium-ion battery and provides a simple solution for charging the battery via a USB connection. This allows the battery to remain installed in the system while being recharged, improving the usability of the prototype.

5 Mechanical Coupling System

5.1 Mechanical Design Requirements

The mechanical coupling interface represents the central connection between the main unit and the interchangeable prosthetic gadgets. Besides providing a reliable mechanical attachment, the interface must also enable electrical power transmission and communication between both subsystems. Consequently, the mechanical design is not limited to structural considerations but also has to support the electrical functionality of the developed platform.

Based on the system objectives presented in Chapter 3, a set of mechanical design requirements was derived to guide the development of the coupling interface. These requirements address the interchangeability of gadgets, the reliability of the mechanical connection, the integration of the electrical interface, and the overall modularity of the platform. Furthermore, practical aspects such as the compact integration into the prosthesis, compatibility with additive manufacturing, and the compensation of manufacturing tolerances were considered during the design process.

The resulting mechanical design requirements are summarized in Table 5.1. The following sections describe how these requirements influenced the concept development and the final implementation of the coupling interface.

Table 5.1: Mechanical design requirements for the coupling interface

ID	Requirement	Rationale
M1	The mechanical interface shall allow interchangeable prosthetic gadgets to be attached and removed without the use of tools.	Enables quick exchange of task-specific gadgets during daily use.
M2	The interface shall provide repeatable positioning of the gadget with respect to the main unit.	Ensures reliable mechanical alignment and consistent electrical contact.
M3	The interface shall securely retain the attached gadget during normal operation.	Prevents accidental detachment while transmitting operational forces.
M4	The mechanical interface shall integrate the electrical connection between the main electronics and the interchangeable gadgets.	Allows automatic establishment of power supply and communication during attachment.
M5	The interface shall occupy as little installation space as possible.	Minimizes the overall dimensions of the prosthetic system.
M6	The mechanical interface shall be identical for all developed and future gadgets.	Enables a standardized and expandable platform architecture.
M7	The interface shall be designed for fabrication using additive manufacturing.	Allows rapid prototyping using FDM 3D printing.
M8	The interface shall tolerate manufacturing and assembly tolerances while maintaining reliable mechanical and electrical contact.	Compensates for dimensional deviations introduced by additive manufacturing and repeated assembly.
M9	The interface shall guide the gadget into its final position during attachment.	Simplifies handling and improves usability by providing self-alignment during insertion.

5.2 Concept Development

Based on the mechanical requirements presented in the previous section, the coupling interface was divided into tree independent functional elements. Separating these functions simplifies the design process and allows each requirement to be addressed individually.

5.2.1 Guiding Function

The first function of the coupling interface is to guide the interchangeable gadget into its correct position during attachment. Since the electrical connection between the main unit and the gadget is established automatically when both components are connected, accurate alignment of the mating interfaces is essential. Improper positioning could lead to unreliable electrical contacts or increased mechanical wear during repeated insertion.

In addition to ensuring reliable electrical contact, the guiding mechanism simplifies the handling of the prosthesis by reducing the positioning accuracy required from the user. Instead of requiring precise manual alignment, the interface should automatically guide the gadget into its final position during insertion. This improves usability while simultaneously fulfilling the requirements for repeatable positioning (M2) and self-alignment (M9).

Consequently, the guiding function serves as the basis for all subsequent functions of the coupling interface. Only after the gadget has been accurately positioned can a reliable locking mechanism and electrical connection be established.

5.2.2 Locking Function

The second function of the coupling interface is to securely retain the gadget after it has been guided into its final position. Once the correct alignment has been achieved, the connection must prevent unintended separation during normal operation while still allowing the gadget to be removed intentionally by the user without the need for additional tools.

The locking mechanism therefore has to provide sufficient holding force to withstand the loads generated during daily operation while maintaining the repeatable positioning established by the guiding function. At the same time, the required separation force should remain low enough to ensure a convenient and intuitive exchange of gadgets.

Since the locking mechanism acts on an already aligned interface, it does not contribute to the positioning process itself. Instead, its primary purpose is to preserve the established alignment throughout operation and to fulfill the requirements for secure mechanical retention (M3) and tool-free interchangeability (M1).

5.2.3 Standardized Interface Geometry

The coupling interface was designed with a standardized geometry that is shared by all interchangeable gadgets. This ensures mechanical compatibility across the entire platform and allows future gadgets to be integrated without modifications to the main unit.

In addition to standardization, the interface geometry was designed to prevent incorrect attachment. The guiding and locking features allow the gadget to be inserted only in its intended orientation, reducing the risk of improper assembly while protecting both the mechanical and electrical interfaces. This concept fulfills the requirement for a standardized interface (M6) and provides a robust foundation for future platform extensions.

5.3 Mechanical Design

5.3.1 Guiding Geometry

The guiding geometry implements the functional concept introduced in the previous section by combining alignment and orientation within a single mechanical interface. Its primary objective is to guide the connector into a unique and repeatable position while preventing incorrect attachment.

As shown in Figure 5.1, the connector consists of three planar guiding surfaces that engage with the corresponding surfaces of the socket during insertion. These surfaces guide the connector into its final position while simultaneously defining its rotational orientation. Consequently, the gadget can only be inserted in its intended orientation. In addition, the planar surfaces provide flat mounting areas for the pogo-pin connectors, ensuring reliable electrical contact after assembly.

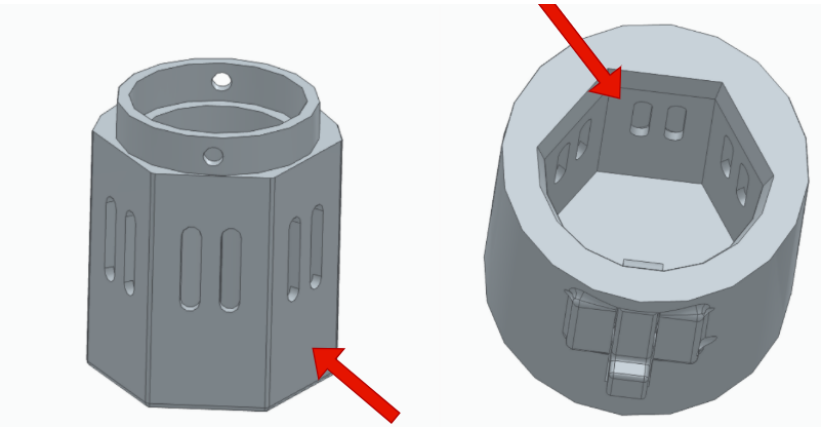


Figure 5.1: Guiding geometry of the connector (left) and socket (right)

To further increase the stability of the assembled interface, an additional cylindrical guiding element was integrated opposite the planar guiding surfaces, as illustrated in Figure 5.2. Since the orientation of the connector is already fully defined by the guiding geometry, the cylindrical guide primarily serves as an additional support point. This reduces relative movement between the connected components and improves the stiffness of the interface without affecting the insertion process.

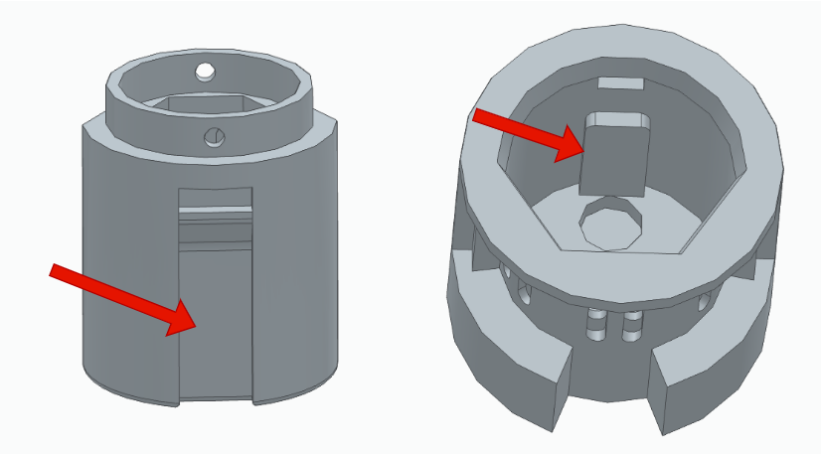


Figure 5.2: Additional cylindrical guide of the connector (left) and the corresponding socket (right)

5.3.2 Locking Mechanism

The locking mechanism was designed to provide a secure mechanical connection while enabling a simple one-handed exchange of interchangeable gadgets. To achieve this objective, a spring-loaded locking latch was combined with an integrated ejection mechanism.

During insertion of the connector, the lower spring-loaded ejector is compressed while the connector simultaneously pushes the locking latch into its open position. Once the connector reaches its final position, the latch automatically returns to its initial position under spring force and engages with the connector. This creates a secure mechanical connection without requiring any additional user interaction.

To remove a gadget, the user presses the release button located on the outside of the socket. This disengages the locking latch and releases the connector. At the same time, the energy stored in the compressed ejector spring pushes the connector slightly out of the socket, allowing the gadget to be removed easily using only one hand.

Since the prototype is manufactured using FDM 3D printing, an additional locking hole was incorporated into the mechanism. A removable safety pin can be inserted through this hole to prevent unintended deformation of the printed locking latch under high mechanical loads. This feature further increases the reliability of the connection while preserving the quick-release functionality when the safety pin is not used.

The developed locking mechanism is illustrated in Figure 5.3.

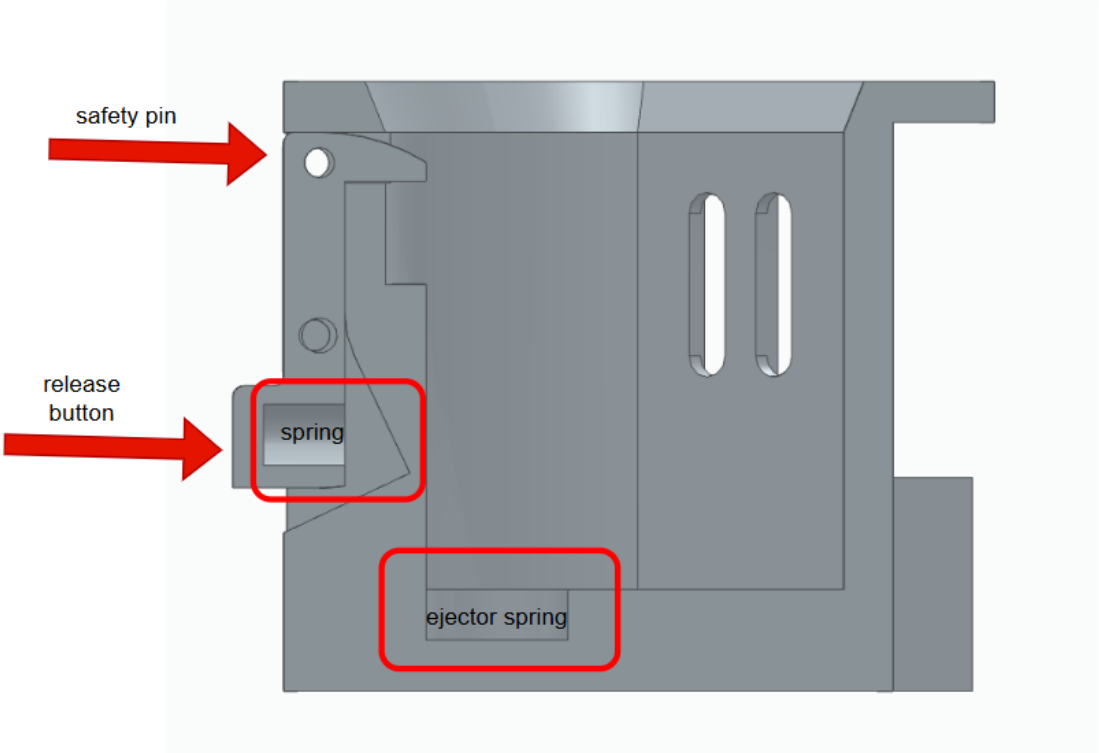


Figure 5.3: Cross-sectional view of the spring-loaded locking mechanism with integrated ejector and safety pin lock

5.3.3 Integration of the Electrical Interface

The electrical interface presented in Chapter 4 was integrated directly into the mechanical coupling system. The pogo-pin connectors were mounted on the planar guiding surfaces of the socket, allowing the electrical connection to be established automatically once the connector reaches its final position. The position of the connectors follows the standardized interface layout introduced in the electrical design, ensuring compatibility with all interchangeable gadgets.

To accommodate the electrical wiring, dedicated cable routing channels were incorporated into the socket geometry, as shown in Figure 5.4. These channels provide a defined routing path for the wires connecting the pogo-pin connectors to the main electronics. After assembly, the wiring is completely enclosed by the socket housing and its cover, protecting the electrical connections while maintaining a compact and organized internal layout.

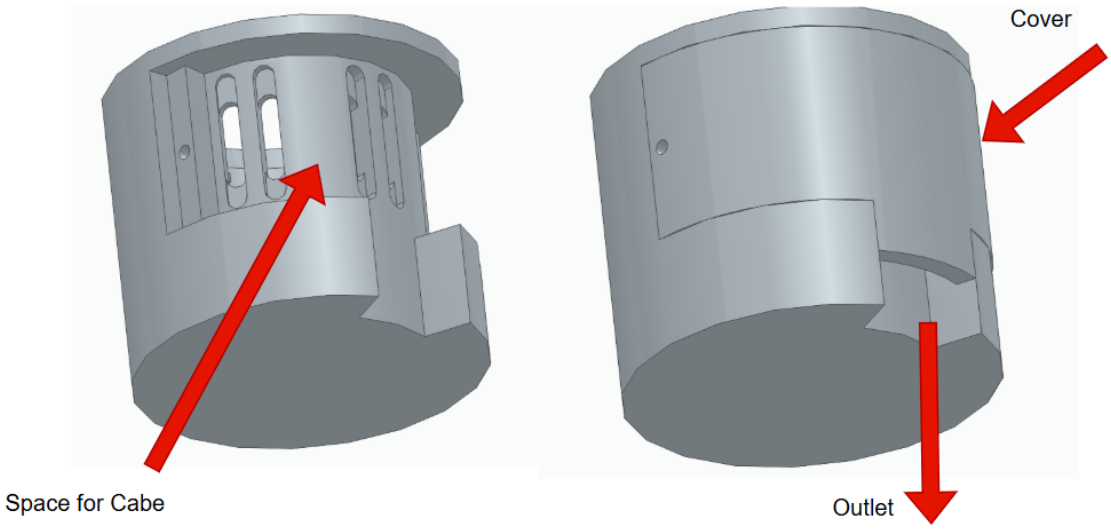


Figure 5.4: Integrated cable routing within the socket housing (left: cable routing channels, right: assembled housing with cover)

5.4 Mechanical Tolerance Evaluation

Due to the manufacturing tolerances inherent to FDM 3D printing, the nominal CAD dimensions could not be directly adopted for the final prototype. Therefore, several test specimens were manufactured to experimentally determine suitable dimensions for the coupling interface. The objective was to ensure reliable assembly while maintaining intuitive handling and accurate positioning of the mechanical and electrical components.

The first evaluation focused on the clearance between the connector and the socket. Three connector variants with diameters of 48 mm, 49 mm, and 50 mm were manufactured and assembled to investigate the influence of the connector diameter on the insertion behavior and the mechanical stability of the coupling interface. The investigated prototypes are shown in Figure 5.5.

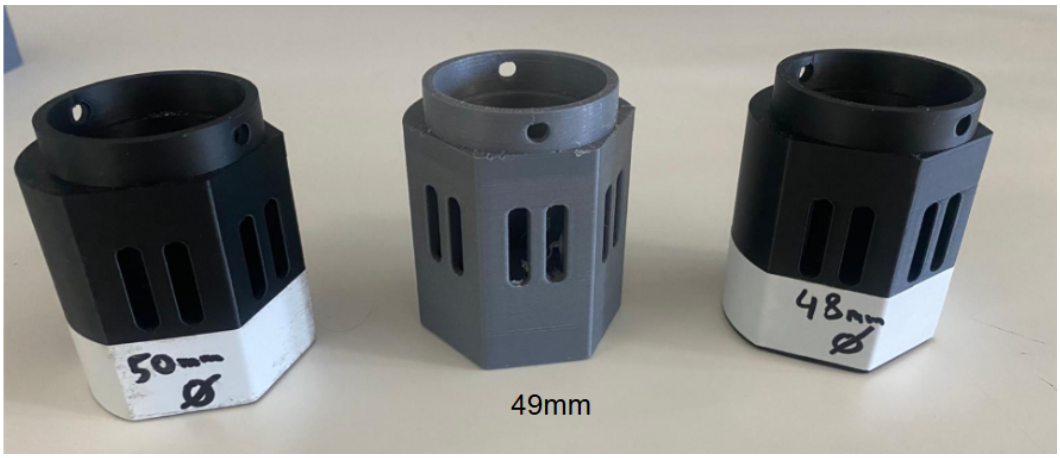


Figure 5.5: Connector prototypes with diameters of 48 mm, 49 mm, and 50 mm used for the tolerance evaluation.

The experimental evaluation showed that the 49 mm connector provided the best compromise between insertion force and mechanical stability. While the 48 mm variant exhibited noticeable mechanical play after assembly, the 50 mm variant required a significantly higher insertion force. Consequently, a connector diameter of 49 mm was selected for the final prototype.

The second evaluation investigated the mounting dimensions of the pogo-pin connectors. According to the manufacturer's datasheet, the connectors have a nominal width of 4.0 mm. Due to the dimensional deviations introduced by the FDM printing process, several slot dimensions were evaluated experimentally. The objective was to identify dimensions that allow the connectors to be inserted easily while minimizing positional play during assembly. A slot width of 4.2 mm combined with a slot length of 18 mm provided the best compromise between assembly and positioning accuracy. After insertion, the pogo-pin connectors were permanently fixed using adhesive. These dimensions were therefore selected for the final design.

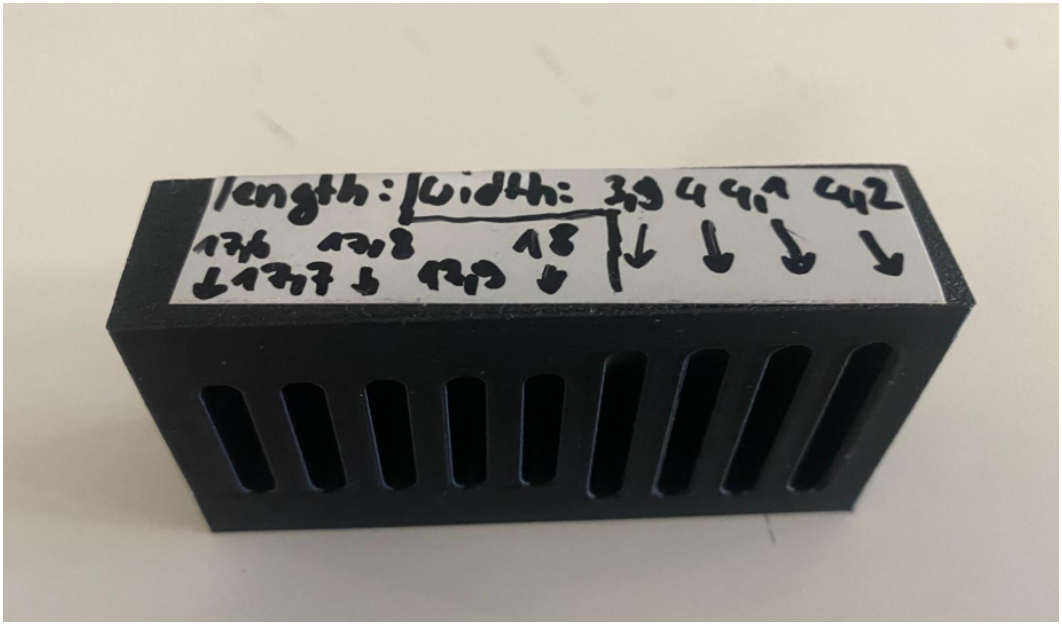


Figure 5.6: Tolerance evaluation of the pogo-pin mounting geometry.

The experimentally determined dimensions were incorporated into the final CAD model and used for the fabrication of all prototype components. The evaluation demonstrated that minor adjustments to the nominal CAD dimensions were required to compensate for the manufacturing tolerances introduced by the additive manufacturing process.

5.5 Final Coupling Assembly

The individual design elements presented in the previous sections were combined to form the final mechanical coupling system. The guiding geometry, spring-loaded locking mechanism, integrated electrical interface, and cable routing were assembled into a compact interface that fulfills the mechanical and electrical requirements defined at the beginning of this chapter.

The resulting assembly provides a standardized connection between the main unit and the interchangeable gadgets while enabling reliable positioning, secure retention, and automatic establishment of the electrical connection. The final prototype of the developed coupling system is shown in Figure 5.7.

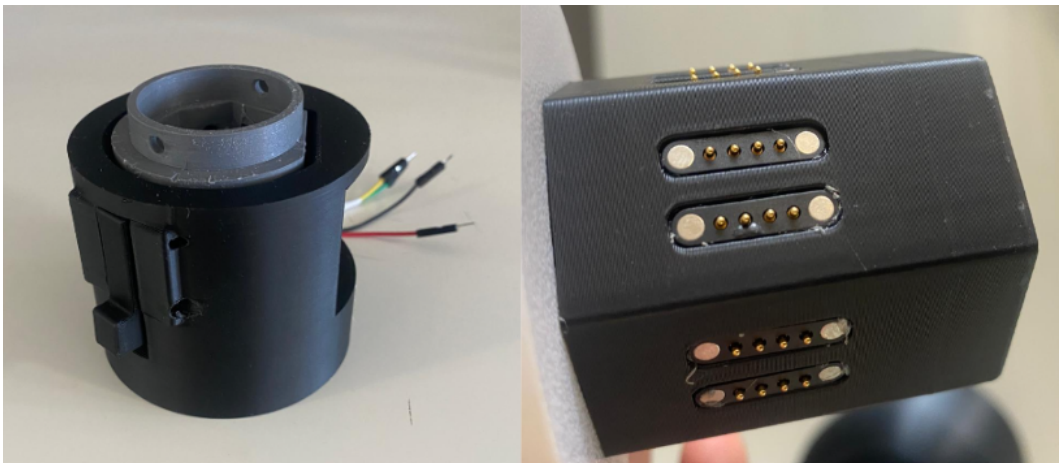


Figure 5.7: Final prototype of the developed coupling system showing the socket (left) and the connector with the integrated pogo-pin interface (right)

6 Main Unit Integration

The main unit housing was developed to accommodate the electronic components of the system. During the design process, two different integration concepts were investigated, as shown in Figure 6.1.

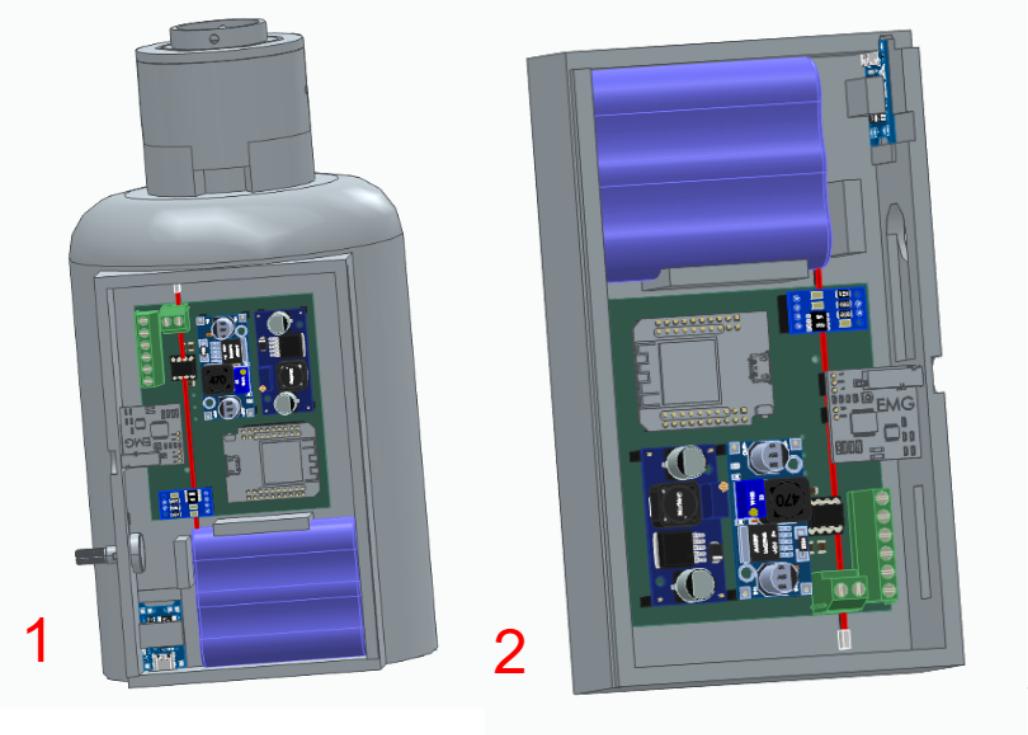


Figure 6.1: Comparison of the investigated integration concepts: (1) integrated main electronics inside the prosthetic glove and (2) separate electronics housing

The first concept integrated both the coupling system and the main electronics into the prosthetic glove. While this resulted in a compact design, the dimensions of the current prototype electronics increased the size of the glove and restricted the available wrist rotation.

Therefore, a second concept based on a separate electronics housing was selected for the functional prototype. By relocating the main electronics outside the glove, the size of the prosthetic hand was reduced while preserving wrist mobility.

The integrated concept could be reconsidered in future development if the current modular electronics are replaced by a more compact PCB design. Integrating the microcontroller, voltage con-

verters, and other electronic components directly onto a single PCB would reduce the required installation space and could enable the main electronics to be integrated into the glove without significantly restricting wrist rotation.

6.1 Mechanical Integration

The selected housing was designed to accommodate the electronic components of the main unit within a compact enclosure. The main PCB is mounted on integrated screw bosses, while the lithium-ion battery is positioned in a dedicated compartment and retained by slightly tapered side walls. In addition, a dedicated mounting space for the TP4056 charging module was incorporated into the housing.

The housing also provides the required openings for the potentiometer, USB-C charging interface, and cable outlet. The final housing design together with the integrated electronic components is shown in Figure 6.2.

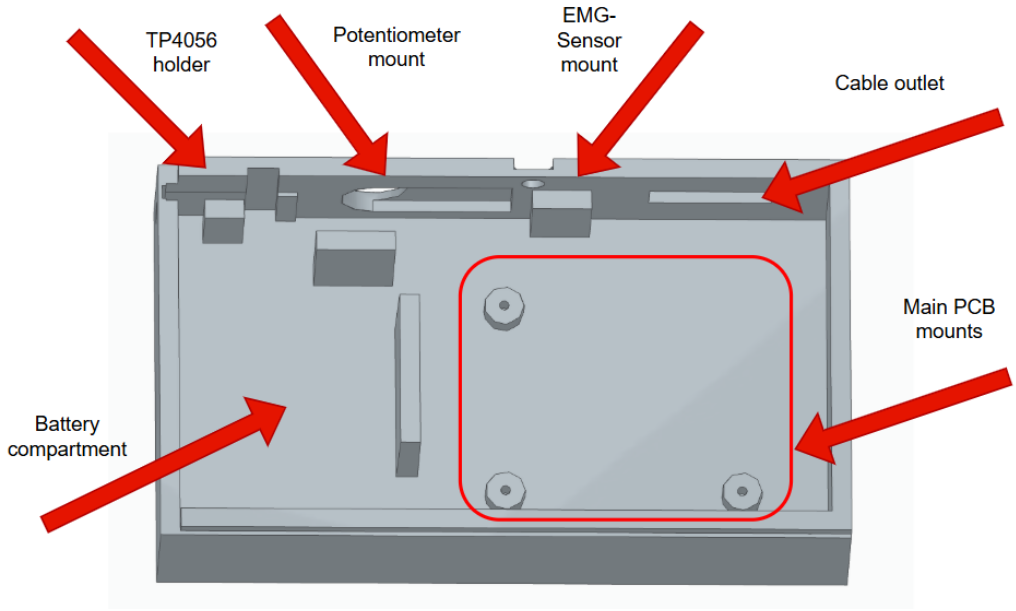


Figure 6.2: Integrated mounting features and external interfaces of the main unit housing

6.2 Charging Module Integration

Due to its dedicated mounting concept, the integration of the TP4056 charging module is described separately.

A dedicated holder was incorporated into the housing to accommodate the TP4056 charging module, as shown in Figure 6.3. The module is inserted into the holder and retained without additional

fasteners. The holder further provides direct access to the USB-C charging connector and includes recesses that provide sufficient clearance for the soldered wire connections. This allows the charging module to be positioned accurately while remaining easily accessible.

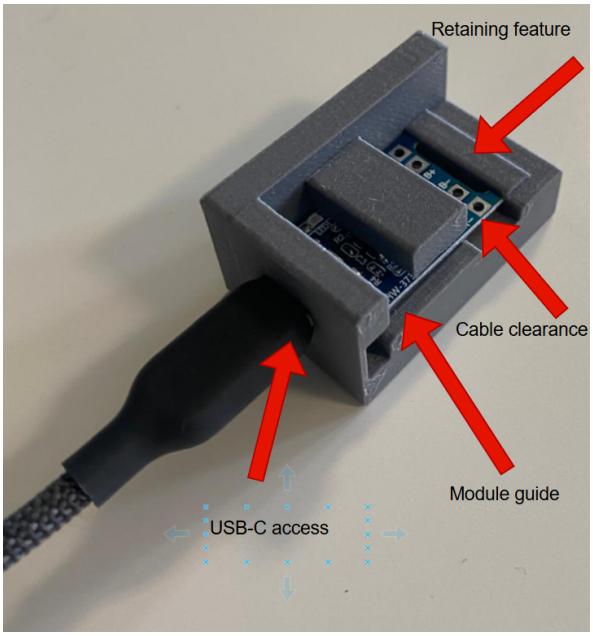


Figure 6.3: Integrated holder for the TP4056 charging module

7 Development of the Modular Gadgets

To demonstrate the functionality and versatility of the proposed modular platform, two interchangeable mechatronic gadgets were developed: a gripper module and a rotation module. Both gadgets are connected to the main unit through the standardized mechanical and electrical interface presented in the previous chapters.

Each gadget incorporates its own ESP32 microcontroller, CAN transceiver, sensors, and actuators according to its intended functionality. While the main unit provides the common electronic infrastructure and user inputs, the gadget electronics execute the application-specific tasks required for the respective module.

7.1 Module 2 Gripper

This chapter presents the development of the gripper module. The focus is placed on the definition of the mechanical requirements, the evaluation of different gripping mechanisms, and the selection of the final concept.

The primary requirement was to develop an actively driven gripper capable of grasping objects of different sizes and geometries. Furthermore, the design should provide the possibility of integrating a pressure sensor for monitoring the gripping process.

Since the gripper is intended as an interchangeable mechatronic module, the mechanical design also had to remain as compact as possible while providing sufficient installation space for the required electronic components

7.1.1 Concept Evaluation

Based on the defined design requirements, three different gripping mechanisms were investigated. The most suitable concept was subsequently selected for the final prototype.

The first gripping mechanism is based on a planar four-bar linkage that converts the rotational motion of the actuator into a synchronous gripping motion of the fingers. Due to the specific geo-

metric configuration of the mechanism, the rotational movement of the motor is transformed into a parallel motion of the two gripper fingers. This enables a uniform and stable gripping process[9, Sec. 10.2.3].

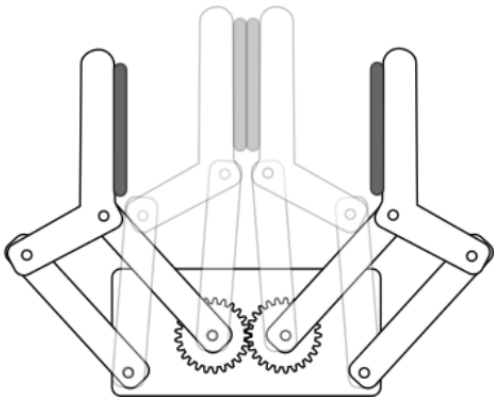


Figure 7.1: Four-bar-Linkage mechanism (adapted from [9, Sec. 10.2.3])

The second gripping mechanism is based on a lead screw drive. The motor is centrally positioned in the lower section of the gripper and actuates a platform along the vertical axis by means of a threaded spindle. The translational movement of the platform is transferred to the gripper fingers, thereby enabling the opening and closing motion of the gripper[18, Sec. 3.3].

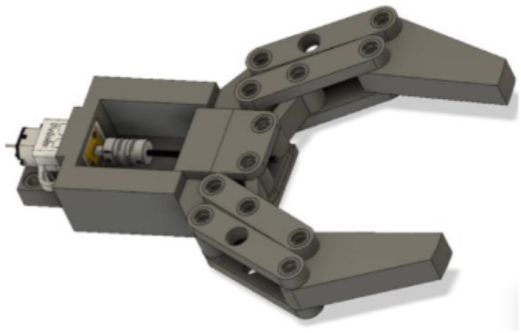


Figure 7.2: lead screw drive mechanism (adapted from [18, Sec. 3.3])

The third gripping mechanism is based on a rack-and-jaw drive. Similar to the second concept, the motor is located in the lower central section of the gripper. The rotational motion of the motor is transmitted via a jaw gear to two racks that move linearly in opposite directions 7.3. The gripper fingers are directly attached to the racks, allowing their linear displacement to be converted directly into the opening and closing motion of the gripper. [9, Sec. 10.2.3].

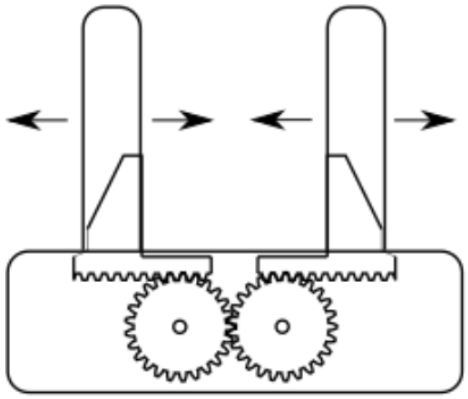


Figure 7.3: rack-and-jaw drive mechanism (adapted from [9, Sec. 10.2.3])

All investigated gripping mechanisms are generally capable of fulfilling the basic requirements of the gripper module. However, within the scope of this work, the four-bar linkage mechanism was considered the most suitable solution with respect to the available installation space.

The proposed modular platform provides only a limited cylindrical installation space inside the gripper module. The planar geometry of the four-bar linkage allows the actuator to be integrated alongside the gripping mechanism, leaving sufficient space for the printed circuit board and the remaining electronic components. This arrangement also simplifies the routing of the wires from the pogo-pin connector to the electronics.

For these reasons, the four-bar linkage mechanism was selected for the final prototype.

Since one of the primary requirements of the gripper is the reliable grasping of objects with different shapes and sizes, the design of the gripper fingers plays a significant role. Conventional rigid fingers provide a limited contact area, which strongly depends on the geometry of the grasped object and may reduce the stability of the grasp.

For this reason, the use of flexible gripper fingers was investigated. Their ability to adapt to the object's surface increases the contact area between the gripper and the object, which can improve gripping stability and reliability when handling objects with varying geometries[23].

7.1.2 Force Analysis

Since this type of gripper is uncommon in prosthetic applications, only limited literature is available regarding the required gripping force. Therefore, the gripping forces of existing anthropomorphic prosthetic hands were used as a reference.

Belter and Dollar compared a variety of anthropomorphic prosthetic hands with respect to the number of actuators and the achievable grip force. The investigated systems covered grip forces ranging from 10.8 N to 60 N using between one and sixteen actuators [4].

Since the proposed four-bar linkage gripper is actuated by a single motor, only prosthetic hands with one actuator were considered relevant for the present design. In the referenced study, the TBM Hand and the MANUS Hand achieve grip forces of approximately 15 N and 60 N, respectively. Consequently, the target gripping force of the proposed gripper was selected within this range [4].

Based on this target range, the required gripping force is derived in the following section to determine a suitable actuator for the gripper.

Figure 7.4 shows the free-body diagram of the four-bar linkage gripper used for the force analysis.

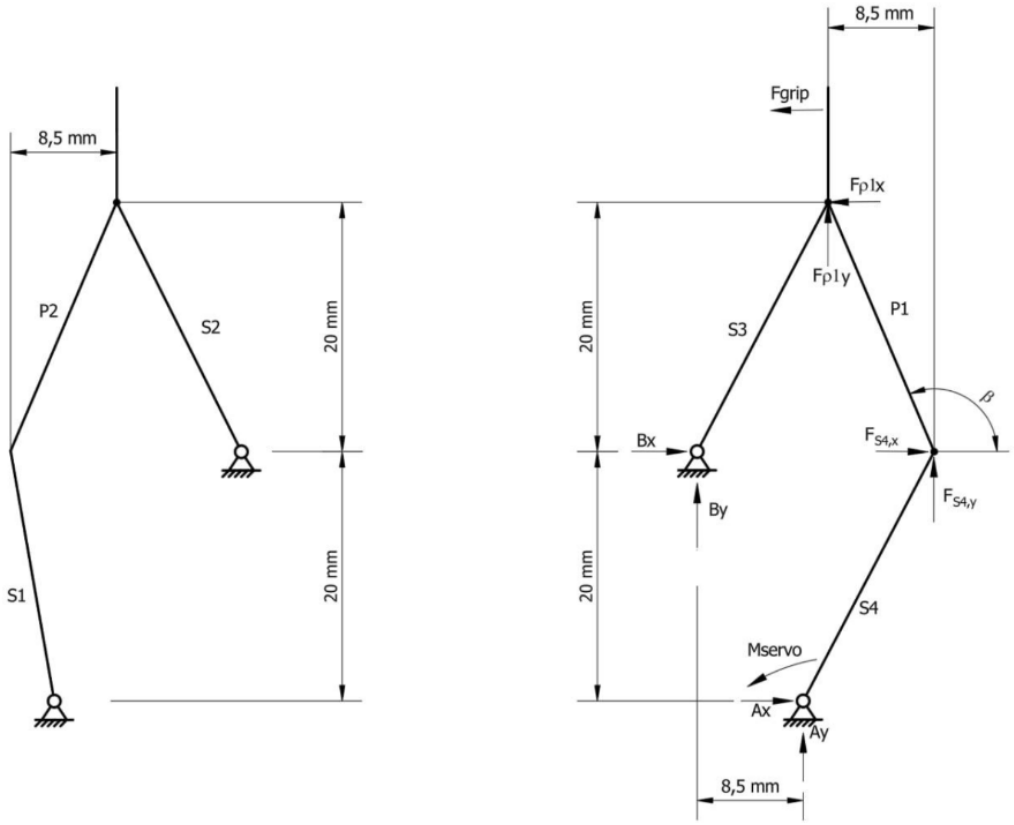


Figure 7.4: Free-body diagram of the four-bar linkage gripper used for the force analysis

Since the actuator is directly connected to only one side of the gripper and the motion is transmitted to the opposite finger via a gear mechanism, the force analysis can be limited to a single side of the gripper. Based on the free-body diagram shown in Figure 7.4, the equilibrium equations can be derived.

S4:

$$\sum F_x = 0 = A_x + F_{S4,x} \Rightarrow F_{S4,x} = F_{S4}\cos(\beta) \tag{7.1}$$

$$\sum F_y = 0 = A_y + F_{S4,y}7 \Rightarrow F_{S4,y} = F_{S4}\sin(\beta) \tag{7.2}$$

$$\sum M_A = 0 = M_{\text{servo}} - F_{S4,x}\sin(\alpha)\,S_4 + F_{S4,y}\cos(\alpha)\,S_4 \tag{7.3}$$

P1:

$$\sum F_x = 0 = F_{S4,x} - F_{P1,x} \tag{7.4}$$

$$\sum F_y = 0 = F_{S4,y} + F_{P1,y} \tag{7.5}$$

$$F_{P1,x} = F_{Grip} \tag{7.6}$$

(7.7)

Using these equilibrium equations, the expression for the gripping force can be derived as a function of the actuator torque and the gripping angle α. The angle β is treated as a constant, as it is defined by the geometry of the mechanism and remains unchanged throughout the entire gripping motion.

$$|M_{\text{servo}}| = \frac{F_{\text{grip}}}{\cos(\beta)}\,S_4\sin(|\alpha - \beta|) \tag{7.8}$$

To estimate the maximum payload of the gripper, an additional equation is derived that relates the gripping force to the weight of the grasped object.

$$F_{\text{grip}} = \frac{m \cdot g}{\mu}\,\upsilon \tag{7.9}$$

A safety factor υ of 1.5 and a coefficient of static friction μ of 1 are assumed for the calculation.

Using the derived equations, the required actuator torque can be calculated as a function of the gripping angle α. Since the gripper is intended to reliably grasp objects of different sizes, the required actuator torque must be evaluated over the complete range of the gripping angle α.

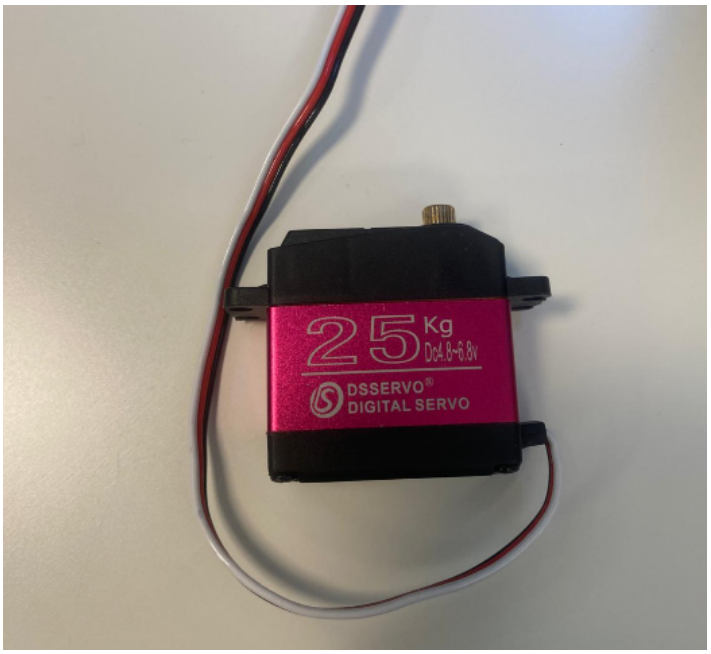


Figure 7.5: Selected Servo

The most powerful servo motor available in the laboratory for this work was a 25 kg servo. According to the manufacturer's specification, this corresponds to a stall torque of approximately 2.5 Nm (25 kg·cm).

Based on the derived equations, it can now be evaluated whether this actuator provides sufficient torque for the proposed gripper design.

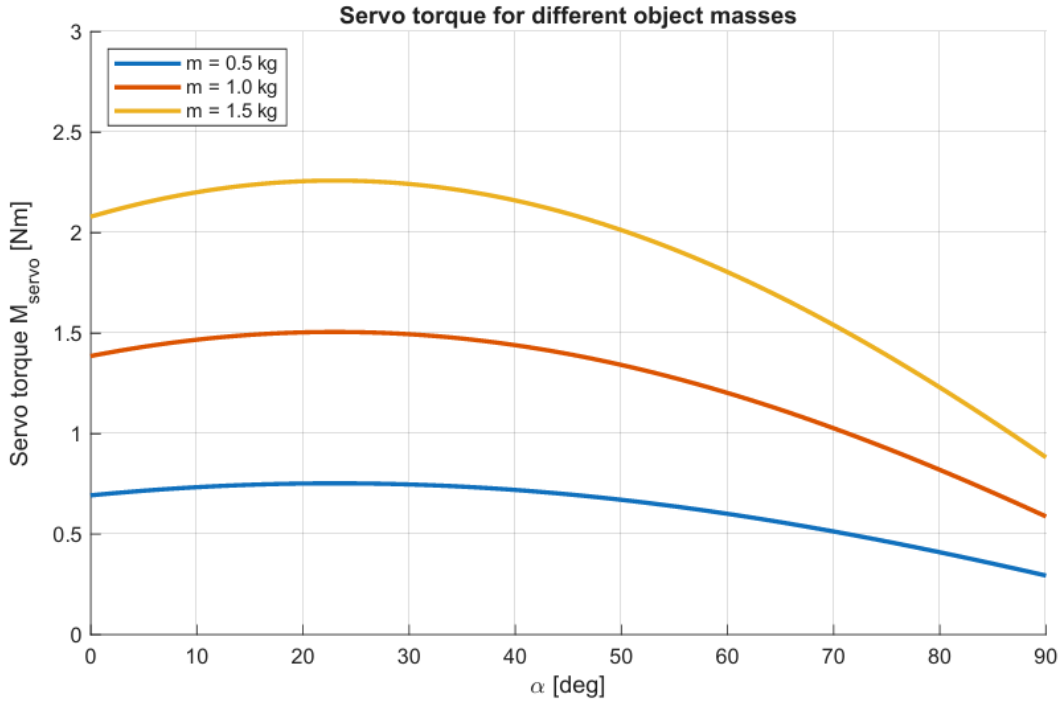


Figure 7.6: Servo torque for different object masses

Figure 7.6 illustrates the relationship between the required servo torque and the weight of the grasped object. Based on the analytical model, the selected servo is capable of generating a gripping force of approximately 20 N, corresponding to a maximum object weight of about 2 kg. Since this gripping force lies within the target range of 15–60 N identified from existing anthropomorphic prosthetic hands [4], the selected servo is considered suitable for the proposed gripper design.

7.1.3 Flexible Finger Design

The flexible gripper fingers were designed based on the Fin-Ray principle, which allows the fingers to passively adapt to the geometry of the grasped object. The geometry was developed specifically for the proposed gripper while following the general design principles reported in the literature [23].

To realize this behavior, the fingers were manufactured from thermoplastic polyurethane (TPU). The elasticity of the material enables the fingers to deform during grasping, allowing them to conform to the surface of the object and increase the contact area. This passive adaptation improves the stability of the grasp without requiring additional mechanical components. The manufactured TPU fingers are shown in Figure 7.7.

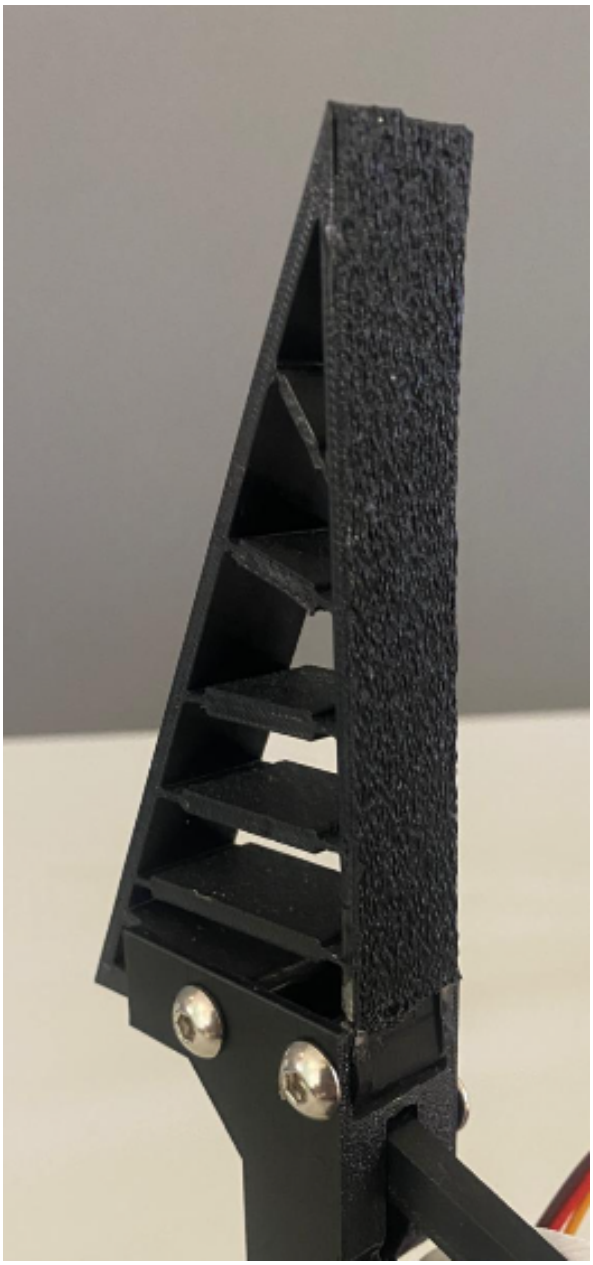


Figure 7.7: Flexible TPU finger based on the Fin-Ray principle.

7.1.4 Final Gripper Design

Based on the concept evaluation and the force analysis presented in the previous sections, the final gripper prototype was designed and manufactured. The design combines the selected four-bar linkage mechanism with flexible TPU fingers and the standardized coupling interface developed in this work. The resulting prototype demonstrates the mechanical implementation of the proposed gripper concept and its integration into the modular gadget platform.

The final CAD model of the gripper is shown in Figure 7.8. The housing integrates the complete gripping mechanism, the servo motor, the module electronics, and the standardized coupling in-

terface within a compact assembly.

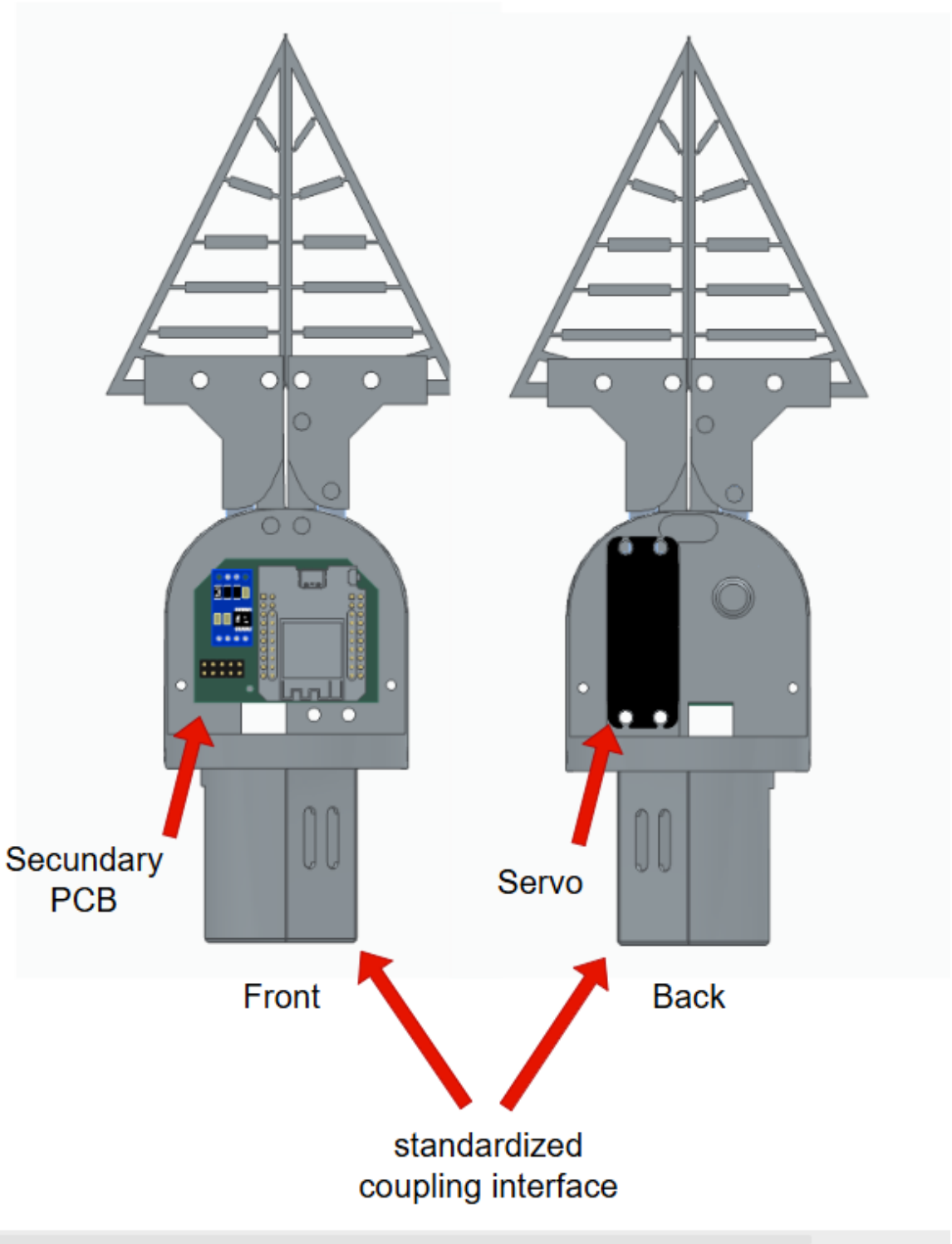


Figure 7.8: Servo torque for different object masses

As shown in Figure 7.9, the gripper is actuated by a single servo motor mounted directly to the driving gear. A second gear with a transmission ratio of 1:1 synchronizes the motion of the opposite finger, ensuring a symmetrical opening and closing movement. The rotational motion is converted into the gripping motion by the selected four-bar linkage mechanism.

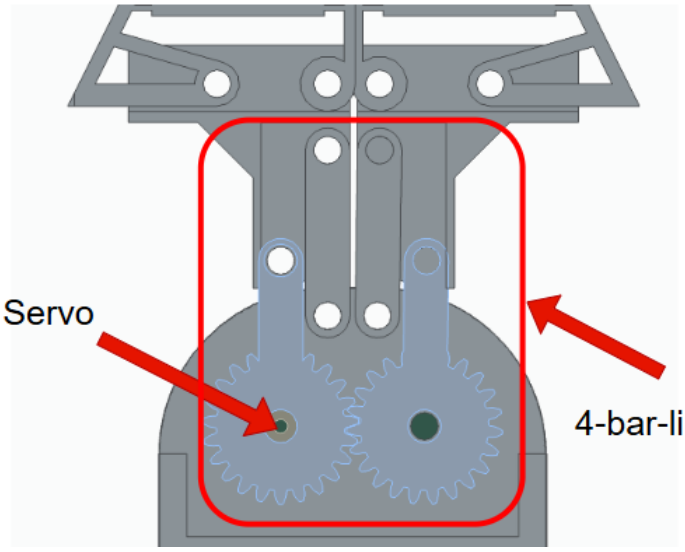


Figure 7.9: Integration of the servo motor and four-bar linkage mechanism inside the gripper housing

A dedicated compartment is provided for the module electronics, while the mechanical drive is enclosed separately within the housing, as illustrated in Figure 7.8. This arrangement enables a compact integration of the mechanical and electronic components while maintaining the standardized coupling interface at the bottom of the module.

The final gripper was manufactured using fused deposition modeling (FDM) based on the CAD model presented in the previous section. The rigid structural components were printed from PLA, while the flexible gripper fingers were fabricated from TPU to provide the desired compliance. Figure 7.10 shows the assembled prototype.

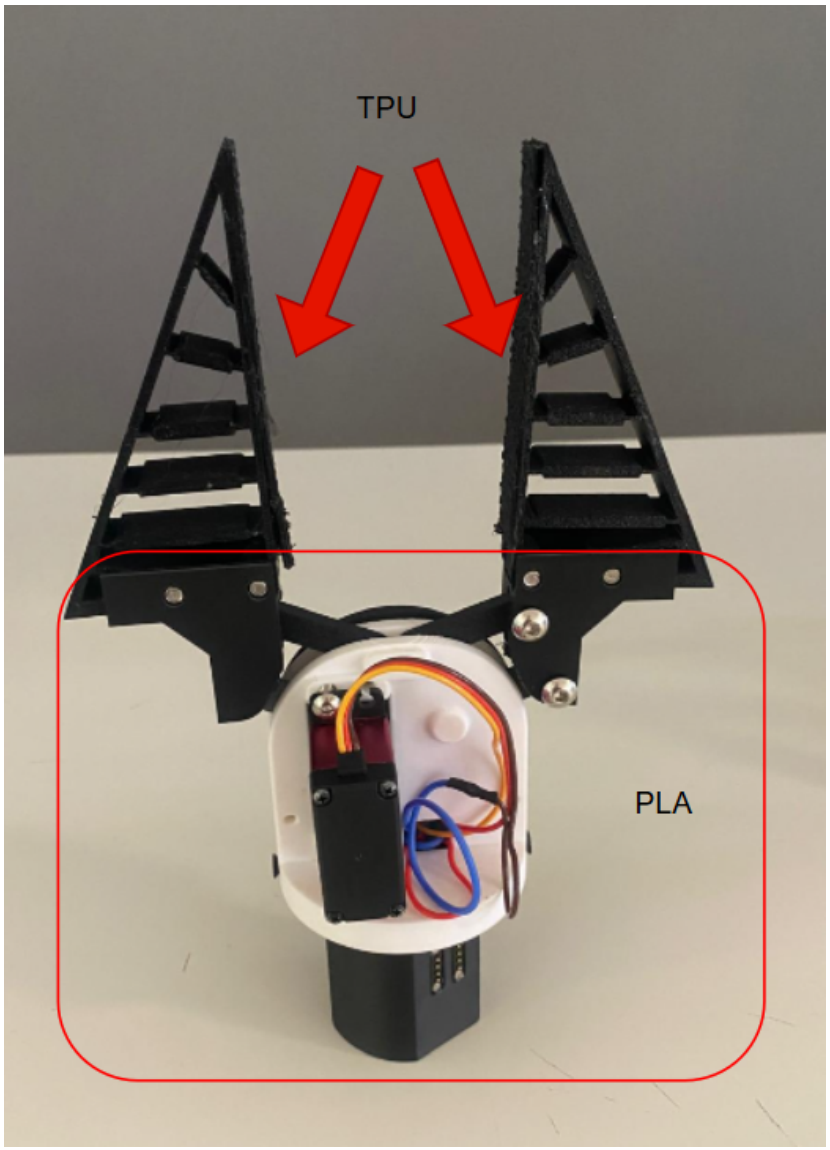


Figure 7.10: Manufactured gripper prototype showing the rigid PLA housing and linkage components together with the flexible TPU gripper fingers

7.2 Module 1 Rotator

7.2.1 Requirements

The second module developed within this work is a rotation module. In contrast to the gripper, its primary purpose is not a dedicated application but to demonstrate that the proposed modular platform is capable of supporting interchangeable modules with different actuation principles. The module is required to provide controlled bidirectional rotational motion while maintaining compatibility with the standardized coupling interface and the secondary PCB developed in this work. Furthermore, the mechanical design should remain compact to accommodate both the actuator and the module electronics within the available installation space. To fulfill these requirements, a DC gear motor with an integrated Hall encoder was selected. The encoder enables rotational feedback for future closed-loop control, while the geared motor provides sufficient torque for demonstrating the functionality of the modular platform.

7.2.2 Mechanical Design

The final CAD model of the rotation module is shown in Figure 7.11. The module consists of a two-part housing containing the actuator, the secondary PCB, and the standardized coupling interface. The overall design follows the same modular architecture as the gripper module while providing a different mechanical functionality.

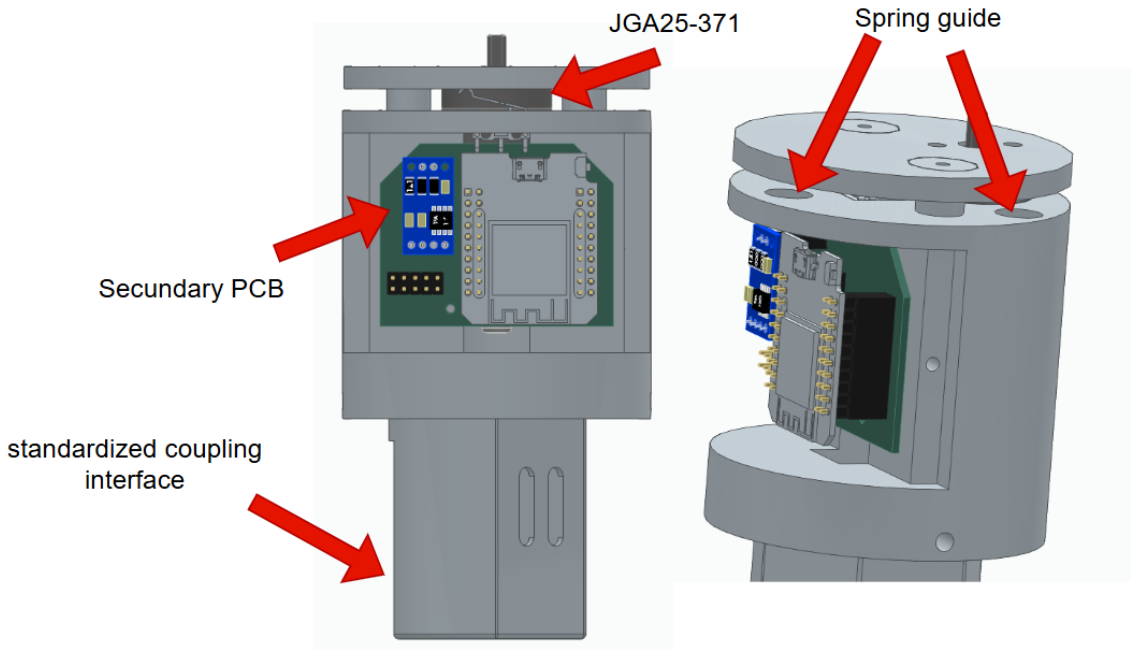


Figure 7.11: Final CAD model of the rotation module showing the integration of the JGA25-371 DC gear motor, the secondary PCB, the spring mounts, and the standardized coupling interface

A JGA25-371 DC gear motor is mounted inside the housing and directly connected to the output shaft located at the top of the module. The motor is attached to the spring-loaded actuator plate, allowing both components to move vertically together when an external force is applied. The secondary PCB is installed in a dedicated compartment within the housing. Similar to the gripper module, the standardized coupling interface is integrated into the bottom section of the housing, allowing the module to be connected to the modular platform.

The spring-loaded actuator plate was designed to accommodate the integration of a limit switch. This concept was intended for future extensions requiring interaction with the environment by detecting an externally applied force before activating the rotational motion. However, this functionality was not implemented within the scope of this work.

7.2.3 Final Prototype

The rotation module was manufactured using FDM based on the CAD model presented in the previous section. All structural components were printed from PLA. The assembled prototype is shown in Figure 7.12. The prototype incorporates the mechanical housing, the spring-loaded actuator plate, the secondary PCB, and the standardized coupling interface.

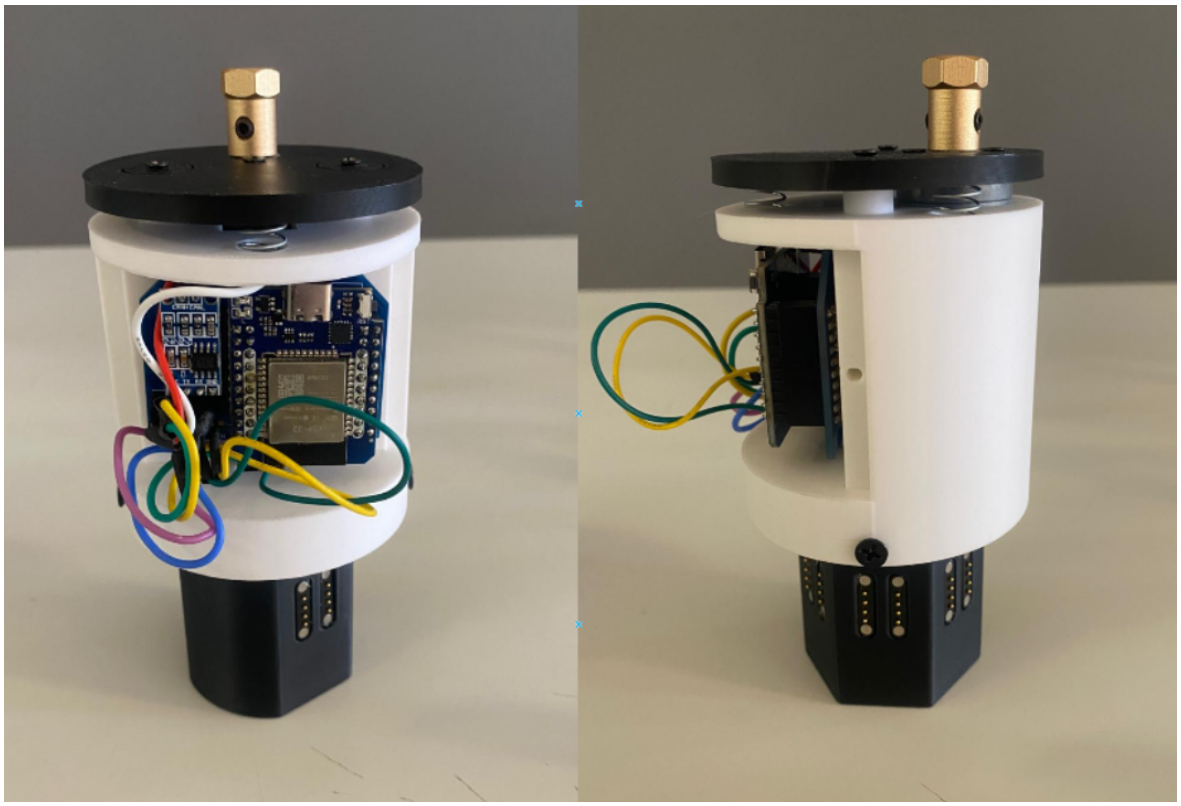


Figure 7.12: Manufactured prototype of the rotation module

The manufactured prototype demonstrates the successful implementation of the proposed mechanical design and confirms that the rotation module can be integrated into the modular platform using the same standardized interface as the gripper module. Although the mechanical design provides the necessary features for integrating a limit switch, this functionality was not implemented within the scope of this work.

8 Software Architecture & Implementation

8.1 System Overview

The developed software is based on a master–slave architecture, in which the gadgets represent the slave units, while the ESP32 integrated into the main electronics acts as the master. In the implemented system, the rotation module is referred to as Gadget 1, whereas the gripper module is designated as Gadget 2.

The objective of the software architecture is to enable the independent operation and replacement of different functional modules (gadgets) without requiring any modifications to the master's software. Each gadget performs a clearly defined function, while the master is responsible for the central coordination of the overall system.

The master acquires the user input signals from an surface electromyography (sEMG) sensor and a potentiometer, manages the communication between the individual system components, and provides the connected modules with the information required for their operation. In contrast, the slave units exclusively implement the control logic of their respective functional modules. This results in a clear separation between central system management and module-specific functionality.

Communication between the master and the slave units is realized via a Controller Area Network (CAN) bus, which serves as the common communication medium for all system components. In addition to transmitting sensor and control data, the communication interface enables the automatic detection of connected modules, their configuration during system initialization, and continuous monitoring of the communication link. Consequently, functional modules can be connected or disconnected during operation without requiring any modifications to the software.

To improve maintainability and adaptability, hardware- and communication-specific parameters are not embedded directly in the source code but are instead loaded from an external configuration file during system startup. The configuration loaded by the master is subsequently transmitted to the connected module. As a result, hardware-related modifications, such as changes in pin assignments, can be accommodated by updating the configuration file alone, without requiring changes to the source code of either the master or the slave units.

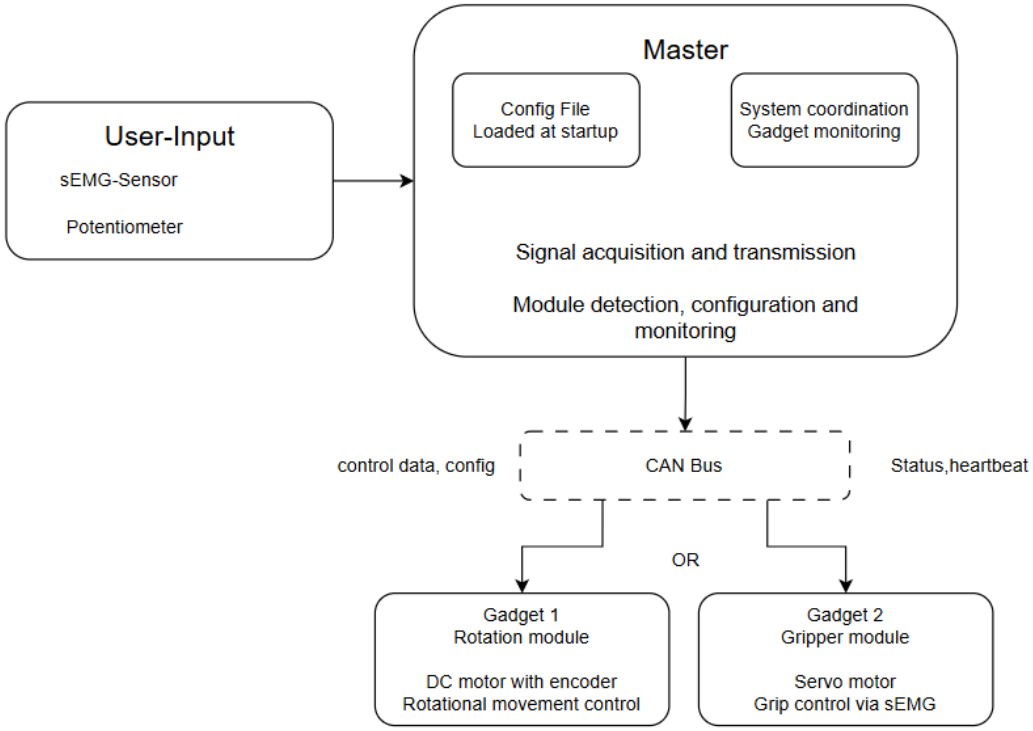


Figure 8.1: System Overview

Figure 8.1 illustrates the overall software architecture of the developed system. The master processes the user input signals and transmits the corresponding control information to the currently active gadget via the CAN bus. The respective gadget interprets the received data and controls the associated actuator according to its implemented functionality.

8.2 Development Environment

The software was developed using PlatformIO in combination with Visual Studio Code and the Arduino framework. Compared to the conventional Arduino IDE, PlatformIO provides a more structured project organization, integrated library management, and the ability to manage multiple firmware versions within a single project. These features are particularly advantageous for modular embedded systems consisting of multiple microcontrollers.

Since the developed system consists of one master unit and two different slave modules, the project was divided into three separate build environments (master, slave1, and slave2). Each build environment has its own configuration regarding source code, libraries, and build settings, while all firmware versions are maintained within the same project. This approach allows the software for the individual modules to be developed, compiled, and uploaded to the corresponding ESP32 independently, without the need to maintain multiple projects.

The Arduino framework was selected as the software foundation because it provides a comprehensive hardware abstraction layer for the ESP32 while still allowing access to low-level hardware features such as the CAN interface and hardware interrupts. This enables both rapid software development and direct control of the required hardware components.

Several libraries were incorporated to implement the required software functionality. The ESP32-TWAI-CAN library provides access to the ESP32's integrated CAN controller and offers functions for transmitting and receiving CAN messages. TinyXML2 was used to parse and process the XML configuration file. The configuration file itself is stored in the master's LittleFS file system and is loaded during system startup. In addition, the ESP32Servo library is used to control the servo motor of the gripper module by generating the required PWM signals.

The combination of PlatformIO, the Arduino framework, and the selected libraries provides a structured and extensible development environment that supports the implementation of the modular software architecture. In particular, the separation of the firmware into multiple build environments simplifies project maintenance and enables the independent development of the individual system components.

8.3 Configuration Management

To avoid hard-coded configuration parameters, all hardware-, communication-, and control-related settings are stored in an external XML configuration file. This approach allows modifications to the system configuration without requiring changes to the source code or recompilation of the firmware.

The configuration file is stored in the LittleFS file system of the master ESP32 and is loaded during system startup. After being successfully loaded, the configuration parameters are stored in dedicated data structures, making them available to the entire software. This approach establishes a clear separation between the application logic and the system-specific configuration data.

The XML configuration file contains all parameters required for system operation. These include the pin assignments of the master and the individual gadgets, CAN communication parameters, input signal scaling ranges, motor and servo control parameters, as well as the CAN identifiers used throughout the system. In addition, parameters for future extensions, such as the PID controller of the actuator modules, are already included in the configuration file.

The XML file is parsed using the TinyXML2 library. After parsing the XML structure, the individual configuration parameters are assigned to their corresponding data structures within the software as seen in Figure 8.2. Separate configuration structures are defined for the master, the rotation module, the gripper module, and the CAN identifiers. This organization improves code maintainability and provides a structured and intuitive way to access the configuration parameters.

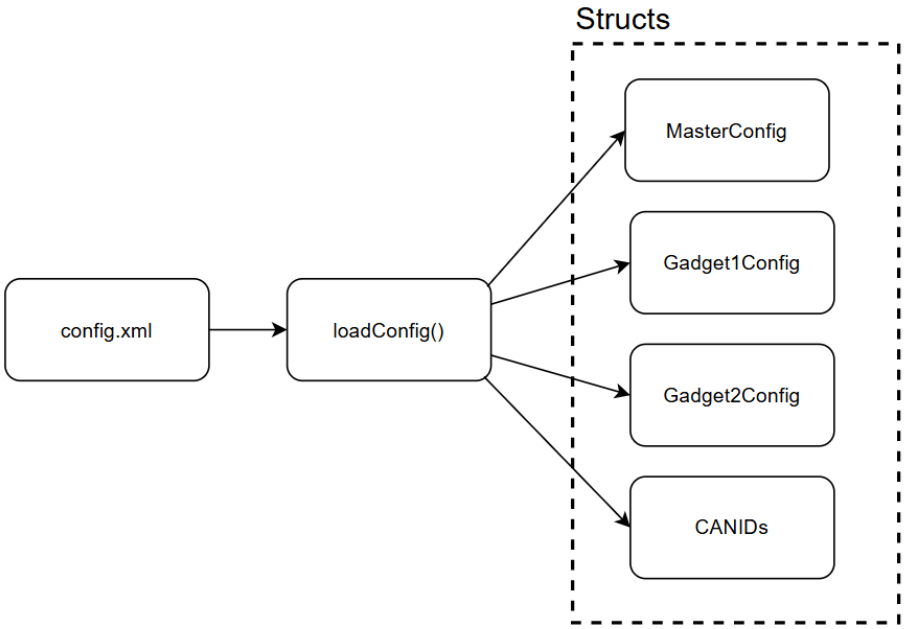


Figure 8.2: XML configuration loading and assignment to the configuration structures

Using an external configuration file significantly increases the flexibility of the system. Changes to the hardware configuration, such as modified pin assignments, adjusted scaling ranges, or updated communication parameters, can be implemented by modifying only the XML configuration file. Consequently, the application software remains unchanged and can be reused across different hardware configurations.

```
<config>
    <master>
        ...
    </master>

    <gadget> <!-- Gadget 1: Rotation Module -->
        ...
    </gadget>

    <gadget> <!-- Gadget 2: Gripper Module -->
        ...
    </gadget>

    <can_ids>
        ...
    </can_ids>
</config>
```

Figure 8.3: Config.xml-structure

Figure 8.3 illustrates the general structure of the XML configuration file. Its hierarchical organization enables an unambiguous assignment of configuration parameters to the corresponding system components and contributes to a clear and well-structured organization of the configuration data.

8.4 CAN Communication Protocol

Communication between the master and the individual gadgets is established via the integrated Controller Area Network (CAN) controller of the ESP32 microcontrollers. For the developed modular prosthetic system, a dedicated communication protocol was defined to support the automatic detection of newly connected modules, the transmission of configuration data, and the continuous exchange of sensor and control information.

Each message is identified by a unique CAN identifier (CAN ID), allowing different message types to be distinguished unambiguously. These include messages for module announcement, configuration data transmission, sensor data from the master, and heartbeat messages used for monitoring the communication link. An overview of all CAN identifiers used in the system is provided in Table 8.1.

Table 8.1: CAN message identifiers

ID (hex)	ID (dec)	Direction	Payload	Purpose
0x01	1	Slave → Master	data[0] = Slave ID (1 byte)	Module announcement
0x01	1	Master → Slave	data[0] = potiDataId, data[1] = heartbeatId, data[2] = configId (3 bytes)	CAN ID assignment
0x10	16	Master → Gadget 1	5 frames: pins, encoder, PID gains, output limits	Gadget 1 configuration
0x20	32	Master → Gadget 2	4 frames: pins, PID gains, output limits	Gadget 2 configuration
0x30	48	Master → Active gadget	Scaled float value, 4 bytes	Potentiometer value
0x40	64	Master → Active gadget	Raw ADC value, 2 bytes	sEMG raw signal
0x50	80	Slave → Master	data[0] = Slave ID, 1 byte, every 100 ms	Connection monitoring

Note: ID 0x01 is hardcoded on all units and used before any CAN ID assignment. All other IDs are loaded from the configuration file and transmitted during the handshake.

Since a single CAN frame can transmit a maximum payload of eight data bytes, larger configuration datasets are divided into multiple consecutive messages. This approach enables the complete transmission of pin assignments, floating-point values, PID parameters, and additional configuration data to the respective gadget. The sequence of these messages is predefined, allowing the receiving module to assign the transmitted parameters correctly to the corresponding configuration structures.

A special feature of the communication protocol is the announce message, which is transmitted

by a newly connected gadget immediately after system startup or after being connected to the CAN bus. The corresponding CAN identifier is hard-coded in both the master and all gadgets. Since this message represents the initial communication step between the system components, its identifier cannot be loaded from the external configuration file. Only after a successful module announcement does the master transmit the remaining CAN identifiers, which are subsequently used for all further communication.

In addition to the transmission of configuration data, the communication protocol supports the continuous exchange of control information. While the master cyclically transmits the acquired sEMG and potentiometer values to the active gadget, the gadgets periodically send heartbeat messages back to the master. This mechanism enables the master to continuously monitor the communication status and detect the disconnection or failure of a connected module.

8.5 Handshake & Connection Management

After system startup, the master enters a waiting state and listens for the announcement of an available gadget. Immediately after startup or after being connected to the CAN bus, each gadget transmits an announce message containing its unique module identifier. Based on this identifier, the master determines which functional module has been connected.

After successfully detecting the gadget, the master responds with a message containing the CAN identifiers required for subsequent communication. The master then transmits the complete configuration of the corresponding gadget. Due to the limited payload size of a CAN frame, the configuration data is transferred using multiple consecutive messages. Once all configuration data has been received, the gadget initializes its hardware using the transmitted parameters and subsequently enters normal operation.

During normal operation, data is exchanged continuously between the master and the connected gadget. The master cyclically transmits the current EMG and potentiometer values, while the connected gadget periodically sends heartbeat messages to the master. This ensures continuous communication between both system components.

To monitor the communication status, the master implements a heartbeat mechanism. Whenever a heartbeat message is received, the timestamp of the last successful communication is updated. If the elapsed time since the last received heartbeat exceeds a predefined timeout, the corresponding gadget is considered disconnected and is removed from the system. This enables the automatic detection of communication failures or the physical removal of a module.

The current connection state is managed by the variable activeGadgetId. The master only starts transmitting sensor and control data after a gadget has been successfully detected, configured, and registered as active. If a timeout occurs or a gadget is disconnected, activeGadgetId is reset,

automatically stopping data transmission until a valid gadget is detected again.

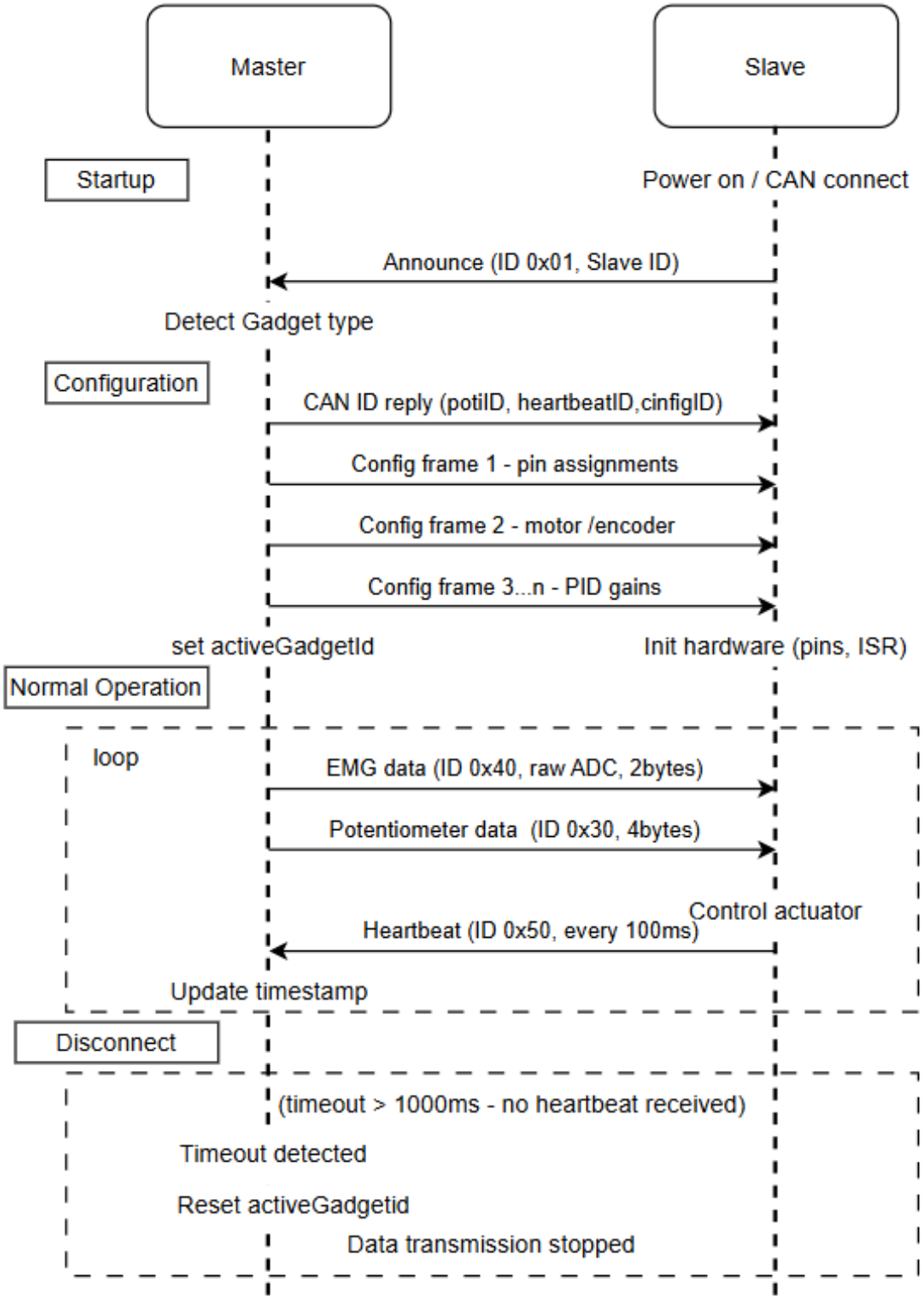


Figure 8.4: Handshake and connection management sequence between the master and a gadget.

Figure 8.4 illustrates the complete connection management procedure, beginning with the announce message and ending with the disconnection of the Gadget.

8.6 Gadget 1 - Rotation Module

The rotation module is responsible for controlling the DC motor that generates the rotational movement of the prosthetic hand. The software of the module performs hardware initialization, acquires encoder data, calculates the current rotational speed, and controls the motor based on the control data received from the master.

After the handshake procedure has been completed and the configuration data has been received, all hardware components are initialized. This includes the encoder inputs, the PWM outputs used to drive the H-bridge, and the encoder interrupt. Since all hardware parameters are transmitted by the master during initialization, the same firmware can be operated with different hardware configurations.

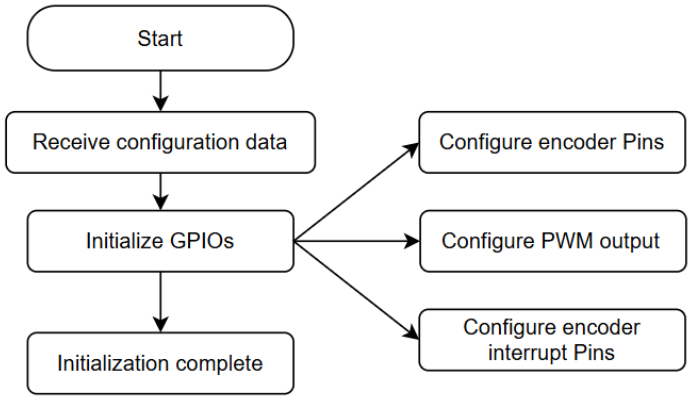


Figure 8.5: Initialization sequence of the rotation module

To measure the rotational movement, the integrated Hall-effect encoder of the DC motor is used. The pulses generated by encoder channel A are detected using a hardware interrupt, ensuring reliable pulse counting regardless of the execution time of the main loop. The direction of rotation is determined by evaluating the logic level of encoder channel B, allowing both clockwise and counterclockwise movements to be detected.

The current rotational speed is calculated at fixed time intervals based on the number of detected encoder pulses. Both the encoder resolution and the gearbox ratio of the motor are taken into account to determine the rotational speed of the output shaft. The rotational speed is calculated according to Equation 8.1:

$$\mathrm{RPM} = \frac{N_{\mathrm{pulses}}}{\Delta t} \cdot \frac{60000}{\mathrm{PPR} \cdot i} \tag{8.1}$$

where

N_{pulses} is the number of detected encoder pulses,

Δt is the measurement interval in milliseconds,

PPR is the number of encoder pulses per motor revolution, and

i is the gearbox ratio.

The rotational speed is updated every 20 ms, providing a continuous estimate of the motor speed during operation. In the current implementation, the calculated RPM value is used for monitoring and debugging purposes and forms the basis for a future closed-loop PID controller.

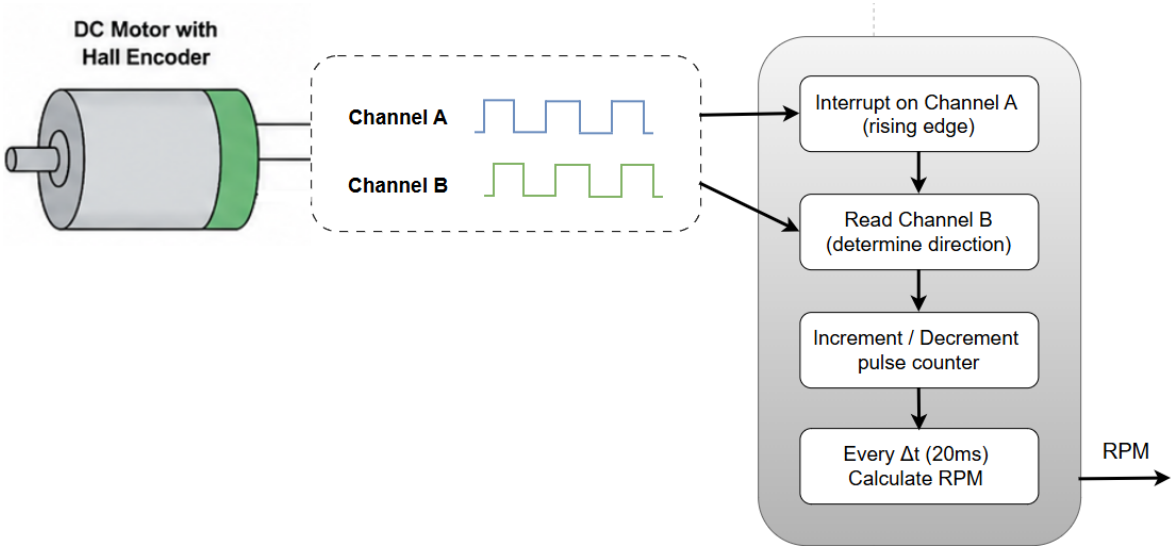


Figure 8.6: Principle of encoder signal evaluation and RPM calculation

Motor control is based on the potentiometer values received from the master. The potentiometer range is divided into two regions, with the center position representing the neutral position. Potentiometer values below the center position generate rotation in one direction, whereas values above the center position produce rotation in the opposite direction. The distance from the center position simultaneously determines the motor speed.

To generate the corresponding motor speed, the received potentiometer value is linearly mapped to the PWM range of the motor driver. For the two directions of rotation, the PWM value is calculated according to

$$\text{PWM} = \frac{50 - P}{50} \cdot 255 \qquad (P \leq 50) \tag{8.2}$$

$$\text{PWM} = \frac{P - 50}{50} \cdot 255 \qquad (P > 50) \tag{8.3}$$

where (P) represents the potentiometer value transmitted by the master in the range from 0 to 100. The calculated PWM values are subsequently limited to the valid range of 0 to 255 before being applied to the corresponding input of the H-bridge (Figure 8.7).

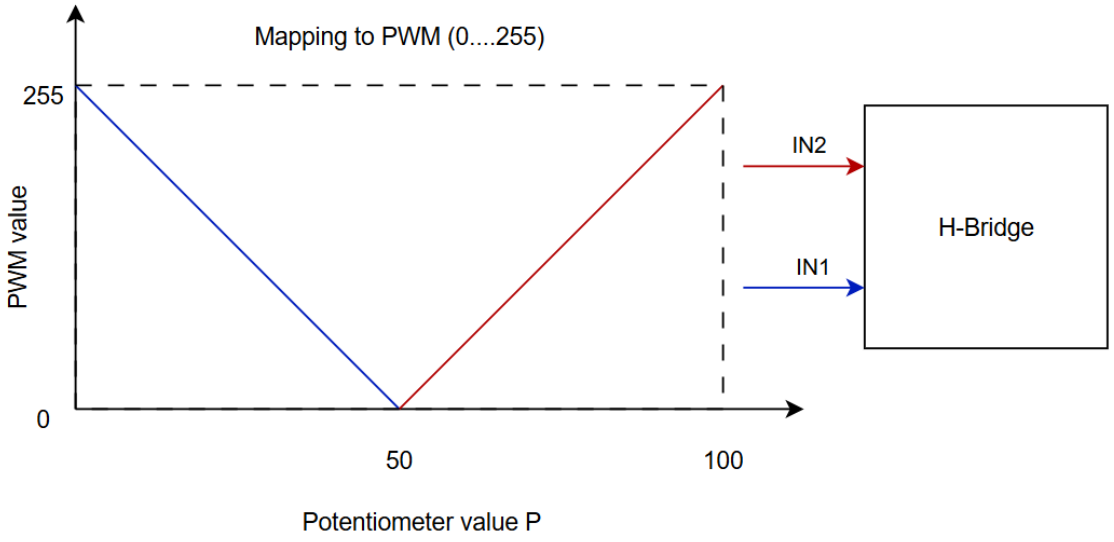


Figure 8.7: Mapping of the potentiometer value to motor direction and PWM duty cycle

The software already contains the basic structure required for a future PID controller. The corresponding controller parameters are transmitted by the master during initialization and stored in the configuration structure. However, the current implementation uses an open-loop PWM control strategy. This approach prepares the software for future closed-loop control without requiring modifications to either the communication interface or the overall software architecture.

8.7 Gadget 2 Gripper Module

The gripper module is responsible for controlling a servo motor that performs the opening and closing movement of the gripper. The software initializes the required hardware components and converts the EMG control information received from the master into the corresponding servo movement.

After the handshake procedure has been completed and the configuration data has been received, the servo output and the remaining hardware components are initialized. As with the rotation module, all hardware parameters are transmitted by the master during initialization, allowing the same firmware to be used with different hardware configurations.

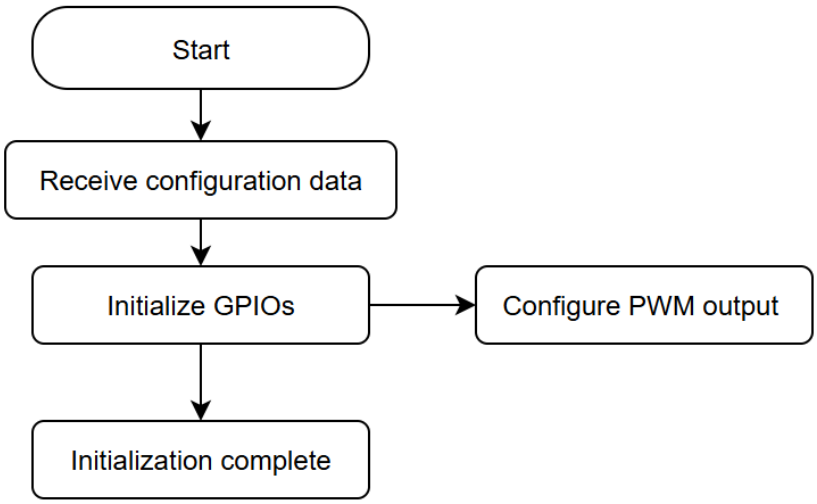


Figure 8.8: Initialization sequence of the gripper module

The gripper is controlled exclusively based on the EMG signal received from the master. The current EMG value is compared with a predefined threshold. If the measured value exceeds this threshold, the target position of the servo motor is set to 90°. Otherwise, the target position is set to 0°.

The target position can therefore be described by

$$\theta_{\text{target}} = \begin{cases} 90°, & \text{if } EMG > \text{Threshold} \\ 0°, & \text{if } EMG \leq \text{Threshold} \end{cases} \tag{8.4}$$

To avoid abrupt gripper movements, the servo does not move directly to the target position. Instead, the current servo position is incrementally adjusted by 1° toward the target position every 20 ms. This incremental approach results in a smooth servo motion while reducing mechanical stress on the gripper mechanism.

The servo position is updated according to

$$\theta_{k+1} = \begin{cases} \theta_k + 1°, & \text{if } \theta_k < \theta_{\text{target}} \\ \theta_k - 1°, & \text{if } \theta_k > \theta_{\text{target}} \\ \theta_k, & \text{if } \theta_k = \theta_{\text{target}} \end{cases} \tag{8.5}$$

where θ_k denotes the current servo position and θ_{target} represents the target position determined from the EMG signal.

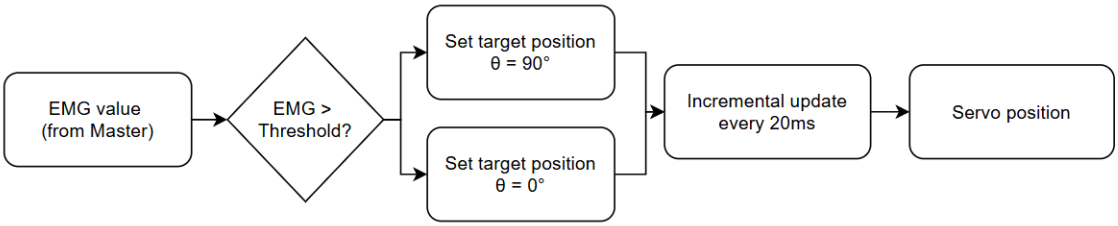


Figure 8.9: Incremental servo movement towards the target position

Although an input for a pressure sensor is already included in the configuration, the sensor is not yet evaluated in the current implementation. Nevertheless, the existing software structure allows future extensions of the gripper module with sensor-based control without requiring fundamental modifications to the overall software architecture.

8.8 Sensor Data Acquisition and Visualization

8.8.1 Overview

The development of a closed-loop control system first requires an analysis of the system behavior. For this purpose, sensor data must be acquired, visualized, and analyzed under defined operating conditions. Since the serial interface integrated into the microcontroller is only of limited suitability for this task, a dedicated wireless data acquisition tool was developed as part of this work.

The system consists of two independent software components. The first component is executed on an ESP32 microcontroller and is responsible for the periodic acquisition of sensor data as well as its transmission over a Wi-Fi network using UDP. The second component is a desktop application developed with the Qt framework. It receives the transmitted measurement data, visualizes the sensor signals in real time, and additionally provides the functionality to store the recorded data in CSV format.

The recorded datasets provide the basis for further investigations, such as characterizing the system behavior, analyzing sensor signals, and serving as input data for the future design and parameterization of a closed-loop control system.

8.8.2 Software Architecture

The functionality of the system is divided into two separate software components. While the ESP32 is exclusively responsible for the time-critical acquisition and transmission of sensor data,

the desktop application performs the processing and visualization of the received measurement data.

The ESP32 firmware follows a modular design and consists of a main application that controls the program flow, as well as separate classes for data buffering and Wi-Fi/UDP communication. This modular structure clearly separates the individual tasks, allowing each component to be extended or modified independently.

The desktop application follows a similar architectural approach. It combines the reception of UDP data packets, real-time visualization of the acquired sensor data, and the export of recorded measurements within a single graphical user interface. This structure ensures a clear separation between data acquisition, communication, and data processing.

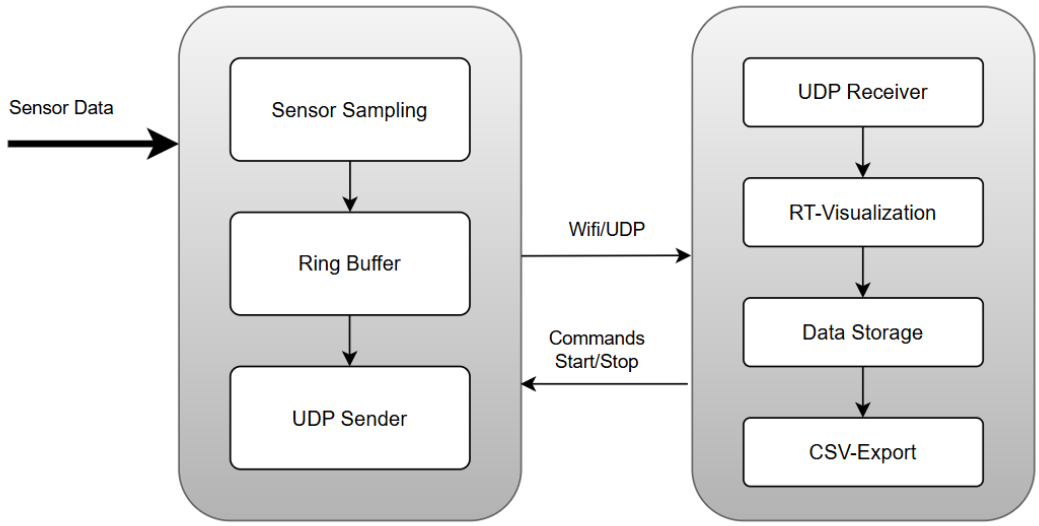


Figure 8.10: System architecture of the developed data acquisition tool

8.8.3 Data Acquisition

The ESP32 firmware is responsible for the time-critical acquisition of sensor data as well as its wireless transmission to the desktop application. Sensor data is acquired at fixed sampling intervals. In the current implementation, a sampling rate of 200 Hz was selected, as the system was primarily developed for the acquisition and analysis of EMG sensor signals. For other sensor types, the sampling rate can be adjusted according to their dynamic characteristics.

The acquired measurement data is initially stored in a ring buffer rather than being transmitted immediately over the network. The use of the ring buffer is necessary because the maximum achievable UDP transmission rate does not necessarily match the sampling rate of the connected sensor. In particular, sensor data may be lost when using sensors with higher sampling frequencies if every measurement were transmitted immediately after acquisition.

The implemented ring buffer therefore decouples data acquisition from data transmission. While newly acquired samples are continuously written to the write buffer, a previously filled memory buffer can simultaneously be transmitted over the network. After the transmission has been completed successfully, the two buffers are swapped and the acquisition process continues. This double-buffering approach reduces the communication overhead, enables the transmission of contiguous blocks of measurement data, and ensures that the acquired samples are received in the correct chronological order.

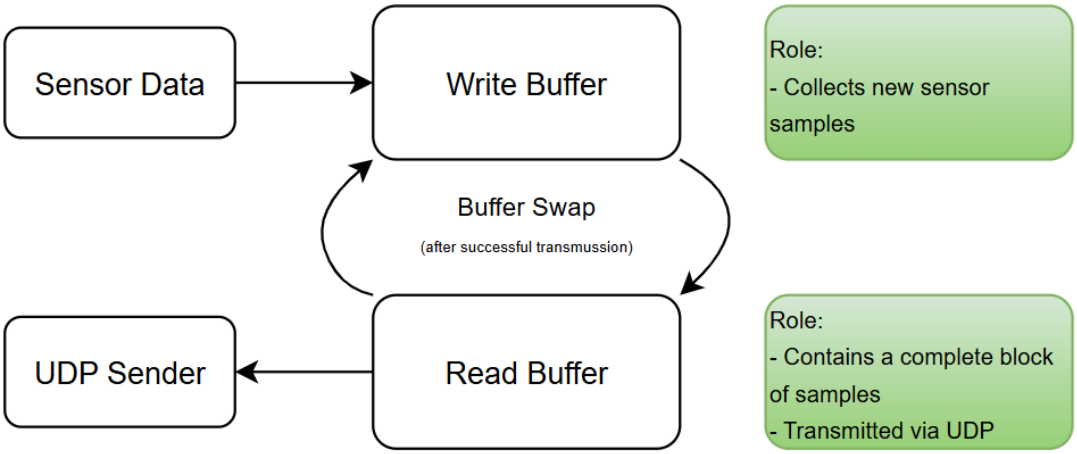


Figure 8.11: Double-buffer implementation for real-time data acquisition

Data acquisition is controlled by simple commands sent from the desktop application. Upon receiving a START command, the ESP32 begins recording sensor data. A STOP command terminates the acquisition process and discards any partially filled data block. As a result, only complete measurement packets are transmitted to the desktop application.

8.8.4 Data Visualization and User Interface

The desktop application serves as the user interface of the developed data acquisition system. It receives the UDP data packets transmitted by the ESP32, visualizes the acquired sensor data in real time, and enables the recorded measurements to be stored for subsequent analysis.

Sensor data is received via a QUdpSocket, which continuously listens for incoming UDP datagrams. After successful reception, the measurement data is converted into floating-point values and displayed using the QCustomPlot library. Instead of transmitting individual samples, each UDP packet contains a complete data block corresponding to the size of the ring buffer. After reception, all samples of the received block are sequentially added to the plot, resulting in a continuous real-time visualization despite the block-based transmission.

The graphical user interface also provides basic control of the data acquisition process through the Start and Stop buttons. In addition, previously recorded measurement data can be deleted or

exported in CSV format. The CSV export stores both the timestamp and the corresponding sensor value for each recorded sample, allowing the acquired data to be used for further analysis, for example in MATLAB.

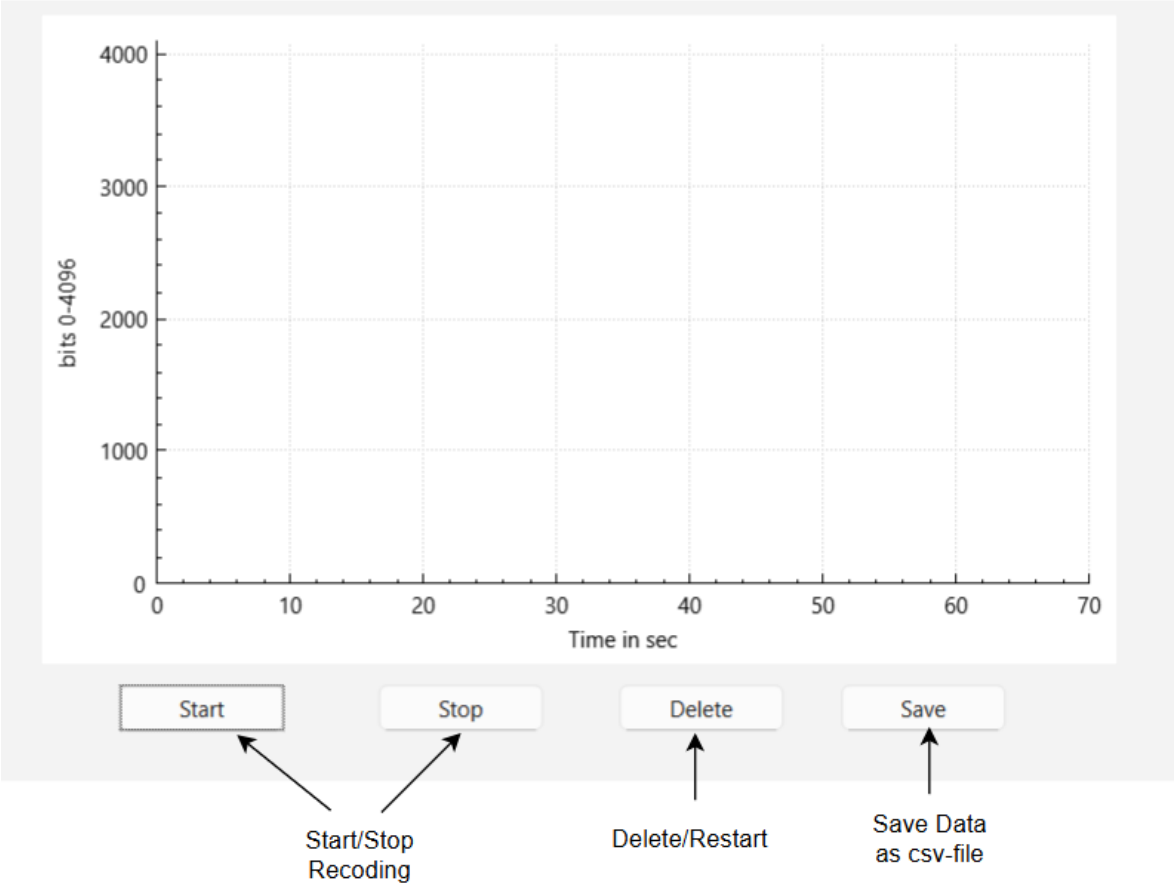


Figure 8.12: Graphical user interface for real-time visualization and recording of sensor data

9 Cost Analysis and Project Schedule

9.1 Engineering Costs

The engineering costs comprise all activities required to conceptually develop the proposed modular prosthetic platform before its physical realization. These costs include the research and development effort necessary to establish the mechanical, electrical, and software concepts as well as the corresponding system architecture. The development of the proposed platform required a total engineering effort of 606 working hours based on a full-time development period of 3.5 months and an engineering rate of 30 €/h.

Table 9.1 summarizes the engineering effort required for the individual development activities. The distribution reflects the complete engineering process from the initial research phase to the development of the mechanical, electrical, and software concepts of the modular prosthetic platform.

Table 9.1: Engineering costs for the development of the modular platform.

Engineering Activity	**Effort [h]**	**Hourly Rate [€/h]**	**Cost [€]**
Research and Requirement Analysis	95	30	2,850
System Design	90	30	2,700
Mechanical Design	165	30	4,950
Electrical Design	131	30	3,930
Software Design	125	30	3,750
Total	**606**		**18,180**

In addition to engineering effort, software tools are required throughout the complete development process. Most of the tools used during the development of the proposed platform are available free of charge. The only commercial software required for the engineering process was the CAD software Solid Edge. Table 9.2 provides an overview of the development tools together with their corresponding commercial license costs.

Table 9.2: Development tools and estimated commercial license costs.

Tool	Application	License Cost
Solid Edge	Mechanical CAD design	Approx. 2900€/year
KiCad	Schematic and PCB design	Free
Visual Studio Code	Embedded software and XML development	Free
draw.io	System diagrams and technical illustrations	Free
TeXstudio	Technical documentation	Free
Bambu Studio	Preparation of FDM manufacturing files	Free

The listed Solid Edge cost represents the full annual commercial subscription based on the publicly available starting price. The actual license cost may vary depending on the selected software package, regional pricing, and contractual conditions.

Combining the estimated engineering effort with the required software licenses provides an overall estimation of the engineering costs associated with the development of the proposed modular platform. The resulting cost estimation is summarized in Table 9.3.

Table 9.3: Summary of the estimated engineering costs.

Cost Category	Estimated Cost [€]
Engineering activities	18,180
Development tools	Approx. 2,900
Total engineering costs	**Approx. 21,080**

9.2 Execution Costs

The execution costs include all resources required for the physical realization of the proposed modular prosthetic platform. In contrast to the engineering costs, these costs comprise the hardware components, manufacturing equipment, and labour required to assemble, integrate, and validate the complete prototype.

Table 9.4 summarizes the material costs of the complete prototype. The listed components include the electronic hardware, mechanical components, additive manufacturing materials, and PCB manufacturing required to realize the developed platform.

Table 9.4: Component and material costs of the complete prototype.

Component or Material	Quantity	Unit Cost [EUR]	Total Cost [EUR]
ESP32 Mini	3	5.99	17.97
TJA1050 CAN transceiver	3	1.49	4.47
XL6009 DC-DC converter	2	2.49	4.98
LTC1871 DC-DC converter	1	4.99	4.99
TP4056 charging module	1	1.49	1.49
MAX1044 charge-pump IC	1	0.79	0.79
EMG Sensor V3.0	1	21.99	21.99
DSSERVO 25 kg servo motor	1	20.99	20.99
JGA25-371 DC geared motor with encoder	1	13.99	13.99
L9110 motor driver	1	1.99	1.99
Li-Ion battery	1	29.99	29.99
10 kΩ potentiometer	1	0.29	0.29
Power switch	1	0.19	0.19
100 μF capacitors	2	0.03	0.06
PCB manufacturing (PCBWay)	2	10.00	20.00
Pogo pins	12	2.92	35.04
Terminal block set	1	8.00	8.00
Screw set	1	15.00	15.00
Spring set	1	10.00	10.00
PLA filament	3 kg	14.99	44.97
TPU filament	35 g	0.90	0.90
Total			**281.10**

In addition to the prototype components, specialized equipment is required for manufacturing, assembly, and validation. Table 9.5 lists the primary equipment used throughout the realization of the prototype together with the corresponding acquisition costs.

Table 9.5: Development equipment required for prototype realization.

Equipment	Quantity	Cost [EUR]
Bambu Lab A1 Mini 3D printer	1	189.00
Hanmatek HM310 laboratory power supply	1	66.50
Portable oscilloscope	1	35.00
Digital multimeter	1	150.00
Soldering station	1	120.00
Soldering consumables (solder wire, flux, desoldering braid, solder sucker)	1	35.00
Total		**595.50**

Besides material and equipment costs, the realization of the prototype requires a considerable amount of manual labour. The implementation includes PCB assembly, mechanical assembly, system integration, troubleshooting, and functional testing. The corresponding labour costs are summarized in Table 9.6.

Table 9.6: Labour costs for prototype realization and system integration.

Execution Activity	**Effort [h]**	**Hourly Rate [EUR/h]**	**Cost [EUR]**
PCB assembly	16	30	480
Mechanical assembly	120	30	3,600
System integration	40	30	1,200
Troubleshooting	120	30	3,600
Functional testing	40	30	1,200
Total	**336**		**10,080**

The total execution costs are obtained by combining the material costs, required equipment, and labour associated with the realization of the complete prototype. A summary of these costs is provided in Table 9.7.

Table 9.7: Summary of the execution costs.

Cost Category	**Cost [EUR]**
Prototype components and materials	281.10
Development equipment	605.50
Labour costs	10,080.00
Total execution costs	**10,966.60**

9.3 Overall Cost Summary

The overall project costs consist of both the engineering costs required to develop the modular platform and the execution costs associated with its physical realization. Table 9.8 summarizes the total development costs of the proposed modular prosthetic platform.

Table 9.8: Overall cost summary of the developed modular prosthetic platform.

Cost Category	**Cost [EUR]**
Engineering costs	21,080.00
Execution costs	10,966.60
Overall project costs	**32,046.60**

The resulting total development costs provide a realistic estimation of the resources required to design, implement, and validate a functional prototype of the proposed modular prosthetic platform. The presented cost breakdown further illustrates the distribution of effort between engineering activities and prototype realization.

9.4 Project Schedule

The development of the proposed modular prosthetic platform was carried out between February and July 2026. The project was structured into seven main work packages, covering literature research and requirement analysis, mechanical and electrical design, software development, prototype manufacturing, prototype assembly and integration, system validation, and thesis documentation. Figure 9.1 illustrates the chronological sequence and overlap of the individual project phases throughout the development process.

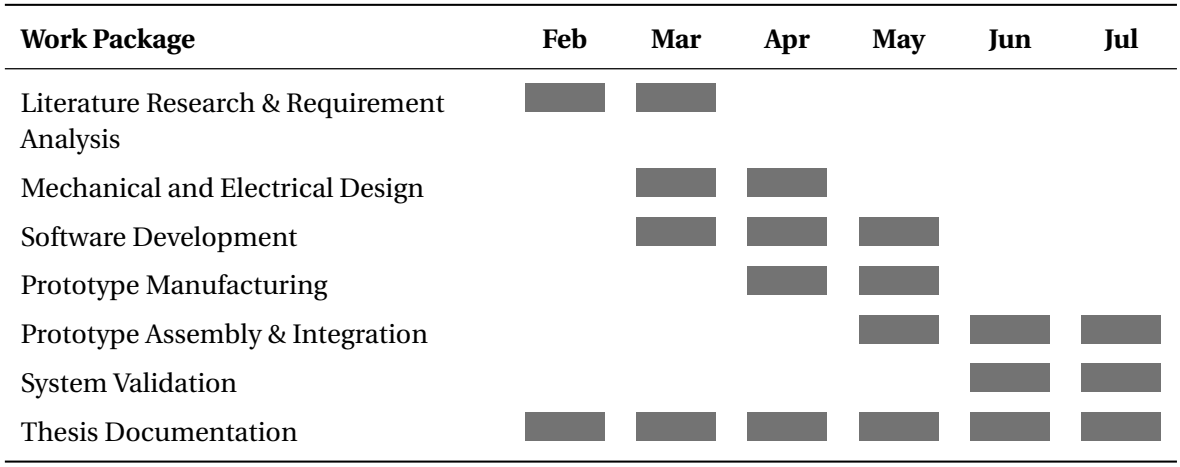


Figure 9.1: Project schedule from February to July 2026.

10 Experimental Validation and Evaluation

10.1 Test Methodology

To evaluate the developed modular prosthetic platform, a series of experiments was conducted. The objective of the evaluation was to verify the functionality of the developed hardware and software as well as to validate the design decisions presented in the previous chapters. The performed tests are summarized below.

Table 10.1: Overview of the experimental validation tests.

Test	**Objective**
Gripper load test	Validate the calculated gripping force and verify the maximum payload of the developed gripper.
Coupling reliability test	Evaluate the mechanical and electrical reliability of the standardized coupling interface during repeated module exchange.
Functional system test	Verify the interaction between user input, communication, and module actuation within the complete modular prosthetic platform.

10.2 Gripper Load Validation

The objective of this experiment was to evaluate the practical gripping capability of the developed gripper module under different payload conditions. The experiment was intended to validate whether the gripper was capable of securely holding objects with masses of up to 2 kg, as predicted during the mechanical design phase.

Initially, a standard PET bottle was used as the test object. The bottle was gradually filled with water to obtain masses of 0.5 kg, 1.0 kg, and 1.5 kg. A final test with a total mass of 2.0 kg was carried out by additionally filling the bottle with stones. During the first test, however, it became apparent that the smooth surface of the PET bottle resulted in insufficient friction between the bottle and the TPU contact surfaces of the gripper. Consequently, the object could not be held reliably, preventing a meaningful evaluation of the maximum payload.

To investigate the influence of the contact surface, the experiment was repeated using a glass bottle. Due to the higher surface friction between the glass bottle and the TPU gripper pads, the

bottle could be securely held with a mass of 0.5 kg. At a mass of 1.0 kg, however, the object could no longer be gripped reliably.

10.3 Coupling and Connection Reliability

The objective of this experiment was to evaluate the reliability of the standardized coupling interface during repeated module exchanges. The test focused on the mechanical locking mechanism and the establishment of a reliable electrical connection between the main unit and the interchangeable modules.

The experiment consisted of 100 consecutive module exchange cycles, with the gripper module and the rotation module connected alternately, resulting in 50 connection cycles for each module. The modular prosthetic platform remained powered throughout the entire experiment. After each module insertion, it was verified that the locking mechanism was fully engaged and that a reliable electrical connection was established. The electrical connection was verified using a multimeter by checking the continuity of the interface contacts. Table 10.2 summarizes the results of the experiment.

Table 10.2: Results of the coupling reliability test.

Parameter	**Value**
Total connection cycles	100
Gripper module cycles	50
Rotation module cycles	50
Successful connections	98
Connections requiring reinsertion	2
Success rate	98%

Table 10.3: Results of the gripper load validation.

Test Object	**Mass [kg]**	**Result**	**Remark**
PET bottle	0.5	Fail	Insufficient friction
PET bottle	1.0	Fail	Insufficient friction
PET bottle	1.5	Fail	Insufficient friction
PET bottle	2.0	Fail	Insufficient friction
Glass bottle	0.5	Pass	Stable grasp
Glass bottle	1.0	Fail	Grip not reliable

Out of the 100 performed connection cycles, 98 resulted in a reliable mechanical and electrical connection immediately after insertion. In two cases, the electrical contact was not fully established and the module had to be slightly repositioned to ensure reliable contact between the interface pins.

10.4 Functional System Validation

The objective of this experiment was to validate the functionality of the complete modular prosthetic platform by verifying the interaction between the user interface, the communication architecture, power supply, and the implemented modules. The experiment evaluated the complete signal chain from user input to module actuation using both developed prosthetic modules.

The experiment was carried out with the modular prosthetic platform operating continuously. Both the gripper module and the rotation module were connected to the system and tested using the implemented user interface. The platform was evaluated using both an external laboratory power supply and the integrated battery supply. During the experiment, the functionality of the EMG-based activation, module recognition, XML configuration loading, CAN communication, potentiometer control, power supply, and module actuation was assessed. Table 10.3 summarizes the observed results.

Table 10.4: Results of the functional system validation.

Function	**Observation**	**Result**
EMG-based activation	The EMG sensor successfully activated the system. Variations in activation reliability were observed during operation.	Partial
Master controller	User input was successfully processed and forwarded to the connected module.	Pass
Module recognition	Connected modules were detected in approximately every second connection attempt.	Partial
XML configuration loading	The corresponding module configuration was successfully loaded after module recognition.	Pass
CAN communication	Communication between the master controller and the connected module operated reliably throughout the experiment.	Pass
Power supply	The initial 5 V DC-DC converter exhibited unstable output voltage when the system was powered from an external laboratory power supply and was therefore replaced.	Partial
Potentiometer control	The implemented potentiometer control of the rotation module operated as intended.	Pass
Module actuation	Both implemented modules successfully executed the commanded mechanical motion.	Pass

The experiment demonstrates that the developed platform is capable of controlling different interchangeable modules using a common hardware and software architecture. During system validation, limitations were observed in the automatic module recognition and in the initially selected 5 V DC-DC converter when the system was supplied from an external laboratory power supply. The converter exhibited an unstable output voltage, resulting in damage to connected electronic components, and was therefore replaced by an alternative component that provided stable operation during the remaining experiments. The updated converter selection and the corresponding PCB revision are documented in the Appendix. The observed limitations and their implications

are discussed in Section 10.5.

10.5 Evaluation

The gripper load validation revealed that the achievable payload of the developed gripper is not solely determined by its mechanical design, but is strongly influenced by the contact conditions between the gripper and the manipulated object. During the experiment, the gripper was able to securely hold a glass bottle with a mass of up to 0.5 kg, whereas a PET bottle could not be gripped reliably even at the same load due to its low surface friction. As a result, the planned validation up to 2 kg could not be completed, since the observed failure was dominated by the surface properties of the test objects rather than by the mechanical capability of the gripper itself. Consequently, the experiment demonstrates that the selection of the contact material plays a crucial role in the overall gripping performance and should be considered in future iterations of the gripper design.

The coupling reliability test demonstrated that the proposed standardized interface provides a reliable mechanical and electrical connection during repeated module exchange. Out of 100 consecutive connection cycles, 98 were completed successfully without requiring any adjustment. In the remaining two cases, a slight repositioning of the module was necessary to establish a reliable electrical contact. Overall, the experiment confirms that the developed coupling interface provides a robust and repeatable mechanical and electrical connection.

The functional system validation confirmed the feasibility of the proposed modular platform. CAN communication, XML-based module configuration, and module actuation operated as intended, demonstrating that both implemented modules can be controlled using the same hardware and software architecture. The experiment further confirmed the successful interaction between the master unit and the interchangeable modules.

The functional validation also revealed two limitations of the current prototype. The automatic module recognition succeeded only in approximately every second connection attempt. Since the coupling reliability test verified the reliability of both the mechanical connection and the electrical contacts, the observed behavior is attributed to the current software implementation of the module detection algorithm rather than to the hardware interface. A likely cause is the heartbeat monitoring or timeout handling implemented in the communication software.

Furthermore, the initially selected 5 V DC-DC converter exhibited unstable output behavior when the system was supplied from an external laboratory power supply, resulting in damage to connected electronic components. Replacing the converter with an alternative device eliminated the observed instability during the remaining experiments. This result highlights the importance of validating not only the system architecture but also the reliability of individual hardware components during prototype development.

11 Summary

This thesis presented the design, implementation, and validation of a modular mechatronic platform for interchangeable assistive modules. The objective of this work was to establish a standardized hardware and software platform that enables the integration of application-specific modules through common mechanical, electrical, and software interfaces.

To achieve this objective, a complete prototype system was developed consisting of a master unit, a standardized coupling interface, dedicated electronic hardware, and a distributed embedded software architecture based on CAN communication. The proposed concept was demonstrated by implementing two interchangeable proof-of-concept modules with different mechanical functionalities: a gripper module and a rotation module.

The developed prototype successfully demonstrated the feasibility of the proposed modular architecture. The standardized coupling interface enabled reliable mechanical locking and electrical connectivity, while the communication architecture allowed different modules to operate using the same hardware and software infrastructure. Experimental validation confirmed the functionality of the developed platform and demonstrated that different application-specific modules can be integrated without modifying the underlying system architecture.

Although limitations were identified during system validation, particularly regarding the software implementation of the automatic module detection and the initially selected DC-DC converter, these issues do not affect the validity of the proposed platform concept. Overall, the presented work establishes a reusable foundation for the development of future interchangeable mechatronic assistive modules and demonstrates the feasibility of a modular prosthetic platform based on standardized interfaces.

12 Outlook

The modular platform presented in this thesis establishes a standardized foundation for the development of future interchangeable mechatronic assistive modules. While the proposed concept was demonstrated using a gripper module and a rotation module, the underlying hardware and software architecture is not limited to these applications. Future work may therefore focus on extending the platform by developing additional application-specific modules while reusing the existing standardized mechanical, electrical, and software interfaces.

Since the developed system represents a first functional prototype, further hardware optimization offers considerable potential. Future revisions may focus on reducing the size of both the main PCB and the secondary PCB by integrating the currently distributed electronics into more compact circuit board layouts. Likewise, the standardized coupling interface could be further miniaturized while preserving its mechanical robustness and electrical functionality. Such improvements would reduce the overall dimensions of the platform and facilitate the development of smaller and more compact interchangeable modules.

Another promising area for future work is the implementation of closed-loop control strategies. Although the current software architecture already provides support for configurable controller parameters, the implemented modules operate using open-loop control. Integrating appropriate feedback sensors would enable more accurate position and force control while improving the overall performance of the developed modules. Furthermore, the existing XML-based configuration concept could be extended by supporting over-the-air parameter updates, allowing controller parameters such as the proportional, integral, and derivative gains to be adjusted without modifying the embedded software.

Future work should also investigate more advanced processing of the acquired EMG signals. The presented prototype employs a simple threshold-based activation strategy, which was sufficient to demonstrate the functionality of the proposed modular platform. However, reliable interpretation of muscle activity for practical assistive applications requires more sophisticated signal processing or classification methods. Integrating approaches such as pattern recognition could significantly improve the robustness and intuitiveness of the user interface while enabling more complex control strategies.

Finally, future work may focus on the development of a complete wearable prosthetic system based on the proposed modular platform. While this thesis concentrated on the mechatronic platform and the interchangeable modules, further development of the wearable glove and its in-

tegration with the presented architecture would represent an important next step. Building upon the concepts established within the eSuperheroes project, such an integrated system could combine a customized wearable interface with standardized mechatronic modules, thereby further extending the applicability and functionality of the overall assistive platform.

Bibliography

[1] Advancer Technologies. *EMG Muscle Sensor V3.0 with Cable and Electrodes User Manual*. Accessed: 2026-04-15. Advancer Technologies. 2013. URL: https://www.rajguruelectronics.com/Product/582/Advance%20Technologies%20EMG%20Muscle%20Sensor%20V3.0%20With%20Cable%20And%20Electrodes.pdf.

[2] Arduino. *Arduino Mega 2560 Rev3 Datasheet*. Accessed: 2026-04-02. 2026. URL: https://docs.arduino.cc/resources/datasheets/A000067-datasheet.pdf.

[3] Nikhil M. Bajaj, Adam J. Spiers, and Aaron M. Dollar. "State of the Art in Artificial Wrists: A Review of Prosthetic and Robotic Wrist Design". In: *IEEE Transactions on Robotics* 35 (2019). DOI: 10.1109/TRO.2018.2865890. URL: https://ieeexplore.ieee.org/document/8624352.

[4] Joseph T. Belter and Aaron M. Dollar. "Performance Characteristics of Anthropomorphic Prosthetic Hands". In: *Proceedings of the IEEE International Conference on Rehabilitation Robotics (ICORR)*. Zurich, Switzerland: IEEE, 2011. DOI: 10.1109/ICORR.2011.5975340. URL: https://mindtrans.narod.ru/pdfs/Performance_Characteristics_of_Anthropomorphic_Prosthetic_Hands.pdf.

[5] Armand Beneteau et al. "Low-Cost Wireless Surface EMG Sensor Using the MSP430 Microcontroller". In: *Proceedings of the 6th European Embedded Design in Education and Research Conference (EDERC)*. Milan, Italy: IEEE, 2014. ISBN: 9781479968411. DOI: 10.1109/EDERC.2014.6924401. URL: https://strathprints.strath.ac.uk/50788/1/Beneteau_etal_IEEE_2014_Low_cost_wireless_surface_EMG_sensor.pdf.

[6] Kyler C. Bingham et al. "Adaptive Modular Prosthetic Wrists with Quick Swap, Full Rotation, and Electrical Connectivity". In: Paper No. DETC2025-169025. Anaheim, CA, USA: ASME, 2025. URL: https://asmedigitalcollection.asme.org/IDETC-CIE/proceedings/IDETC-CIE2025/89251/V005T08A043/1226041.

[7] Digby Chappell et al. "Beyond Humanoid Prosthetic Hands: Modular Terminal Devices That Improve User Performance". In: *IEEE Transactions on Neural Systems and Rehabilitation Engineering* 33 (2025). DOI: 10.1109/TNSRE.2025.3528725. URL: https://spiral.imperial.ac.uk/server/api/core/bitstreams/be60c752-abdf-4955-8d30-1a3ad845d4cb/content.

[8] I-Ming Chen, Yang Gao, and Guilin Zhang. "Design and Fabrication of a Novel Quick-Change System". In: *Mechatronics* 10 (2000). DOI: 10.1016/S0957-4158(99)00041-0. URL: https://www.sciencedirect.com/science/article/pii/S0957415899000410.

[9] Nikolaus Correll et al. “Introduction to Autonomous Robots: Mechanisms, Sensors, Actuators, and Algorithms”. In: 1st ed. Cambridge, MA: MIT Press, 2022. Chap. 10.2.3. ISBN: 9780262047555. URL: https://nikolaandro.github.io/books/ML/intro-to-autonomous-robots.pdf.

[10] Espressif Systems. *ESP32-WROOM-32 Datasheet*. Version 3.6. Accessed: 2026-4-03. URL: https://documentation.espressif.com/esp32-wroom-32_datasheet_en.pdf.

[11] María González-García et al. “The ‘Superheroes’ Project: Design and Fabrication of Low-Cost Personalized Prostheses”. In: *Societal Impacts* 6 (2025). DOI: 10.1016/j.socimp.2025.100137. URL: https://www.sciencedirect.com/science/article/pii/S2949697725000360.

[12] Handson Technology. *STM32F103C8T6 ARM Cortex-M3 Microcontroller Board Datasheet*. Accessed: 2026-04-05. URL: https://handsontec.com/dataspecs/module/STM32F103%20module.pdf.

[13] Sheng Hu et al. “A General Framework of Mechatronic Modular Architecture”. In: *Advances in Mechanical Engineering* 5 (2013). DOI: 10.1155/2013/969304. URL: https://journals.sagepub.com/doi/10.1155/2013/969304.

[14] Matthew S. Johannes et al. “An Overview of the Developmental Process for the Modular Prosthetic Limb”. In: *Johns Hopkins APL Technical Digest* 30 (2011). URL: https://secwww.jhuapl.edu/techdigest/content/techdigest/pdf/V30-N03/30-3-Johannes.pdf.

[15] Andrea Marinelli et al. “Active Upper Limb Prostheses: A Review on Current State and Upcoming Breakthroughs”. In: *Progress in Biomedical Engineering* 5 (2023). DOI: 10.1088/2516-1091/acac57. URL: https://iopscience.iop.org/article/10.1088/2516-1091/acac57/pdf.

[16] Álvaro Martín Martín. *Preprocessing of EMG Signals*. Bachelor’s Thesis. Madrid, Spain. URL: https://repositorio.comillas.edu/xmlui/handle/11531/68375.

[17] NXP Semiconductors. *TJA1050 High-Speed CAN Transceiver Datasheet*. Version Rev. 4.0. Accessed: 2026-04-15. NXP Semiconductors. 2003. URL: https://www.nxp.com/docs/en/data-sheet/TJA1050.pdf.

[18] Kolade Alexander Paul-Ajuwape. *Design and Manufacturing of a Lead Screw Robotic Gripper*. Bachelor’s Thesis. Cambridge, MA, USA, 2022. URL: https://studylib.net/doc/28156193/gripper.

[19] Alok Prakash, Bindu Kumari, and Shiru Sharma. “A Low-Cost, Wearable sEMG Sensor for Upper Limb Prosthetic Application”. In: *Journal of Medical Engineering & Technology* 43 (2019). DOI: 10.1080/03091902.2019.1653391. URL: https://www.researchgate.net/publication/335191421_A_low-cost_wearable_sEMG_sensor_for_upper_limb_prosthetic_application.

[20] Raspberry Pi Ltd. *Raspberry Pi Pico W Datasheet*. Accessed: 2026-04-04. 2022. URL: https://pip-assets.raspberrypi.com/categories/686-raspberry-pi-pico-w/documents/RP-008312-DS-1-pico-w-datasheet.pdf.

[21] Linda Resnik, Shana L. Klinger, and Katherine Etter. “The DEKA Arm: Its Features, Functionality, and Evolution During the Veterans Affairs Study to Optimize the DEKA Arm”. In: *Prosthetics and Orthotics International* 38 (). DOI: 10.1177/0309364613506913. URL: https://pubmed.ncbi.nlm.nih.gov/24150930/.

[22] Christian Schlette et al. “Concepts of a Modular System Architecture for Distributed Robotic Systems”. In: *Computers* 8 (2019). DOI: 10.3390/computers8010025. URL: https://www.mdpi.com/2073-431X/8/1/25.

[23] Ji Hyeon Shin et al. “A Universal Soft Gripper with the Optimized Fin Ray Finger”. In: *International Journal of Precision Engineering and Manufacturing-Green Technology* 8 (2021). DOI: 10.1007/s40684-021-00348-1. URL: https://link.springer.com/article/10.1007/s40684-021-00348-1.

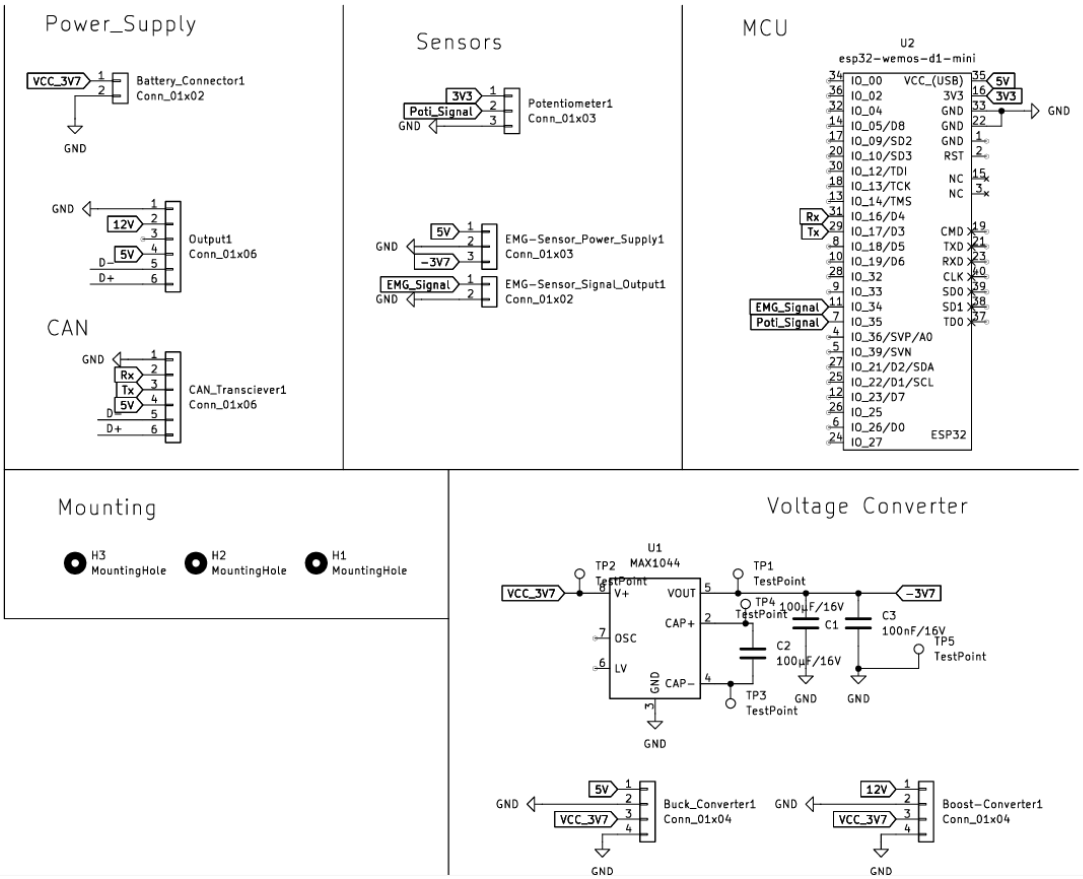
[24] Tim Stilson. *Potentiometers*. 1996. URL: https://soundlab.cs.princeton.edu/learning/tutorials/sensors/node10.html.

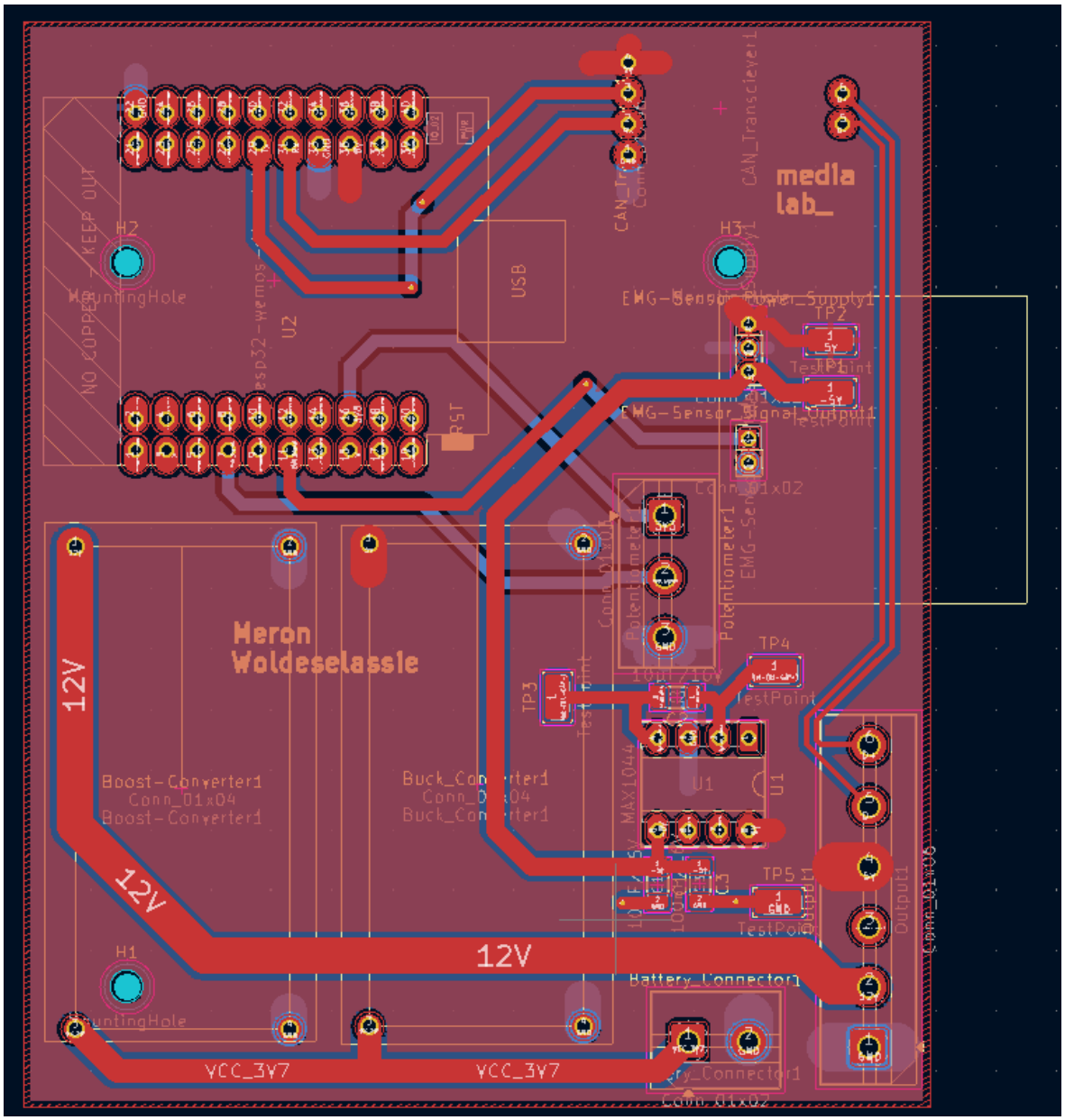
[25] David A. Winter. *Biomechanics and Motor Control of Human Movement*. 4th ed. Hoboken, NJ, USA: John Wiley & Sons, 2009. ISBN: 9780470398180. DOI: 10.1002/9780470549148. URL: https://books.google.com/books?id=_bFHL08IWfwC.

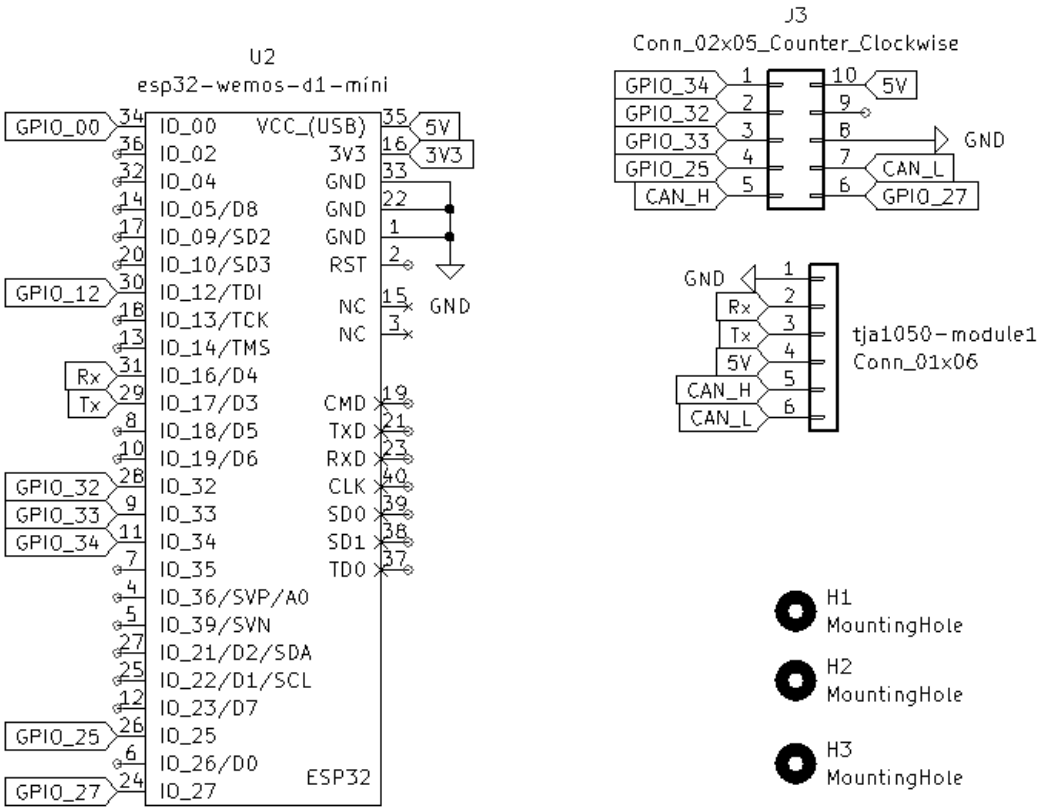
[26] Yan Zhang, Ning Xi, and Robert G. Fenton. “Distributed Autonomous Control of Modular Robot Systems Using Parallel Programming”. In: *Robotics and Autonomous Systems* 45 (2003). DOI: 10.1016/S0921-8890(03)00138-7. URL: https://www.sciencedirect.com/science/article/pii/S0924013603002887.

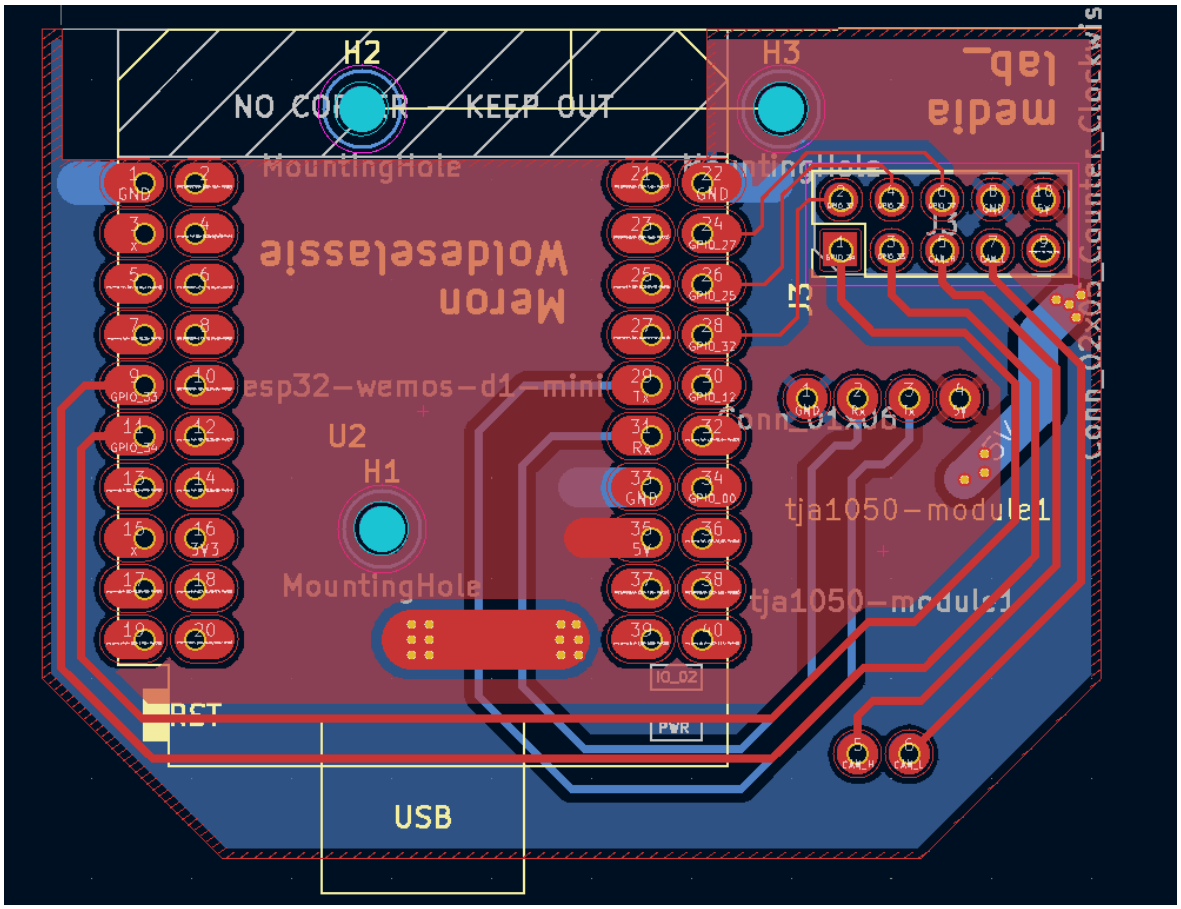
13 Appendix

13.1 Electronic

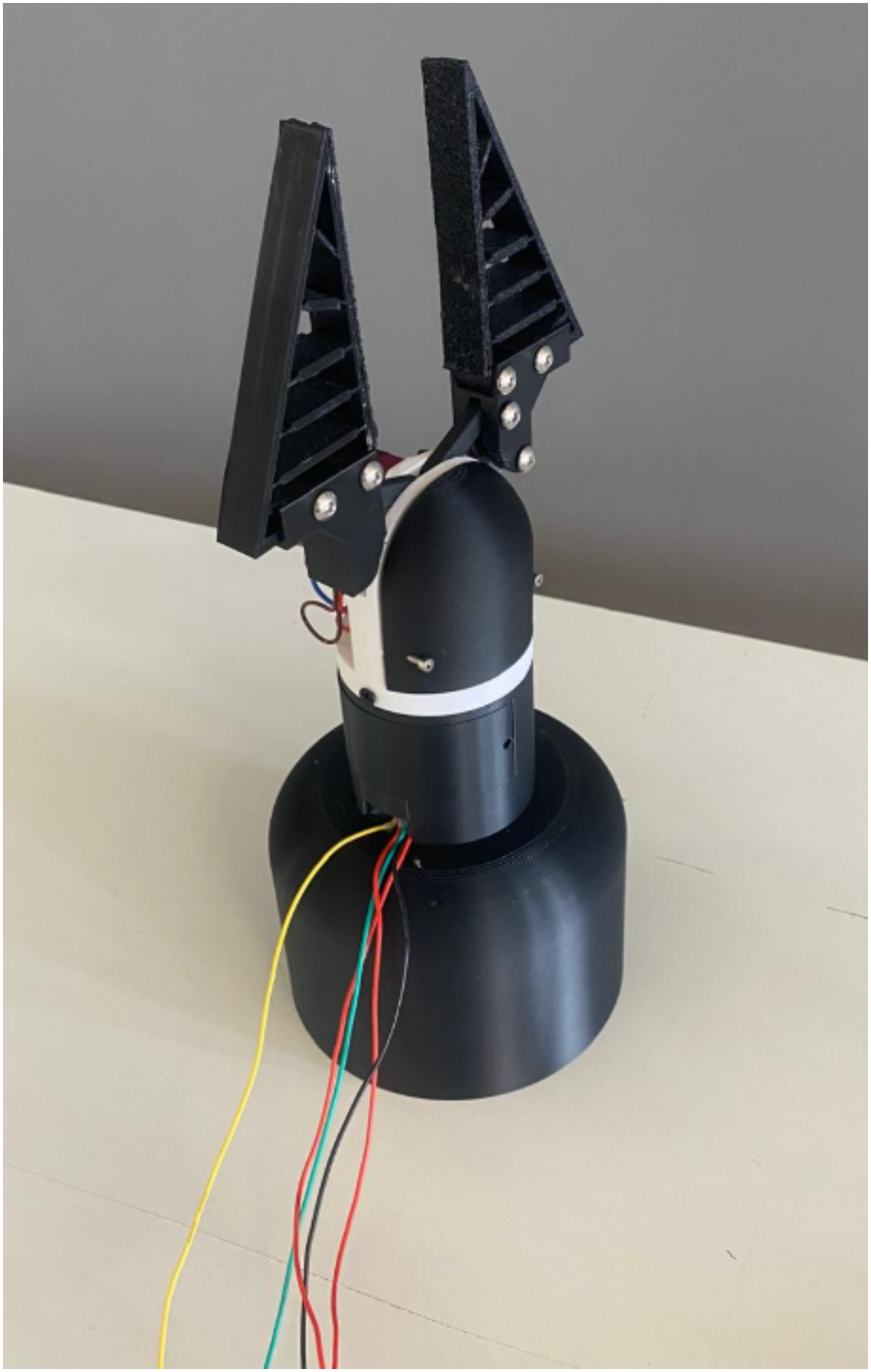


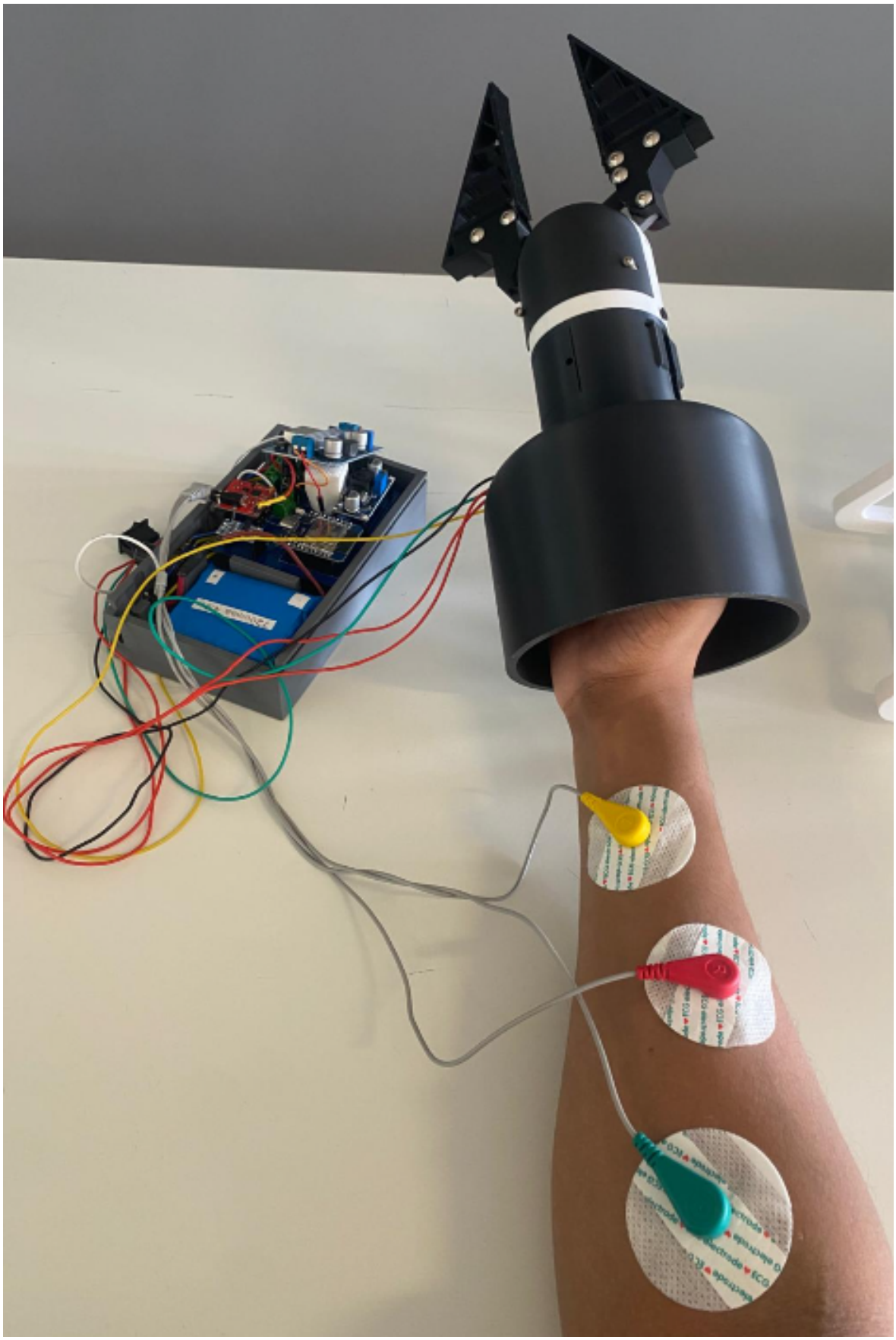

media lab_
Meron Woldeselassie
12V
12V
12V
VCC_3V7
VCC_3V7
USB
RST
H1
H2
H3
TP1
TP2
TP3
TP4
TP5
U1
U2
Battery_Connector1
Boost-Converter1
Buck_Converter1
Potentiometer1
EMG-Sensor_Signal_Output1
CAN_Transceiver1
MountingHole
TestPoint


U2
esp32-wemos-d1-mini
GPIO_00
IO_00
VCC_(USB)
5V
IO_02
3V3
3V3
IO_04
GND
IO_05/D8
GND
IO_09/SD2
GND
IO_10/SD3
RST
GPIO_12
IO_12/TDI
NC
GND
IO_13/TCK
NC
IO_14/TMS
Rx
IO_16/D4
Tx
IO_17/D3
CMD
IO_18/D5
TXD
IO_19/D6
RXD
GPIO_32
IO_32
CLK
GPIO_33
IO_33
SD0
GPIO_34
IO_34
SD1
IO_35
TD0
IO_36/SVP/A0
IO_39/SVN
IO_21/D2/SDA
IO_22/D1/SCL
IO_23/D7
GPIO_25
IO_25
IO_26/D0
ESP32
GPIO_27
IO_27
J3
Conn_02x05_Counter_Clockwise
GPIO_34
GPIO_32
GPIO_33
GPIO_25
CAN_H
5V
GND
CAN_L
GPIO_27
GND
Rx
Tx
5V
CAN_H
CAN_L
tja1050-module1
Conn_01x06
H1
MountingHole
H2
MountingHole
H3
MountingHole


H2
KEEP OUT
H3
media lab
MountingHole
Meron Woldeselassie
esp32-wemos-d1-mini
U2
H1
MountingHole
tja1050-module1
USB
RST
PWR

13.2 Overall System





13.3 Software

```
1   /**
2    * @file master_main.cpp
3    * @brief Control software for the master unit.
4    *
5    * The master unit manages communication with connected gadgets via CAN
     ↪ bus.
6    * It reads EMG and potentiometer signals, loads configuration data from
     ↪ XML,
7    * and transmits control data to the active gadget.
8    *
9    * The system supports modular prosthetic components such as a rotation
     ↪ module
10   * and a gripper module.
11   */
12  #include <Arduino.h>
13  #include <LittleFS.h>
14  #include <tinyxml2.h>
15  #include <ESP32-TWAI-CAN.hpp>
16
18  #define ANNOUNCE_ID 0x01
19  #define HEARTBEAT_ID 0x02
20
21
22  using namespace tinyxml2;
23
24  /**
25   * @struct MasterConfig
26   * @brief Stores the hardware and communication configuration of the
     ↪ master unit.
27   *
28   * This structure contains all configuration parameters required by the
     ↪ master
29   * ESP32, including EMG input, potentiometer input, CAN bus pins and
     ↪ timing values.
30   */
31  struct MasterConfig {
32      /** @brief Analog input pin connected to the EMG sensor signal. */
33      int emgPin;
34      /** @brief Threshold value for EMG signal activation. */
35      int emgThreshold;
36      /** @brief Analog input pin connected to the potentiometer signal. */
37      int potSignalPin;
38      int potPowerPin;//
39      // CAN
40      /** @brief CAN transmit pin of the master ESP32. */
41      int canTxPin;
42      /** @brief CAN receive pin of the master ESP32. */
43      int canRxPin;
44      /** @brief Interval in milliseconds for heartbeat communication. */
45      int heartbeatIntervalMs;
```

```
46          /** @brief Timeout in milliseconds after which a gadget is considered
    ↪  disconnected. */
47          int timeoutMs;
48          /** @brief CAN bus baud rate loaded from the XML configuration. */
49          int baudRate;
50      };
51
52      /**
53      * @struct Gadget1Config
54      * @brief Configuration for the rotation module (Gadget 1).
55      *
56      * This structure contains all parameters required to control a DC motor
57      * with encoder feedback, including pin assignments, scaling parameters,
58      * PID controller values and safety limits.
59      */
60      struct Gadget1Config {
61          /** @brief CAN transmit pin: Not inuse right now-> Tx pin is
    ↪ Hardcoded in Gadget1 */
62          int canTxPin;
63          /** @brief CAN receive pin: Not inuse right now-> Rx pin is Hardcoded
    ↪  in Gadget1 */
64          int canRxPin;
65          // Moter + encoder pins
66          /** @brief Encoder channel A pin. */
67          int encoderPinA;
68          /** @brief Encoder channel B pin. */
69          int encoderPinB;
70          /** @brief Motor control pin 1 (direction/PWM). */
71          int motorIn1Pin;
72          /** @brief Motor control pin 2 (direction/PWM). */
73          int motorIn2Pin;
74          // Motor characteristics
75          /** @brief Number of encoder pulses per motor revolution. */
76          int pulsesPerRevolution;
77          /** @brief Gear ratio of the motor. */
78          int gearRatio;
79          // Setpoint scaling
80          /** @brief Minimum input value for linear scaling for Gadget1. */
81          int inputMin;
82          /** @brief Maximum input value for linear scaling for Gadget1. */
83          int inputMax;
84          /** @brief Minimum output value for linear scaling for Gadget1. */
85          float outputMin;
86          /** @brief Maximum output value for linear scaling for Gadget1. */
87          float outputMax;
88          // PID
89          /** @brief Proportional gain for the PID controller. Implemented but
    ↪ not yet used. */
90          float kp;
91          /** @brief Integral gain for the PID controller. Implemented but not
    ↪ yet used. */
92          float ki;
```

```
93          /** @brief Derivative gain for the PID controller. Implemented but
        ↪ not yet used. */
94          float kd;
95          /** @brief Maximum output value for the PID controller. Implemented
        ↪ but not yet used. */
96          int maxOutput;
97          /** @brief Sample time for the PID controller in milliseconds.
        ↪ Implemented but not yet used. */
98          int sampleTimeMs;
99          /** @brief Maximum RPM for the motor. Implemented but not yet used.
        ↪ */
100         float maxRpm;
101     };
102
103     /**
104     * @struct Gadget2Config
105     * @brief Configuration for the gripper module (Gadget 2).
106     *
107     * This structure contains all parameters required to control a servo-
        ↪ based
108     * gripper module. It includes hardware pin assignments, input/output
        ↪ scaling
109     * parameters, PID controller settings and safety limits.
110     */
111     struct Gadget2Config {
112         /** @brief CAN transmit pin: Not inuse right now-> Tx pin is
        ↪ Hardcoded*/
113         int canTxPin;
114         /** @brief CAN receive pin: Not inuse right now-> Rx pin is Hardcoded
        ↪ */
115         int canRxPin;
116         // Pins
117         /** @brief Analog input pin for sensor signal (e.g. pressure sensor).
        ↪  Implemented but not yet used. */
118         int sensorPin;
119         /** @brief PWM output pin for servo control.*/
120         int servoPwmPin;
121
122         // Setpoint scaling
123         /** @brief Minimum input value for linear scaling for Gadget2. */
124         int inputMin;
125         /** @brief Maximum input value for linear scaling for Gadget2. */
126         int inputMax;
127         /** @brief Minimum output value for linear scaling for Gadget2. */
128         float outputMin;
129         /** @brief Maximum output value for linear scaling for Gadget2. */
130         float outputMax;
131
132         // PID
133         /** @brief Proportional gain for the PID controller. Implemented but
        ↪ not yet used. */
134         float kp;
```

```
135         /** @brief Integral gain for the PID controller. Implemented but not
        ↪ yet used. */
136         float ki;
137         /** @brief Derivative gain for the PID controller. Implemented but
        ↪ not yet used. */
138         float kd;
139         /** @brief Maximum output value for the PID controller. Implemented
        ↪ but not yet used. */
140         int maxOutput;
141         /** @brief Sample time for the PID controller in milliseconds.
        ↪ Implemented but not yet used. */
142         int sampleTimeMs;
143         // Safety
144         /** @brief Maximum value for the sensor input to prevent unsafe
        ↪ operation. Implemented but not yet used. */
145         float maxValue;
146     };
147
148     /**
149     * @struct CanIds
150     * @brief Stores all CAN message identifiers used in the system.
151     *
152     * These IDs define the communication protocol between the master and
        ↪ gadgets.
153     */
154     struct CanIds {
155         /** @brief CAN ID used by gadgets to announce themselves. */
156         int gadgetAnnounce;
157         /** @brief CAN ID used to send configuration to Gadget 1. */
158         int gadget1Config;
159         /** @brief CAN ID used to send configuration to Gadget 2. */
160         int gadget2Config;
161         /** @brief CAN ID used to transmit potentiometer data. */
162         int potiData;
163         /** @brief CAN ID used to transmit EMG sensor data. */
164         int emgData;
165         /** @brief CAN ID used for heartbeat messages. */
166         int heartbeat;
167     };
168
169     /** @brief Configuration data for the master unit loaded from the XML
        ↪ file. */
170     MasterConfig masterConfig;
171     /** @brief Configuration data for the rotation module. */
172     Gadget1Config gadget1Config;
173     /** @brief Configuration data for the gripper module. */
174     Gadget2Config gadget2Config;
175     /** @brief CAN message identifiers used by the system. */
176     CanIds canIds;
177     /** @brief ID of the currently connected gadget. Zero means no gadget is
        ↪ active. */
178     int activeGadgetId = 0;
179     /** @brief Timestamp of the last received heartbeat message. */
```

```
180     unsigned long lastHeartbeatReceived = 0;
181
182
183     /**
184     * @brief Initializes the LittleFS file system.
185     *
186     * This function mounts the LittleFS file system. If mounting fails, it
        ↪ attempts
187     * to format the file system and mount it again.
188     *
189     * @return true if the file system was mounted successfully, false
        ↪ otherwise.
190     */
191     bool initFilesystem() {
192         if (!LittleFS.begin(false)) {
193             Serial.println("[FS] Mount failed, formatting...");
194             if (!LittleFS.begin(true)) {
195                 Serial.println("[FS] Format failed!");
196                 return false;
197             }
198         }
199         Serial.println("[FS] Mounted successfully");
200         return true;
201     }
202
203     /**
204     * @brief Loads and parses the XML configuration file.
205     *
206     * This function reads the XML configuration file from LittleFS and
        ↪ extracts
207     * the master configuration, gadget configurations and CAN message IDs.
208     *
209     * @param path Path to the XML configuration file.
210     * @return true if the configuration was loaded successfully, false
        ↪ otherwise.
211     */
212     bool loadConfig(const char* path) {
213         File file = LittleFS.open(path, "r");
214         if (!file) {
215             Serial.println("[XML] Failed to open config file");
216             return false;
217         }
218
219         size_t fileSize = file.size();
220         char* buffer = new char[fileSize + 1];
221         file.readBytes(buffer, fileSize);
222         buffer[fileSize] = '\0';
223         file.close();
224
225         XMLDocument doc;
226         if (doc.Parse(buffer) != XML_SUCCESS) {
227             Serial.println("[XML] Parse error!");
228             delete[] buffer;
```

```
229             return false;
230         }
231
232         // Navigate to: config  master
233         XMLElement* root   = doc.FirstChildElement("config");
234         XMLElement* master = root->FirstChildElement("master");
235
236         // emg
237         XMLElement* emg = master->FirstChildElement("emg");
238         masterConfig.emgPin       = emg->FirstChildElement("pin_signal")->
    ↪ IntText();
239         masterConfig.emgThreshold = emg->FirstChildElement("threshold")->
    ↪ IntText();
240
241         // potentiometer
242         XMLElement* pot = master->FirstChildElement("potentiometer");
243         masterConfig.potSignalPin = pot->FirstChildElement("pin_signal")->
    ↪ IntText();
244         masterConfig.potPowerPin  = pot->FirstChildElement("pin_power")->
    ↪ IntText();
245
246         // can
247         XMLElement* can = master->FirstChildElement("can");
248         masterConfig.canTxPin            = can->FirstChildElement("pin_tx")->
    ↪ IntText();
249         masterConfig.canRxPin            = can->FirstChildElement("pin_rx")->
    ↪ IntText();
250         masterConfig.baudRate            = can->FirstChildElement("baud_rate"
    ↪ )->IntText();
251         masterConfig.heartbeatIntervalMs = can->FirstChildElement("
    ↪ heartbeat_interval_ms")->IntText();
252         masterConfig.timeoutMs           = can->FirstChildElement("timeout_ms
    ↪ ")->IntText();
253
254         // Gadget 1
255         XMLElement* gadget1 = root->FirstChildElement("gadget");
256         if (!gadget1) { Serial.println("[XML] ERROR: Gadget 1 not found!");
    ↪ return false; }
257         Serial.println("[XML] Gadget 1 found");
258
259         // Gadget 2
260         XMLElement* gadget2 = gadget1->NextSiblingElement("gadget");
261         if (!gadget2) { Serial.println("[XML] ERROR: Gadget 2 not found!");
    ↪ return false; }
262         Serial.println("[XML] Gadget 2 found");
263
264         //  Gadget 1 data
265         XMLElement* can1 = gadget1->FirstChildElement("can");
266         gadget1Config.canTxPin = can1->FirstChildElement("pin_tx")->IntText()
    ↪ ;
267         gadget1Config.canRxPin = can1->FirstChildElement("pin_rx")->IntText()
    ↪ ;
268
```

```
269         XMLElement* pins1 = gadget1->FirstChildElement("pins");
270         gadget1Config.encoderPinA = pins1->FirstChildElement("encoder_a")->
    ↪ IntText();
271         gadget1Config.encoderPinB = pins1->FirstChildElement("encoder_b")->
    ↪ IntText();
272         gadget1Config.motorIn1Pin = pins1->FirstChildElement("motor_int1")->
    ↪ IntText();
273         gadget1Config.motorIn2Pin = pins1->FirstChildElement("motor_int2")->
    ↪ IntText();
274
275         XMLElement* setpoint1 = gadget1->FirstChildElement("setpoint");
276         gadget1Config.inputMin  = setpoint1->FirstChildElement("input_min")->
    ↪ IntText();
277         gadget1Config.inputMax  = setpoint1->FirstChildElement("input_max")->
    ↪ IntText();
278         gadget1Config.outputMin = setpoint1->FirstChildElement("output_min")
    ↪ ->FloatText();
279         gadget1Config.outputMax = setpoint1->FirstChildElement("output_max")
    ↪ ->FloatText();
280
281         XMLElement* encoder1 = gadget1->FirstChildElement("encoder");
282         gadget1Config.pulsesPerRevolution = encoder1->FirstChildElement("
    ↪ pulses_per_revolution")->IntText();
283         gadget1Config.gearRatio           = encoder1->FirstChildElement("
    ↪ gear_ratio")->IntText();
284
285         XMLElement* pid1 = gadget1->FirstChildElement("pid");
286         gadget1Config.kp           = pid1->FirstChildElement("kp")->FloatText
    ↪ ();
287         gadget1Config.ki           = pid1->FirstChildElement("ki")->FloatText
    ↪ ();
288         gadget1Config.kd           = pid1->FirstChildElement("kd")->FloatText
    ↪ ();
289         gadget1Config.maxOutput    = pid1->FirstChildElement("max_output")->
    ↪ IntText();
290         gadget1Config.sampleTimeMs = pid1->FirstChildElement("sample_time_ms"
    ↪ )->IntText();
291
292         XMLElement* safety1 = gadget1->FirstChildElement("safety");
293         gadget1Config.maxRpm = safety1->FirstChildElement("max_rpm")->
    ↪ FloatText();
294
295         Serial.println("[XML] Gadget 1 config loaded");
296
297         //  Gadget 2 data
298         XMLElement* can2 = gadget2->FirstChildElement("can");
299         gadget2Config.canTxPin = can2->FirstChildElement("pin_tx")->IntText()
    ↪ ;
300         gadget2Config.canRxPin = can2->FirstChildElement("pin_rx")->IntText()
    ↪ ;
301
302         XMLElement* pins2 = gadget2->FirstChildElement("pins");
```

```
303         gadget2Config.sensorPin   = pins2->FirstChildElement("sensor_signal")
    ↪ ->IntText();
304         gadget2Config.servoPwmPin = pins2->FirstChildElement("servo_pwm")->
    ↪ IntText();
305
306         XMLElement* setpoint2 = gadget2->FirstChildElement("setpoint");
307         gadget2Config.inputMin  = setpoint2->FirstChildElement("input_min")->
    ↪ IntText();
308         gadget2Config.inputMax  = setpoint2->FirstChildElement("input_max")->
    ↪ IntText();
309         gadget2Config.outputMin = setpoint2->FirstChildElement("output_min")
    ↪ ->FloatText();
310         gadget2Config.outputMax = setpoint2->FirstChildElement("output_max")
    ↪ ->FloatText();
311
312         XMLElement* pid2 = gadget2->FirstChildElement("pid");
313         gadget2Config.kp          = pid2->FirstChildElement("kp")->FloatText
    ↪ ();
314         gadget2Config.ki          = pid2->FirstChildElement("ki")->FloatText
    ↪ ();
315         gadget2Config.kd          = pid2->FirstChildElement("kd")->FloatText
    ↪ ();
316         gadget2Config.maxOutput   = pid2->FirstChildElement("max_output")->
    ↪ IntText();
317         gadget2Config.sampleTimeMs = pid2->FirstChildElement("sample_time_ms"
    ↪ )->IntText();
318
319         XMLElement* safety2 = gadget2->FirstChildElement("safety");
320         gadget2Config.maxValue = safety2->FirstChildElement("max_value")->
    ↪ FloatText();
321
322         Serial.println("[XML] Gadget 2 config loaded");
323
324         //  CAN IDs
325         XMLElement* can_ids = root->FirstChildElement("can_ids");
326         if (!can_ids) { Serial.println("[XML] ERROR: can_ids not found!");
    ↪ return false; }
327
328         canIds.gadgetAnnounce = can_ids->FirstChildElement("gadget_announce")
    ↪ ->IntText();
329         canIds.gadget1Config  = can_ids->FirstChildElement("gadget1_config")
    ↪ ->IntText();
330         canIds.gadget2Config  = can_ids->FirstChildElement("gadget2_config")
    ↪ ->IntText();
331         canIds.potiData       = can_ids->FirstChildElement("poti_data")->
    ↪ IntText();
332         canIds.emgData        = can_ids->FirstChildElement("emg_data")->
    ↪ IntText();
333         canIds.heartbeat      = can_ids->FirstChildElement("heartbeat")->
    ↪ IntText();
334
335         Serial.println("[XML] CAN IDs loaded");
336
```

```
337         delete[] buffer;
338         return true;
339     }
340 
341     /**
342     * @brief Initializes the CAN bus interface.
343     *
344     * Sets the CAN TX and RX pins according to the master configuration and
    ↪ starts
345     * CAN communication.
346     *
347     * @return true if CAN initialization was successful, false otherwise.
348     */
349     bool initCAN() {
350         ESP32Can.setPins(masterConfig.canTxPin, masterConfig.canRxPin);
351         ESP32Can.setSpeed(ESP32Can.convertSpeed(500));
352         if (!ESP32Can.begin()) {
353             Serial.println("[CAN] Initialization failed!");
354             return false;
355         }
356         Serial.println("[CAN] Initialized successfully");
357         return true;
358     }
359 
360     /**
361     * @brief Sends configuration data to Gadget 1 via CAN bus.
362     *
363     * The configuration is split into multiple CAN frames due to payload size
    ↪  limits.
364     * It includes pin assignments, encoder parameters, PID values and output
    ↪ limits.
365     */
366     void sendGadget1Config() {
367         CanFrame f;
368         f.identifier = canIds.gadget1Config;
369 
370         // Message 1: Pins
371         f.data_length_code = 4;
372         f.data[0] = gadget1Config.encoderPinA;
373         f.data[1] = gadget1Config.encoderPinB;
374         f.data[2] = gadget1Config.motorIn1Pin;
375         f.data[3] = gadget1Config.motorIn2Pin;
376         ESP32Can.writeFrame(f);
377         delay(10);
378 
379         // Message 2: Encoder
380         f.data_length_code = 2;
381         f.data[0] = gadget1Config.pulsesPerRevolution;
382         f.data[1] = gadget1Config.gearRatio;
383         ESP32Can.writeFrame(f);
384         delay(10);
385 
386         // Message 3: kp, ki
```

```
387         f.data_length_code = 8;
388         memcpy(&f.data[0], &gadget1Config.kp, 4);
389         memcpy(&f.data[4], &gadget1Config.ki, 4);
390         ESP32Can.writeFrame(f);
391         delay(10);
392 
393         // Message 4: kd, maxOutput
394         f.data_length_code = 8;
395         memcpy(&f.data[0], &gadget1Config.kd, 4);
396         memcpy(&f.data[4], &gadget1Config.maxOutput, 4);
397         ESP32Can.writeFrame(f);
398         delay(10);
399 
400         // Message 5: outputMin, outputMax
401         f.data_length_code = 8;
402         memcpy(&f.data[0], &gadget1Config.outputMin, 4);
403         memcpy(&f.data[4], &gadget1Config.outputMax, 4);
404         ESP32Can.writeFrame(f);
405 
406         Serial.println("motorIn1Pin: " + String(gadget1Config.motorIn1Pin) +
    ↪ ", motorIn2Pin: " + String(gadget1Config.motorIn2Pin));
407 
408         Serial.println("[CAN] Gadget 1 config sent");
409     }
410 
411 
412     /**
413     * @brief Sends configuration data to Gadget 2 via CAN bus.
414     *
415     * The configuration is transmitted in multiple CAN frames and includes
416     * pin assignments, PID parameters and output limits.
417     */
418     void sendGadget2Config() {
419         CanFrame f;
420         f.identifier = canIds.gadget2Config;
421 
422         // Message 1: Pins
423         f.data_length_code = 2;
424         f.data[0] = gadget2Config.sensorPin;
425         f.data[1] = gadget2Config.servoPwmPin;
426         ESP32Can.writeFrame(f);
427         delay(10);
428 
429         // Message 2: kp, ki
430         f.data_length_code = 8;
431         memcpy(&f.data[0], &gadget2Config.kp, 4);
432         memcpy(&f.data[4], &gadget2Config.ki, 4);
433         ESP32Can.writeFrame(f);
434         delay(10);
435 
436         // Message 3: kd, maxOutput
437         f.data_length_code = 8;
438         memcpy(&f.data[0], &gadget2Config.kd, 4);
```

```
439         memcpy(&f.data[4], &gadget2Config.maxOutput, 4);
440         ESP32Can.writeFrame(f);
441         delay(10);
442 
443         // Message 4: outputMin, outputMax
444         f.data_length_code = 8;
445         memcpy(&f.data[0], &gadget2Config.outputMin, 4);
446         memcpy(&f.data[4], &gadget2Config.outputMax, 4);
447         ESP32Can.writeFrame(f);
448 
449         Serial.println("[CAN] Gadget 2 config sent");
450     }
451 
452     /**
453     * @brief Initializes the master system.
454     *
455     * This function:
456     * - Starts serial communication
457     * - Mounts the file system
458     * - Loads configuration from XML
459     * - Initializes hardware pins
460     * - Starts CAN communication
461     */
462     void setup() {
463         Serial.begin(115200);
464         delay(3000);
465 
466         if (!initFilesystem()) return;
467         if (!loadConfig("/config.xml")) return;
468 
469         /*
470         // Init potentiometer power pin
471         pinMode(masterConfig.potPowerPin, OUTPUT);
472         digitalWrite(masterConfig.potPowerPin, HIGH);
473         */
474 
475         if (!initCAN()) return;
476     }
477 
478     /**
479     * @brief Main execution loop of the master.
480     *
481     * This function continuously:
482     * - Listens for CAN messages from gadgets
483     * - Handles gadget connection and configuration
484     * - Monitors heartbeat signals
485     * - Detects communication timeouts
486     * - Reads sensor values (EMG and potentiometer)
487     * - Sends sensor data to the active gadget
488     */
489     void loop() {
490         // CAN receive
491         CanFrame frame;
```

```
492         if (ESP32Can.readFrame(frame, 0)) {
493
494             // Gadget announces itself
495             if (frame.identifier == ANNOUNCE_ID) {
496                 int gadgetId = frame.data[0];
497                 Serial.print("[CAN] Gadget announced: ");
498                 Serial.println(gadgetId);
499
500                 if (gadgetId == 1) {
501                     activeGadgetId = 1;
502                     lastHeartbeatReceived = millis();
503                     CanFrame reply;
504                     reply.identifier       = ANNOUNCE_ID;
505                     reply.data_length_code = 3;
506                     reply.data[0] = canIds.potiData;
507                     reply.data[1] = canIds.heartbeat;
508                     reply.data[2] = canIds.gadget1Config;
509                     ESP32Can.writeFrame(reply);
510                     sendGadget1Config();
511                     Serial.println("[CAN] Gadget 1 connected and configured!"
    ↪ );
512
513                 } else if (gadgetId == 2) {
514                     activeGadgetId = 2;
515                     lastHeartbeatReceived = millis();
516                     CanFrame reply;
517                     reply.identifier       = ANNOUNCE_ID;
518                     reply.data_length_code = 3;
519                     reply.data[0] = canIds.potiData;
520                     reply.data[1] = canIds.heartbeat;
521                     reply.data[2] = canIds.gadget2Config;
522                     ESP32Can.writeFrame(reply);
523                     sendGadget2Config();
524                     Serial.println("[CAN] Gadget 2 connected and configured!"
    ↪ );
525                 }
526             }
527
528             // Heartbeat received
529             else if (frame.identifier == canIds.heartbeat) {
530                 lastHeartbeatReceived = millis();
531             }
532         }
533
534         // Timeout check
535         if (activeGadgetId != 0) {
536             if (millis() - lastHeartbeatReceived > masterConfig.timeoutMs) {
537                 Serial.println("[CAN] Gadget disconnected - timeout!");
538                 activeGadgetId = 0;
539                 lastHeartbeatReceived = millis();
540             }
541         }
542
```

```
543         // Send sensor data (only when gadget connected)
544         if (activeGadgetId != 0) {
545 
546             // Read sensors
547             int rawPoti = analogRead(masterConfig.potSignalPin);
548             int rawEmg  = analogRead(masterConfig.emgPin);
549 
550             // Scale poti to output range based on active gadget
551             float potiValue = 0.0f;
552 
553             if (activeGadgetId == 1) {
554                 potiValue = gadget1Config.outputMin +
555                 (float)(rawPoti - gadget1Config.inputMin) *
556                 (gadget1Config.outputMax - gadget1Config.outputMin) /
557                 (gadget1Config.inputMax - gadget1Config.inputMin);
558             } else if (activeGadgetId == 2) {
559                 potiValue = gadget2Config.outputMin +
560                 (float)(rawPoti - gadget2Config.inputMin) *
561                 (gadget2Config.outputMax - gadget2Config.outputMin) /
562                 (gadget2Config.inputMax - gadget2Config.inputMin);
563             }
564 
565 
566 
567             // Send poti data
568             CanFrame fPoti;
569             fPoti.identifier       = canIds.potiData;
570             fPoti.data_length_code = 4;
571             memcpy(&fPoti.data[0], &potiValue, 4);
572             ESP32Can.writeFrame(fPoti);
573 
574             // Send emg data (2 Byte: high byte + low byte)
575             CanFrame fEmg;
576             fEmg.identifier       = canIds.emgData;
577             fEmg.data_length_code = 2;
578             fEmg.data[0] = (rawEmg >> 8) & 0xFF;
579             fEmg.data[1] = rawEmg & 0xFF;
580             ESP32Can.writeFrame(fEmg);
581         }
582         delay(10);
583     }
```

Listing 13.1: Master Main.cpp code

```
1     /**
2     * @file slave1/slave1_main.cpp
3     * @brief Control software for Gadget 1 (rotation module).
4     *
5     * This module controls a DC motor with encoder feedback and provides
6     * rotational movement for the prosthetic hand.
7     *
8     * The module communicates with the master unit via CAN bus. It receives
9     * configuration data, potentiometer values and EMG signals.
10    *
```

```
11      * The EMG signal is used as an activation input, while the potentiometer
12      * controls direction and speed of rotation.
13      *
14      * The structure of the software supports future implementation of
15      * closed-loop control (e.g. PID), although currently a simple PWM-based
16      * control strategy is used.
17      */
18
19      #include <Arduino.h>
20      #include <ESP32-TWAI-CAN.hpp>
21
22      /** @brief Defines the ID for the slave 1 module */
23      #define SLAVE_ID      1
24      /** @brief CAN TX pin hardcoded */
25      #define CAN_TX_PIN    17
26      /** @brief CAN RX pin hardcoded */
27      #define CAN_RX_PIN    16
28      /** @brief CAN announcement ID */
29      #define ANNOUNCE_ID   0x01
30      /** @brief CAN EMG data ID */
31      #define EMG_DATA_ID   0x40
32      /** @brief Hardcoded EMG activation threshold */
33      #define EMG_THRESHOLD 1500
34
35      /**
36      * @struct SlaveCanIds
37      * @brief Stores CAN message identifiers assigned by the master.
38      */
39      struct SlaveCanIds {
40          /** @brief CAN ID used to receive potentiometer data. */
41          int potiDataId;
42          /** @brief CAN ID used for heartbeat communication. */
43          int heartbeatId;
44          /** @brief CAN ID used to receive configuration data. */
45          int configId;
46      };
47
48      /** @brief Configuration structure for Slave 1. */
49      struct Slave1Config {
50          /** @brief Encoder channel A pin used for speed measurement. */
51          int encoderPinA;
52          /** @brief Encoder channel B pin used for direction detection.(left/
        ↪ right)*/
53          int encoderPinB;
54          /** @brief Motor control pin 1 (direction/PWM). */
55          int motorIn1Pin;
56          /** @brief Motor control pin 2 (direction/PWM). */
57          int motorIn2Pin;
58          /** @brief Number of encoder pulses per motor revolution. */
59          int pulsesPerRevolution;
60          /** @brief Gear ratio of the DC motor. */
61          int gearRatio;
62          /** @brief Proportional gain for the PID controller. not used yet */
```

```
63          float kp;
64          /** @brief Integral gain for the PID controller. not used yet */
65          float ki;
66          /** @brief Derivative gain for the PID controller. not used yet */
67          float kd;
68          /** @brief Maximum output value for the PID controller. not used yet
        ↪ */
69          int maxOutput;
70          /** @brief Minimum output value for linear scaling for Gadget1. */
71          float outputMin;
72          /** @brief Maximum output value for linear scaling for Gadget1. */
73          float outputMax;
74      };
75
76      /** @brief Stores CAN message identifiers assigned by the master. */
77      SlaveCanIds slaveCanIds;
78      /** @brief Configuration structure for Slave 1. */
79      Slave1Config slave1Config;
80
81      /** @brief Current potentiometer value. */
82      float currentPotiValue      = 0.0f;
83      /** @brief Current EMG value. */
84      int currentEmgValue         = 0;
85      /** @brief Current RPM value. */
86      float currentRpm            = 0.0f;
87      /** @brief Volatile encoder count for interrupt handling.  */
88      volatile long encoderCount  = 0;
89      /** @brief Last time RPM was calculated. */
90      unsigned long lastRpmCalc   = 0;
91      /** @brief Last time heartbeat was sent. */
92      unsigned long lastHeartbeat = 0;
93
94
95      /** @brief Interrupt service routine for encoder updates. */
96      void IRAM_ATTR encoderISR() {
97          if (digitalRead(slave1Config.encoderPinB) == HIGH) {
98              encoderCount++;
99          } else {
100             encoderCount--;
101         }
102     }
103
104     /** @brief Initializes the CAN bus. */
105     void initCAN() {
106         ESP32Can.setPins(CAN_TX_PIN, CAN_RX_PIN);
107         ESP32Can.setSpeed(ESP32Can.convertSpeed(500));
108         if (ESP32Can.begin()) {
109             Serial.println("[CAN] Initialized");
110         } else {
111             Serial.println("[CAN] Failed");
112         }
113     }
114
```

```
115     /** @brief Sends an announcement message to the master. When connected */
116     void announceToMaster() {
117         CanFrame frame;
118         frame.identifier       = ANNOUNCE_ID;
119         frame.data_length_code = 1;
120         frame.data[0]          = SLAVE_ID;
121         ESP32Can.writeFrame(frame);
122         Serial.println("[CAN] Announced to master");
123     }
124 
125     /** @brief Waits for CAN message identifiers from the master. */
126     bool waitForCanIds() {
127         CanFrame response;
128         if (ESP32Can.readFrame(response, 2000)) {
129             if (response.identifier == ANNOUNCE_ID && response.data[0] !=
    ↪ SLAVE_ID) {
130                 slaveCanIds.potiDataId  = response.data[0];
131                 slaveCanIds.heartbeatId = response.data[1];
132                 slaveCanIds.configId    = response.data[2];
133                 Serial.print("[CAN] Poti Data ID: ");  Serial.println(
    ↪ slaveCanIds.potiDataId);
134                 Serial.print("[CAN] Heartbeat ID: ");  Serial.println(
    ↪ slaveCanIds.heartbeatId);
135                 Serial.print("[CAN] Config ID: ");     Serial.println(
    ↪ slaveCanIds.configId);
136                 return true;
137             }
138         }
139         Serial.println("[CAN] ERROR: No response from master!");
140         return false;
141     }
142 
143     /** @brief Receives configuration messages from the master. */
144     bool receiveConfig() {
145         CanFrame f;
146 
147         // Message 1: Pins
148         if (!ESP32Can.readFrame(f, 2000)) { Serial.println("[CAN] ERROR:
    ↪ Config msg 1 timeout"); return false; }
149         slave1Config.encoderPinA = f.data[0];
150         slave1Config.encoderPinB = f.data[1];
151         slave1Config.motorIn1Pin = f.data[2];
152         slave1Config.motorIn2Pin = f.data[3];
153         Serial.println("[CAN] Pins received");
154 
155         // Message 2: Encoder
156         if (!ESP32Can.readFrame(f, 2000)) { Serial.println("[CAN] ERROR:
    ↪ Config msg 2 timeout"); return false; }
157         slave1Config.pulsesPerRevolution = f.data[0];
158         slave1Config.gearRatio           = f.data[1];
159         Serial.println("[CAN] Encoder config received");
160 
161         // Message 3: kp, ki
```

```
162         if (!ESP32Can.readFrame(f, 2000)) { Serial.println("[CAN] ERROR:
    ↪ Config msg 3 timeout"); return false; }
163         memcpy(&slave1Config.kp, &f.data[0], 4);
164         memcpy(&slave1Config.ki, &f.data[4], 4);
165         Serial.println("[CAN] kp, ki received");
166
167         // Message 4: kd, maxOutput
168         if (!ESP32Can.readFrame(f, 2000)) { Serial.println("[CAN] ERROR:
    ↪ Config msg 4 timeout"); return false; }
169         memcpy(&slave1Config.kd,        &f.data[0], 4);
170         memcpy(&slave1Config.maxOutput, &f.data[4], 4);
171         Serial.println("[CAN] kd, maxOutput received");
172
173         // Message 5: outputMin, outputMax
174         if (!ESP32Can.readFrame(f, 2000)) { Serial.println("[CAN] ERROR:
    ↪ Config msg 5 timeout"); return false; }
175         memcpy(&slave1Config.outputMin, &f.data[0], 4);
176         memcpy(&slave1Config.outputMax, &f.data[4], 4);
177         Serial.println("[CAN] outputMin, outputMax received");
178
179         return true;
180     }
181
182     /** @brief Initializes the GPIO pins for the rotation module.
183     */
184     void initPins() {
185         pinMode(slave1Config.encoderPinA, INPUT_PULLUP);
186         pinMode(slave1Config.encoderPinB, INPUT_PULLUP);
187         pinMode(slave1Config.motorIn1Pin, OUTPUT);
188         pinMode(slave1Config.motorIn2Pin, OUTPUT);
189         attachInterrupt(digitalPinToInterrupt(slave1Config.encoderPinA),
    ↪ encoderISR, RISING);
190         Serial.println("[PIN] Pins initialized");
191     }
192
193     /** @brief Calculates the RPM based on encoder counts.
194     * This function is called every 20 ms to update the current RPM value.
195     * It reads the encoder count, resets it, and calculates the RPM based on
196     * the number of pulses received in the elapsed time, taking into account
197     * the pulses per revolution and gear ratio.
198     */
199     void calcRpm() {
200         unsigned long now     = millis();
201         unsigned long elapsed = now - lastRpmCalc;
202
203         if (elapsed >= 20) {
204             long count    = encoderCount;
205             encoderCount  = 0;
206             lastRpmCalc   = now;
207
208             float pulsesPerMinute = (float)count / elapsed * 60000.0f;
209             currentRpm = pulsesPerMinute /
210             (slave1Config.pulsesPerRevolution * slave1Config.gearRatio);
```

```
211
212             Serial.print("[ENC] RPM: ");
213             Serial.println(currentRpm);
214         }
215     }
216
217     /** @brief Controls the motor based on potentiometer and EMG values.
218     * Raw poti value is currentPotiValue (0-100 scaled from master
219     * Re-interpret as raw ADC 0-4095 for direction logic
220     * We use the raw poti from master which is already 0-100
221     * So we need to remap: 0-50 = left, 50-100 = right
222     */
223     void controlMotor() {
224         // EMG inactive: stop motor
225         /*if (currentEmgValue < EMG_THRESHOLD) {
226             analogWrite(slave1Config.motorIn1Pin, 0);
227             analogWrite(slave1Config.motorIn2Pin, 0);
228             return;
229         }*/
230
231         int pwm = 0;
232
233         if (currentPotiValue <= 50.0f) {
234             // Left: 0  fast, 50  slow
235             pwm = (int)((50.0f - currentPotiValue) / 50.0f * 255.0f);
236             pwm = constrain(pwm, 0, 255);
237             analogWrite(slave1Config.motorIn1Pin, pwm);
238             analogWrite(slave1Config.motorIn2Pin, 0);
239         } else {
240             // Right: 50  slow, 100  fast
241             pwm = (int)((currentPotiValue - 50.0f) / 50.0f * 255.0f);
242             pwm = constrain(pwm, 0, 255);
243             analogWrite(slave1Config.motorIn1Pin, 0);
244             analogWrite(slave1Config.motorIn2Pin, pwm);
245         }
246
247
248     }
249
250     /**
251     * @brief Initializes the rotation module.
252     *
253     * This function:
254     * 1. Initializes the serial communication for debugging.
255     * 2. Initializes the CAN bus and announces the slave to the master.
256     * 3. Waits for the master to send CAN message identifiers and
        ↪ configuration data.
257     * 4. Initializes the GPIO pins based on the received configuration.
258     */
259     void setup() {
260         Serial.begin(115200);
261         initCAN();
262         announceToMaster();
```

```
263         if (!waitForCanIds()) return;
264         if (!receiveConfig()) return;
265         initPins();
266     }
267
268
269     /** * @brief Main loop for the rotation module.
270     * This loop continuously:
271     *  1. Checks for incoming CAN messages and updates potentiometer and EMG
        ↪ values.
272     *  2. Calculates the current RPM based on encoder counts.
273     *  3. Controls the motor based on the potentiometer and EMG values.
274     *  4. Sends heartbeat messages to the master at regular intervals.
275     *
276     */
277     void loop() {
278         //  CAN receive
279         CanFrame frame;
280         if (ESP32Can.readFrame(frame, 0)) {
281
282             // Poti data received
283             if (frame.identifier == slaveCanIds.potiDataId) {
284                 memcpy(&currentPotiValue, &frame.data[0], 4);
285             }
286
287             // EMG data received
288             else if (frame.identifier == EMG_DATA_ID) {
289                 currentEmgValue = (frame.data[0] << 8) | frame.data[1];
290             }
291         }
292
293         //  RPM calculation
294         calcRpm();
295
296         //  Motor control
297         controlMotor();
298
299         //analogWrite(slave1Config.motorIn2Pin, 200);
300         //  Heartbeat senden
301         unsigned long now = millis();
302         if (now - lastHeartbeat >= 100) {
303             lastHeartbeat = now;
304             CanFrame f;
305             f.identifier       = slaveCanIds.heartbeatId;
306             f.data_length_code = 1;
307             f.data[0]          = SLAVE_ID;
308             ESP32Can.writeFrame(f);
309         }
310     }
311
312
```

Listing 13.2: Slave1 Rotation modul Main.cpp

```
1   /**
2   * @file slave2/slave2_main.cpp
3   * @brief Control software for Gadget 2 (gripper module).
4   *
5   * This module controls a servo-based gripper for the prosthetic hand.
6   *
7   * It communicates with the master via CAN bus, receives configuration
8   * data and user input signals (EMG and potentiometer), and controls
9   * the gripper accordingly.
10  *
11  * The EMG signal is used to open or close the gripper, enabling
12  * intuitive control based on muscle activity.
13  */
14
15  #include <Arduino.h>
16  #include <ESP32-TWAI-CAN.hpp>
17  #include <ESP32Servo.h>
18
19  /** @brief Defines the ID for the slave 2 module */
20  #define SLAVE_ID    2
21  /** @brief CAN TX pin hardcoded */
22  #define CAN_TX_PIN  17
23  /** @brief CAN RX pin hardcoded */
24  #define CAN_RX_PIN  16
25  /** @brief CAN announcement ID */
26  #define ANNOUNCE_ID 0x01
27  /** @brief CAN EMG data ID */
28  #define EMG_DATA_ID 0x40  // hardcoded, never changes
29  /** @brief Hardcoded EMG activation threshold */
30  #define   EMG_THRESHOLD 1500 // hardcoded EMG activation threshold
31
32  /**
33  * @struct SlaveCanIds
34  * @brief Stores CAN message identifiers assigned by the master.
35  */
36  struct SlaveCanIds {
37      /** @brief CAN ID used for receiving potentiometer data from master
    ↪ */
38      int potiDataId;
39      /** @brief CAN ID used for heartbeat communication */
40      int heartbeatId;
41      /** @brief CAN ID used for configuration message */
42      int configId;
43  };
44
45  /** @brief Struct for Slave 2 configuration */
46  struct Slave2Config {
47      /** @brief Pin for the pressure sensor */
48      int sensorPin;
49      /** @brief Pin for the servo PWM signal */
50      int servoPwmPin;
51      /** @brief Proportional gain for PID controller. not used yet */
52      float kp;
```

```
53          /** @brief Integral gain for PID controller. not used yet */
54          float ki;
55          /** @brief Derivative gain for PID controller. not used yet */
56          float kd;
57          /** @brief Maximum output value for PID controller. not used yet */
58          int maxOutput;
59          /** @brief Minimum output value for PID controller. not used yet */
60          float outputMin;
61          /** @brief Maximum output value for PID controller. not used yet */
62          float outputMax;
63      };
64
65      /** @brief Stores CAN message identifiers assigned by the master */
66      SlaveCanIds slaveCanIds;
67      /** @brief Configuration structure for Slave 2 */
68      Slave2Config slave2Config;
69
70      /** @brief Current potentiometer value */
71      float currentPotiValue = 0.0f;
72      /** @brief Current EMG value */
73      int currentEmgValue    = 0;
74      /** @brief Current servo position in degrees */
75      float currentServoPos  = 0.0f;
76
77      /** @brief Servo object */
78      Servo servo;
79
80      /** @brief Initializes the CAN bus */
81
82      void initCAN() {
83          ESP32Can.setPins(CAN_TX_PIN, CAN_RX_PIN);
84          ESP32Can.setSpeed(ESP32Can.convertSpeed(500));
85          if (ESP32Can.begin()) {
86              Serial.println("[CAN] Initialized");
87          } else {
88              Serial.println("[CAN] Failed");
89          }
90      }
91
92
93      /** @brief Sends an announcement message to the master */
94      void announceToMaster() {
95          CanFrame frame;
96          frame.identifier       = ANNOUNCE_ID;
97          frame.data_length_code = 1;
98          frame.data[0]          = SLAVE_ID;
99          ESP32Can.writeFrame(frame);
100         Serial.println("[CAN] Announced to master");
101     }
102
103     /** @brief Waits for CAN message identifiers from the master */
104     bool waitForCanIds() {
105         CanFrame response;
```

```
106         if (ESP32Can.readFrame(response, 2000)) {
107             if (response.identifier == ANNOUNCE_ID && response.data[0] !=
    ↪ SLAVE_ID) {
108                 slaveCanIds.potiDataId  = response.data[0];
109                 slaveCanIds.heartbeatId = response.data[1];
110                 slaveCanIds.configId    = response.data[2];
111                 Serial.print("[CAN] Poti Data ID: ");  Serial.println(
    ↪ slaveCanIds.potiDataId);
112                 Serial.print("[CAN] Heartbeat ID: ");  Serial.println(
    ↪ slaveCanIds.heartbeatId);
113                 Serial.print("[CAN] Config ID: ");     Serial.println(
    ↪ slaveCanIds.configId);
114                 return true;
115             }
116         }
117         Serial.println("[CAN] ERROR: No response from master!");
118         return false;
119     }
120
121     /** @brief Receives configuration data from the master */
122     bool receiveConfig() {
123         CanFrame f;
124
125         // Message 1: Pins
126         if (!ESP32Can.readFrame(f, 2000)) { Serial.println("[CAN] ERROR:
    ↪ Config msg 1 timeout"); return false; }
127         slave2Config.sensorPin   = f.data[0];
128         slave2Config.servoPwmPin = f.data[1];
129         Serial.println("[CAN] Pins received");
130
131         // Message 2: kp, ki
132         if (!ESP32Can.readFrame(f, 2000)) { Serial.println("[CAN] ERROR:
    ↪ Config msg 2 timeout"); return false; }
133         memcpy(&slave2Config.kp, &f.data[0], 4);
134         memcpy(&slave2Config.ki, &f.data[4], 4);
135         Serial.println("[CAN] kp, ki received");
136
137         // Message 3: kd, maxOutput
138         if (!ESP32Can.readFrame(f, 2000)) { Serial.println("[CAN] ERROR:
    ↪ Config msg 3 timeout"); return false; }
139         memcpy(&slave2Config.kd,        &f.data[0], 4);
140         memcpy(&slave2Config.maxOutput, &f.data[4], 4);
141         Serial.println("[CAN] kd, maxOutput received");
142
143         // Message 4: outputMin, outputMax
144         if (!ESP32Can.readFrame(f, 2000)) { Serial.println("[CAN] ERROR:
    ↪ Config msg 4 timeout"); return false; }
145         memcpy(&slave2Config.outputMin, &f.data[0], 4);
146         memcpy(&slave2Config.outputMax, &f.data[4], 4);
147         Serial.println("[CAN] outputMin, outputMax received");
148
149         return true;
150     }
```

```
151
152     /** @brief Initializes the GPIO pins for the Gripper Module */
153     void    initPins() {
154         pinMode(slave2Config.sensorPin, INPUT);
155         servo.attach(slave2Config.servoPwmPin);
156         servo.write(0);  // start position
157         currentServoPos = 0.0f;
158         Serial.println("[PIN] Pins initialized");
159     }
160
161     /** @brief Controls the servo motor based on EMG input
162     *   * This function reads the current EMG value and moves the servo to 90
        ↪  degrees if the EMG value exceeds a defined threshold, otherwise it
        ↪ moves it back to 0 degrees.
163     */
164     void controlServo() {
165         float targetPos = (currentEmgValue > EMG_THRESHOLD) ? 90.0f : 0.0f;
166         /*
167         float targetPos;
168         if(currentEmgValue > EMG_THRESHOLD)
169         {targetPos = 90.0f;
170             servo.write(90);
171             Serial.print("[SERVO] Moving "); Serial.print(90); Serial.println
        ↪ (" degrees");}
172         else    {targetPos = 0.0f;
173             servo.write(0);
174             Serial.print("[SERVO] Moving "); Serial.print(0); Serial.println
        ↪ (" degrees");}
175         */
176
177         // Move servo step by step towards target
178         if (currentServoPos < targetPos) {
179             currentServoPos += 1.0f;
180             if (currentServoPos > targetPos) currentServoPos = targetPos;
181         } else if (currentServoPos > targetPos) {
182             currentServoPos -= 1.0f;
183             if (currentServoPos < targetPos) currentServoPos = targetPos;
184         }
185
186         Serial.print("[SERVO] Moving "); Serial.print(currentServoPos);
        ↪ Serial.println(" degrees");
187         servo.write((int)currentServoPos);
188     }
189
190     /** @brief Sets up the gripper module
191     *  * This function:
192     *   1. Initializes the serial communication for debugging.
193     *   2. Initializes the CAN bus and announces the slave to the master.
194     *   3. Waits for the master to send CAN message identifiers and
        ↪ configuration data.
195     *   4. Initializes the GPIO pins based on the received configuration.
196     */
197     void setup() {
```

```
198         Serial.begin(115200);
199         delay(3000); // wait for serial monitor
200         initCAN();
201         announceToMaster();
202         if (!waitForCanIds()) return;
203         if (!receiveConfig()) return;
204         initPins();
205     }
206
207     /** @brief Last time heartbeat was sent. */
208     unsigned long lastHeartbeat   = 0;
209     /** @brief Last time servo was updated. */
210     unsigned long lastServoUpdate = 0;
211
212
213     /** @brief Main loop for the gripper module.
214     * This loop continuously:
215     *  1. Checks for incoming CAN messages and updates EMG values.
216     *  2. Controls the servo motor based on the EMG values.
217     *  3. Sends heartbeat messages to the master at regular intervals.
218     */
219     void loop() {
220         //  CAN receive
221         CanFrame frame;
222         if (ESP32Can.readFrame(frame, 0)) {
223
224             // Poti data received
225             if (frame.identifier == slaveCanIds.potiDataId) {
226                 memcpy(&currentPotiValue, &frame.data[0], 4);
227                 Serial.print("[CAN] Poti value received: ");
228                 Serial.println(currentPotiValue);
229             }
230
231             // EMG data received
232             else if (frame.identifier == EMG_DATA_ID) {
233                 currentEmgValue = (frame.data[0] << 8) | frame.data[1];
234                 Serial.print("[CAN] EMG value received: ");
235                 Serial.println(currentEmgValue);
236             }
237             Serial.print("      \r");
238
239         }
240
241
242         //  Servo control (every 20ms)
243         unsigned long now = millis();
244         if (now - lastServoUpdate >= 20) {
245             lastServoUpdate = now;
246             controlServo();
247         }
248         //controlServo();
249         //  Heartbeat senden
250         if (now - lastHeartbeat >= 100) {
```

```
251             lastHeartbeat = now;
252             CanFrame f;
253             f.identifier       = slaveCanIds.heartbeatId;
254             f.data_length_code = 1;
255             f.data[0]          = SLAVE_ID;
256             ESP32Can.writeFrame(f);
257         }
258     }
```

Listing 13.3: Slave 1 Gripper Main.cpp