CHALMERS

Designing Next-Gen Adaptive Exoskeleton for Physiotherapy and Rehabilitation

Bachelor's thesis in EENX16

Adam Hansson, Albin Hedberg, Emil Klasson,
Isabella Lago, Malte Lindeberg, Viktor Hjalmarson

DEPARTMENT OF ELECTRICAL ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2025
www.chalmers.se

Bachelor's thesis 2025

Designing Next-Gen Adaptive Exoskeleton for Physiotherapy and Rehabilitation

EENX16, Project group: EENX16-VT25-90

Adam Hansson, Albin Hedberg, Emil Klasson,
Isabella Lago, Malte Lindeberg, Viktor Hjalmarson

CHALMERS

Department of Electrical Engineering
Division of Systems and Control
Chalmers University of Technology
Gothenburg, Sweden 2025

Designing Next-Gen Adaptive Exoskeleton for Physiotherapy and Rehabilitation
EENX16, Project group: EENX16-VT25-90
Adam Hansson, Albin Hedberg, Emil Klasson,
Isabella Lago, Malte Lindeberg, Viktor Hjalmarson

Supervisor: Fabian Just, Electrical Engineering
Examiner: Emmanuel Dean, Electrical Engineering

Bachelor's Thesis 2025
Department of Electrical Engineering
Division of Systems and Control
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2025

Abstract

Robotic knee exoskeletons have emerged as promising tools for enhancing gait training, with the potential to accelerate rehabilitation and aid the therapists throughout the entire process. However, many current solutions are limited by bulky designs, lack of adaptability, and poor real-world usability. This report presents the development of a modular, motorized and instrumented knee exoskeleton prototype designed to be lightweight, user-friendly, and interchangeable between the left and right leg. The design incorporates a quick-release actuator mechanism, integrated force sensors, an IMU and an encoder, laying the groundwork for more accessible and practical assistive devices.

Sammandrag

Robothjälpmedel i form av motoriserade knäortroser har visat sig vara lovande verktyg för att förbättra gångträning, med potential att både påskynda rehabiliteringsprocessen och underlätta fysioterapeuters arbete. Samtidigt är många av dagens lösningar begränsade av skrymmande konstruktioner, bristande anpassningsförmåga och låg användbarhet i verkliga miljöer. Denna rapport presenterar utvecklingen av en modulär, motoriserad och instrumenterad prototyp av en knäortros, framtagen för att vara lättviktig, användarvänlig och utbytbar mellan vänster och höger ben. Designen inkluderar en snabbkopplingsmekanism för aktuatorn, integrerade kraftsensorer, en IMU samt en vinkelgivare, vilket tillsammans utgör grunden för mer tillgängliga och praktiskt användbara hjälpmedel inom rehabilitering.

Acknowledgements

We would like to express our sincere gratitude to our supervisor, Fabian Just, for his invaluable guidance, support, and insightful feedback throughout the project. We would also like to thank our examiner, Emmanuel Dean, for his helpful insights and critical review during the course of the project.

Special thanks to Yuhong Zhou, Domenico Caliandro, and Pengcheng Shen, whose helpful assistance and input were greatly appreciated, and to Gunilla Kjellby Wendt and Jon Karlsson for the valuable feedback during our demonstration of the prototype. Finally, we would like to thank the Department of Electrical Engineering at Chalmers University of Technology for making this project possible by providing resources and space.

Adam Hansson, Albin Hedberg, Emil Klasson, Isabella Lago
Malte Lindeberg, Viktor Hjalmarson
Gothenburg, May 2025

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

ADC	Analog-to-Digital Converter
ACL	Anterior Cruciate Ligament
API	Application Programming Interface
CAD	Computer Aided Design
CAN	Controller Area Network
FDM	Fused Deposition Modeling
HAT	Hardware Attached on Top
I^2C	Inter-Integrated Circuit
IMU	Inertial Measurement Unit
PCB	Printed Circuit Board
PETG	Polyethylene Terephthalate Glycol
PLA	Polyactic Acid
ROS	Robot Operating System
SCL	Serial Clock Line
SDA	Serial Data Line
SDK	Software Development Kit
SLS	Selective Laser Sintering
SPI	Serial Peripheral Interface
UART	Universal Asynchronous Receiver / Transmitter
UPS	Uninterruptible Power Supply

Contents

List of Figures

1

Introduction

The following chapter presents the clinical and technical context for the project, defines its specific purpose, and outlines the limitations within the scope of the project.

1.1 Background

Knee injuries represent a significant clinical and societal concern due to their high prevalence and the impact they have on mobility, quality of life, and healthcare resources [1]. These injuries can result from a variety of causes, including trauma, degenerative joint diseases, neurological conditions, and high-impact physical activity. Among them, injuries to the Anterior Cruciate Ligament (ACL) are particularly common and often require surgical intervention followed by extensive rehabilitation. In Sweden alone, approximately 6,000 ACL injuries occur annually, equivalent to approximately 80 cases per 100,000 inhabitants [2]. While knee injuries affect individuals of all ages, young female athletes are especially vulnerable. For instance, in football, knee injuries account for 30% of all injuries in men and about 48% in women [3]. Although knee injuries are common for both men and women in sports, women have a higher risk ratio [4]. Understanding the frequency and distribution of knee injuries is crucial for developing effective strategies for prevention, treatment, and long-term support. To support rehabilitation and mobility, orthoses are widely used to stabilize and assist joint function.

Traditional orthoses are passive and mainly provide structural support without actively aiding movement. In recent years, active orthoses, also known as motorized or powered exoskeletons, have gained attention for their ability to assist mobility and potentially accelerate the rehabilitation process [5][6]. These systems also enable data collection during use, providing insights into patient performance, tracking therapy progress, and supporting data-driven clinical decisions. However, most active orthoses are still in the prototype stage and present several limitations. Notably, they are often bulky, lack portability, and are typically designed specifically for either the left or right leg, making them non-interchangeable [1]. This lack of modularity limits their practicality in real-world, daily use and increases production complexity and cost.

Given these challenges, there is a clear need for more adaptable and user-friendly solutions. Innovations that focus on these practical challenges could greatly improve the everyday functionality and acceptance of such devices. This context sets the stage for a targeted design approach aimed at creating more effective assistive solutions.

1.2 Purpose

This project aims to develop a motorized, instrumented and modular exoskeleton design that retains the functionality of an existing knee orthosis on which it is based. The design should be interchangeable between the left and right leg and fit 95% of the population. The project also focuses on implementing a user-friendly quick-release actuator mechanism and integrating force sensors, an IMU and a rotary encoder for data collection.

1.3 Limitations

This project is specifically focused on developing the motorized functionality of a knee exoskeleton for use in rehabilitation exercises. Although such a device could potentially assist with daily activities like walking, these applications fall outside the scope of the current design and intended use.

The prototype will not undergo any physical testing, including user trials, structural strength calculations, mechanical simulations, or durability assessments. Consequently, the structural integrity, safety, and real-world performance of the device remain unverified. Furthermore, the prototype is built upon a pre-existing orthosis, the X-ROM from Enovis [7]. This introduced constraints in terms of design flexibility, material selection, and overall adaptability. As a result, the potential for full optimization and customization was significantly reduced.

Since the project has not incorporated user-centered design methods; no input from patients, therapists, or other end-users has been gathered during development. This raises concerns about the device's ergonomic suitability and practical usability in clinical or home settings. The orthosis may not be tested on people when the actuator is turned on the feedback data from the actuator and force sensors cannot be properly evaluated or calibrated.

It is also important to note that this project represents one component of a larger initiative aimed at developing control systems and interactive rehabilitation software, such as therapeutic games. This specific effort focuses solely on the mechanical design and the integration of electrical components, sensors and an actuator. Additionally, the use of the Robot Operating System 2 (ROS 2) as the software architecture was determined in consultation with the project supervisor and was therefore not subject to evaluation or replacement. This decision ensured consistency with related parallel projects but also introduced a set of design constraints

related to software structure and component compatibility.

1.3.1 Adjustments to Project Scope

The project initially aimed to develop a system, versatile enough to support upper and lower limb rehabilitation, offering functionality for both arms and legs. As development progressed, it became clear that this broader scope would have compromised the overall quality, reliability, and feasibility of the design within the available timeframe. After discussions with the project supervisor, the scope was refined to focus solely on bilateral leg support. This allowed for a more targeted and technically robust implementation, though it limited the generalization of the final prototype.

2

Theory

This section outlines the key theoretical principles behind the design and background information on some of the components used.

2.1 Anthropometry

Anthropometry is the measurement of the human body to determine the average and statistical variations in size and weight of body parts [8]. Anthropometry is important to ensure the design accommodates a wide range of individuals.

The knee orthosis is made to fit people of stature ranging from 4'11" (1499mm) to 6'5" (1956mm) [7]. To fit 95% of the population the orthosis needs to fit all those within 2 standard deviations from the mean, which in this case would range from 1538mm to 1932mm [9]. Since the orthosis is made to fit a wider range than 95% of people, no change regarding length adjustability is necessary.

2.2 Components

An actuator is a device that converts an input signal into mechanical motion [10]. The input signal can be a number of sources though it is usually electrical and the type of mechanical motion may be rotational, pushing or pulling.

An Inertial Measurement Unit (IMU) is an electrical component that provides data on a body's orientation, position and angular speed [11]. The IMU can provide such data due to built in accelerometers, gyroscopes, and sometimes magnetometers.

An encoder, or a rotary encoder is an electrical device that converts the angular position or velocity of a rotating axle into an analog or digital output signal [12]. Rotary encoders may rely on phenomena such as optics, magnetic fields, mechanical contact or electric capacitance to determine the angular properties of the axle relative to the encoder.

A force sensor is a sensor that measures a mechanical force, such as tension, compression, pressure or torque and outputs an electrical signal depending on the magnitude

of the force applied [13]. Force sensors usually use materials whose electrical qualities change depending on pressure/strain, but may also rely on magnetic field variations or light intensity.

3

System Architecture and Components

This section presents the hardware and software components used in the development of the prototype. It outlines the selection of key technologies, including sensors, microcontrollers, and processing units, as well as the software framework used to support system functionality and communication.

3.1 Actuator

The actuator used in this project is the RMD-X6 S2, a compact brushless DC servo motor with an integrated controller [14]. It operates within a voltage range of 24–48 V and provides a nominal torque of 18 $N \cdot m$. The motor delivers a nominal speed of 70 rpm and a rated power of 132 W, with an efficiency of approximately 70%. The back drive torque is 0.91 $N \cdot m$, allowing for passive movement under certain conditions. It includes a 36:1 gear reduction and weighs 590 g. The actuator is designed to function in environments ranging from –20°C to 55°C, making it well-suited for typical indoor rehabilitation settings.

3.2 Sensors

Besides the sensoric functionality integrated in the motor, the orthosis incorporates three additional types of sensors: an IMU, a rotary encoder, and force sensors. These provide data on orientation, joint angles, and applied forces, enabling motion tracking and system evaluation. The following subsections detail the selected components and their functions.

3.2.1 IMU

The IMU used in this project is an Adafruit BNO085 [15]. The BNO085 is a 9-axis sensor that integrates a 3-axis accelerometer, 3-axis gyroscope, and 3-axis magnetometer. It features built-in sensor fusion algorithms that output absolute orientation, linear acceleration, rotation vector, and other motion-related data without

requiring external processing. The BNO085 uses Inter-Integrated Circuit (I^2C) to communicate.

3.2.2 Encoder

The Encoder used in this project is an AMT22 [16]. More specifically the encoder is an AMT223B which means it has an axial orientation and an absolute resolution of 14-bit single-turn. Single-turn meaning that the angle only goes from 0-359 degrees and then starts over rather than counting multiple rotations. The AMT223B communicates through Serial Peripheral Interface (SPI).

3.2.3 Force Sensors

The Force sensors used in this project are Flexiforce by Tekscan [17]. The Flexiforce sensors work by having two layers of conductive silver separated by a layer of Pressure-sensitive ink. The Pressure-sensitive ink changes its conductivity/resistance based on the force applied. The relation between the force applied and the sensor's conductivity is linear which makes for an easy conversion from output voltage to force applied.

Since the output from the Flexiforce sensors is analog, an Analog-to-Digital Converter (ADC) is necessary. The ADC used in this project is an ADS1015 [18]. The ADS1015 is a 12-bit ADC with capabilities of sampling at 3300 Hz and 4 single-ended inputs. The ADS1015 also has integrated operational amplifiers to help stabilize input signals. The ADS1015 communicates through I^2C.

3.3 Microcontroller

The microcontroller board used in this project is a Raspberry Pi Pico 2. The Pico 2 uses the microcontroller RP2350 which has a Dual Cortex-M33 processor, 520 kB SRAM and 4MB flash memory [19], thus it is able to store a little bit of data. The Pico 2 can be powered between 1.8V-5.5V and is able to receive and send data to components through different kinds of connections such as USB, I^2C and SPI.

3.4 Raspberry Pi 5

A Raspberry Pi 5 was used in this project and is a single board computer. The Raspberry Pi 5 has a 2.4GHz 64-bit quad-core processor and 4 GB SDRAM [20]. There is a slot for a microSD card to store and transfer data. It can send 5V DC of power through an USB connection.

3.5 PiJuice UPS HAT

The PiJuice UPS HAT used in this project is designed to work as an uninterruptible power solution for a Raspberry Pi computer. The PiJuice UPS HAT has a 1820 mAh

battery, with the option of replacing it with another lithium-ion or lithium-polymer battery [21].

3.6 ROS 2

The software for the system is built using the ROS 2 framework as the underlying software architecture. ROS is an open source Software Development Kit (SDK) designed to build modular and distributed robot applications [22]. ROS 2 is the successor to ROS 1, developed to address modern requirements in robotics such as real-time communication, enhanced security and improved scalability [23].

To understand how ROS 2 was used in the project, it is necessary to understand the terminology and the ROS 2 functionalities that were used. The implementation relied on ROS 2 concepts such as nodes, topics with publishers and subscribers, services, and parameters. They were all leveraged to enable modularity, communication, configuration and control within the system. Each of these concepts is described in more detail in the following paragraphs.

3.6.1 Nodes

A ROS 2 system typically consists of several nodes, that are communicating with each other. Each node should have a clearly defined function, facilitating modularity and separation of tasks within the system architecture [24].

3.6.2 Topics, publishers and subscribers

Topics act as communication channels through which nodes exchange messages [25]. A node can publish data to any number of topics and simultaneously subscribe to any number of topics, which allows for the creation of modular and scalable communication networks. A ROS 2 node uses publishers and subscribers for data exchange. This communication method served as the primary means of acquiring sensor data in the exoskeleton system.

3.6.3 Services

Services provide a synchronized communication mechanism between nodes, following a request-response model [26]. Unlike topics, services do not provide continuous data flow. Instead, they are used when a client explicitly sends a request to the server, which performs a specific action and returns a response. In this project, services were used primarily to send control commands to the motor and to apply hardware configuration changes, such as resetting sensor values.

3.6.4 Parameters

Parameters are configuration values that are associated with individual nodes and are used to adjust node behavior at startup or runtime without modifying the actual

source code [27]. In this project, parameters were used to adjust the sampling rate, effectively determining how often the publishers transmitted data.

3.7 Micro-ROS

Micro-ROS extends ROS 2 capabilities to microcontrollers, enabling resource-constrained devices to function as nodes within a ROS 2 system [28]. To achieve this, micro-ROS uses a lightweight communication middleware that allows seamless integration with larger ROS 2 systems, and provides a client Application Programming Interface (API) specifically optimized for microcontrollers. In this project, micro-ROS is used to extend ROS 2 capabilities to the Pico 2 microcontroller and to enable its integration within the ROS 2 system.

4

Method

To better understand and get a grasp of the project, it was divided up into subproblems. These subproblems were then divided across the members of the group according to expertise. The subproblems addressed the biggest technological challenges such as developing a removable actuator system, developing an enclosed design for the electronics, integrating sensors and setting up the exoskeleton for future control algorithms.

4.1 Removable Actuator System

The two major areas to the removable actuator system were the construction and manufacturing of the different components.

4.1.1 Construction and CAD

The mechanical design process of the project started with dividing the subproblem into concrete problems that needed to be solved. The four problems were:

- Selecting an actuator
- Design a mechanism for connecting the axle of the motor to the axle of the orthosis.
- Design a mounting plate for the motor to attach to.
- Design an attachment mechanism for the motor to attach to the mounting plate.

After concretizing the problem, the group started free idea generation, where the group members freely proposed ideas to solve the different problems. These sessions used creative thinking and helped enable the development of many different solutions. After that, the ideas were discussed and evaluated based on feasibility, availability, complexity, and compatibility with the already existing orthosis. After careful evaluation and discussion, most of the ideas were deemed impractical or not

possible with the scope of the project. There was one concept that remained which was then developed further.

The remaining concept was then drawn in a drawing program to show the different components and how they are meant to interact. That drawing was then translated into a CAD model using Inventor and CATIA V5. This allowed the team to visualize the construction and verify the mechanical compatibility with the original orthosis. Several iterations were made to the construction to optimize dimensions, tolerances, the mounting and coupling mechanics.

4.1.2 Manufacturing

The components of the design were manufactured with a mix of traditional machining and 3D-printing. The attachment plate used to secure the motor to the orthosis on one leg, was made in a workshop at Chalmers using equipment like a mill, lathe and pillar drill. This took time, but made it possible to have tight tolerances. The corresponding plate for the other leg was manufactured using water jet cutting, even though water jetting usually has worse tolerances. Critical features such as holes were intentionally made smaller to later be drilled to the correct dimension, to ensure a proper fit.

In addition to the metal components, several plastic components were also made using 3D-printing. These components include a housing for the motor and new angle limiters to fit the design. PLA and PETG were the materials used, chosen due to their ease of use and mechanical properties. 3D-printing allowed for rapid iteration of the design, which allowed the team to quickly test and update the design and reprint the new parts to ensure proper functionality and interaction with the other components.

4.2 Electronics and Sensors

This section describes the electrical sensor system designed for the exoskeleton. When designing the electrical system, the primary objective was to provide as much value as possible to the physiotherapist, with minimal compromise to the user experience. To achieve this, delivering relevant sensor data and useful functionalities, while ensuring that all components integrated well with the mechanical design, was considered a key priority. The development of the electrical sensor system was organized into four main components. These components are described in the following subsections.

4.2.1 Sensor Testing and Data Preprocessing

Initially each sensor was tested separately by connecting only that sensor to a Pico 2 and testing the data acquisition. When adequate software was written for each sensor and the data acquisition was effective and stable for each sensor the software

was integrated into one program. This program would gather data from each sensor at a given interval and display the data as comma-separated values.

4.2.2 Sensor placement and integration

The force sensors were placed by the lowest cuff of the orthosis, one on the tibia and one on the calf. Each individual force sensor was connected both to ground through a pull-down resistor of 100 kΩ and to the ADS1015 to get a accurate reference. The raw output of the ADS1015 was implemented in a linear function to convert the ADC values into the applied force.

The IMU was placed on the upper part of the orthosis (on the thigh), to monitor the angle of the entire leg relative to the ground. The encoder was placed on the joint of the orthosis since it needs to rotate around a fixed axis and monitors the angle of the joint. The encoder was placed on the inside of the orthosis to not interfere with the actuator and output an angle between 0 and 359 degrees. Together the IMU and encoder provides a complete understanding regarding the position of the leg.

4.2.3 Electronics enclosure

The Housing for the electronics were divided into a larger easily detachable part and a smaller enclosure, onto which the bigger part attaches. The larger enclosure contains the Raspberry Pi 5 and the battery. The smaller enclosure contains the Pico 2, the ADS1015, the BNO085 and most of the wiring.

This housing was placed over the uppermost cuff of the orthosis and attached by clamping it in place using 2 screws. The larger housing containing the Raspberry Pi 5 was ultimately placed above the smaller housing rather than building it on top to avoid excess bulkiness.

In order to have a snug fit with access to necessary connections and buttons the Raspberry Pi 5 was analysed both in a 3D CAD environment [29], but also by using calipers and the physical Raspberry Pi 5.

All housings were designed using CAD, and multiple iterations of 3D-printing were completed to fine-tune the fit and function. The smaller and larger housings included holes made to fit nuts, allowing various components or 3D-printed parts to be securely fastened with screws.

To measure angle of the knee the encoder was fastened to the orthosis by a concept developed by another student, Yuhong Zhou. With his permission, his concept was adapted in CAD to fit the chosen encoder and construction, then 3D printed.

4.2.4 Cable Management

To support a modular design and enable easy removal of all electronic components, the cables were routed along the outside of the orthosis and secured using adhesive

cable clips. Since the orthosis length is adjustable in two sections, the cables in those areas were enclosed in expandable braided sleeving to accommodate movement and reduce strain. For additional cable management, heat shrink tubing was used, particularly around the rotary joint and to secure the ends of the braided sleeving. To fasten the exposed force sensors to the adjustable straps, hook-and-loop tape was applied.

4.3 Software

This section describes the software architecture developed for the exoskeleton system. The overall software design was centered around modularity and seamless integration between embedded hardware, motor control and the user interface. The system was built using the Robot Operating System (ROS 2) framework as the core software infrastructure. Open-source libraries and frameworks were utilized to accelerate development to ensure compatibility with existing tools. The software stack was divided into three main components, which were the motor control layer, the embedded sensor layer and the web-based user interface.

4.3.1 Motor Control

To support ROS 2 Jazzy integration, Ubuntu 24.04 was installed on the Raspberry Pi 5. A ROS 2 workspace was built, with application-specific packages optimized for the exoskeleton system.

The motor unit used in the project provides a set of predefined control commands that enable interaction with the motor via Controller Area Network (CAN) communication. To verify the proper functionality, several of these commands were tested prior to the software implementation. The testing was performed via the Linux terminal using its integrated CAN tools.

After verifying the motor's functionality, a motor node was created with the purpose of handling the CAN-communication between the motor and the Raspberry Pi 5. The node was implemented with Python, to allow for rapid development. The myactuator_rmd SDK was used, which provided a high-level interface for sending and receiving CAN messages to and from the motor [30]. The implementation also included exception handling and a thread lock to safely manage simultaneous access attempts to the CAN bus.

A publisher for motor status data and several services for sending control commands to the motor was created in the node. In the callbacks of these components, functions from the myactuator_rmd SDK were leveraged to send the appropriate CAN messages, receive responses from the motor, and translate them into readable data. Additionally, a parameter defining the publish rate was declared to allow convenient adjustment of the data sampling rate at runtime. Custom ROS 2 message interfaces were designed and integrated into the node components to meet the requirements of the system.

4.3.2 Sensor Data Acquisition

The external sensors, located outside of the motor unit, were connected to the Pico 2 microcontroller. The software on the Pico 2 was built using C++ with the Arduino framework, with PlatformIO as the development environment. The Arduino framework was chosen due to its convenience and the availability of well-maintained driver libraries for the sensors used in the system.

To enable ROS 2 communication on resource-constrained embedded hardware, micro-ROS was utilized on the microcontroller. Its client API was leveraged to bring ROS 2 capabilities to the microcontroller. A ROS 2 node was developed for the Pico 2 to handle sensor data publishing and service requests. Publishers responsible for transmitting sensor data was implemented for each sensor in the system. Two services were also implemented, to support sensor calibration and adjustment. In the same way as for the motor node, a parameter was defined to facilitate runtime tuning of the data publish rate.

To improve data usability and reduce noise, the quaternion output from the IMU was converted to Euler angles (yaw, pitch, roll). Also, a sliding average filter was applied to the force sensor data and updated at each publishing instance to smooth out signal fluctuations.

Communication between the microcontroller and the main ROS 2 system was established by running a micro-ROS agent on the Raspberry Pi 5. The agent was set up to exchange data with the microcontroller via USB serial. To enable the microcontroller to automatically detect and re-establish communication with the micro-ROS agent when the link was unavailable, a reconnection loop was implemented in the software.

4.3.3 Web-based User Interface

To facilitate real-time monitoring and motor control, a web interface was developed as part of the system. This interface was designed to provide an intuitive tool to visualize live sensor data and to issue control commands to the motor through the underlying ROS 2 infrastructure.

The web application was developed using HTML, CSS and Javascript as the underlying programming languages. To facilitate rapid development and consistent design, the CSS framework Bootstrap 5 and the Javascript framework VueJS were utilized. The web interface was then hosted locally on the Raspberry Pi 5 using the Python-based web framework Flask. A Wi-Fi hotspot was configured on the Raspberry Pi 5 to enable wireless access to the web application from external devices.

The integration between the web application and the ROS 2 backend was achieved using the rosbridge WebSocket server, a component of the rosbridge_suite package [31]. On the frontend, the roslibjs JavaScript library was used to communicate with the server, enabling real-time data publishing, subscription, and service calls di-

rectly from the web interface. This setup was then utilized to implement interactive charts for data visualization, as well as buttons for sending control commands to the actuator.

The web interface consisted of three separate pages, each serving a specific purpose. The first page was designed to provide a real-time dashboard for monitoring live sensor data and controlling the exoskeleton. The second page presented a historical dashboard for visualizing and analyzing previously collected data. The third page offered settings and calibration functionality, allowing users to adjust parameters and perform system calibration.

Backend integration was completed for all pages except the historical dashboard. This was due to time constraints and a deliberate decision to postpone the implementation of data storage, as it was not yet required during active operation of the system. However, the system architecture was designed to allow for integration of this functionality at a later stage.

5

Results

In this section the final results of the mechanical, electrical and software solutions are presented.

5.1 Construction

The final construction integrates the original orthosis with a custom motor housing, a steel attachment plate, and embedded sensors.

5.1.1 Attachment Plate

The attachment plate is the connection point for the motor and its housing to attach to the orthosis. The attachment plate not only provided structural support but also featured a cutout that allowed the motor's output axle to pass through and connect directly to the mobile segment of the orthosis, specifically the lower leg (tibia) rail. A CAD picture of the attachment plate is presented in Figure 5.1

To improve stability and prevent the motor housing from rotating during use, alignment holes and pegs (M3 screws) were implemented. The alignment holes in the attachment plate are positioned to match the screws from the motor housing. This makes the motor housing unable to rotate by itself. On the bottom of the attachment plate, two 4 mm pegs of steal protrude down into their respective holes in the plastic cover of the orthosis making it impossible for the attachment plate to rotate. A 5 mm hole in the center allows the axle from the encoder to pass through the attachment plate, enabling joint movement and holding everything together. The attachment plate is made out of 2 mm steel, which gives it high strength and stiffness.

Additionally, grooves were cut into the plate to serve as anchoring points for the eccentric locks, ensuring a secure and precise fit. The two grooves were bended downwards to secure the eccentric locks. These elements together ensured that the plate remained fixed in position even when exposed to torsional forces from the motor.

A recess, visible in Figure 5.1, was made to replicate the original orthosis's locking mechanism in the straight position. By doing this the function is not hindered. A detailed CATIA drawing of the attachment plate is provided in Appendix A.1 for reference.

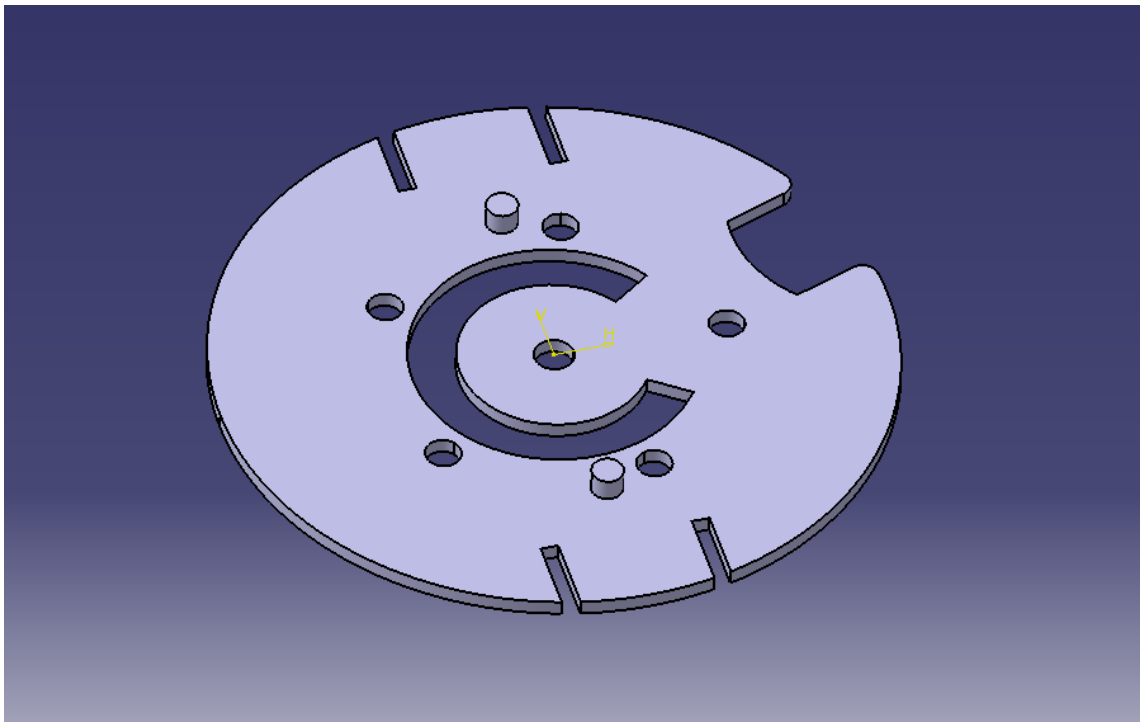


Figure 5.1: Attachment plate CAD.

5.1.2 Motor Housing

To enable proper mounting of the motor to the orthosis, a custom motor housing was also developed in CAD and 3D-printed with PETG. This housing was directly fastened to the motor casing and served as the interface to the attachment plate.

The chosen attachment mechanism was two Wiberger AB eccentric locks made of galvanized steel [32]. They were permanently mounted along the sides of the housings with screws, allowing the motor assembly to be securely fastened to the attachment plate. The housing included a cutout that aligned with the cutout in the attachment plate, allowing the rotor axle to connect to the tibia rail. The cutout goes 290 degrees around the center, allowing the rotor axle to move the tibia rail.

In order to create a flat and perpendicular surface for mounting the attachment plate, the plastic cover of the orthosis, which the plate was mounted against, had to be manually ground down. The corresponding cutout was milled into the cover to allow access for the motor axle. Additional anti-rotation holes were drilled to accommodate the stabilizing bolts from the attachment plate.

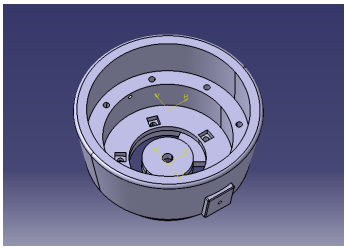
It also incorporated five non-threaded M3 bolts, which were inserted into corre-

sponding holes in the attachment plate, providing passive fixation to further resist rotational movement during operation.

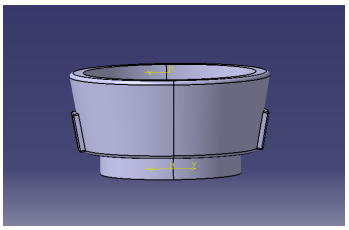
The motor housing includes eight holes designed for M3 screws that secure the housing to the motor. Additionally, the housing is recessed at the bottom, allowing visibility for potential future engravings of angle settings. This design enables users to easily change angle settings without detaching the motor. There is also an indent in the center at the bottom for the screw head from the center axle to fit.

For increased safety, edge fillets were applied to the motor housing in the CAD model to eliminate sharp corners. A 1.5 mm fillet was used along the top and bottom edges of the housing, and a 0.5 mm fillet was applied to the side plates where the eccentric locks are mounted.

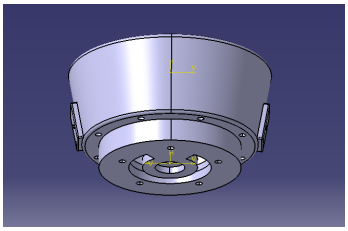
CAD pictures of the motor housing are presented in Figure 5.2a to 5.2c and a detailed CATIA drawing of the motor housing is provided in Appendix A.2 for reference.



(a) Tilted front view.



(b) Side view.



(c) Tilted back view.

Figure 5.2: Three perspectives of the motor housing CAD design.

5.1.3 Angle Limiters

In the original orthosis, the two angle limiters were positioned on either side of the lower leg rail. This meant that there was an angle limiter in the way of the motor axle connecting to the rail. In order for the motor axle to get a clear path into the lower leg rail, the angle limiters had to be redesigned.

The solution was to put both angle limiters under the rail, see Figure 5.3. In order to have the same springiness to the limiter, the same shape of the spring mechanism was used as the original. The height that the limiters sit on had to be changed to accommodate the new position.

The handle also had to be changed as the attachment plate wouldn't fit with the original limiters. The new limiters no longer wrap around the upper and lower covers and the handle is further out from the covers to ensure they can be reached with the now increased radius of the attachment plate. They also feature a slope from all sides to increase grip when changing the angle limits. The limiters were 3D-printed using Selective Laser Sintering (SLS) printing due to it having better mechanical

properties than traditional Fused Deposition Modeling (FDM) printers to ensure mechanical stability. Detailed drawings of the limiter are found in Appendix A.3 and A.4.

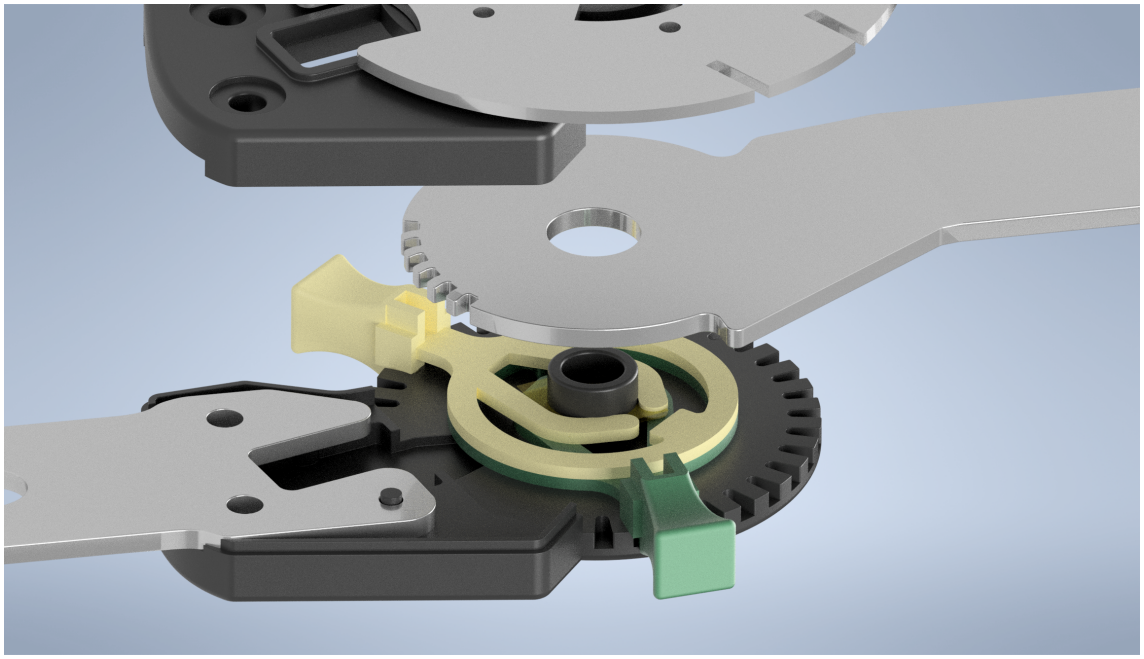


Figure 5.3: Angle limiters.

The limiters function by using two flexible beams that wrap around a central axle, see Figures 5.3 and 5.4. When pulled outwards, the beams flex, generating a spring force. This force allows the limiter to snap back securely into predefined grooves, each corresponding to a specific angle setting.

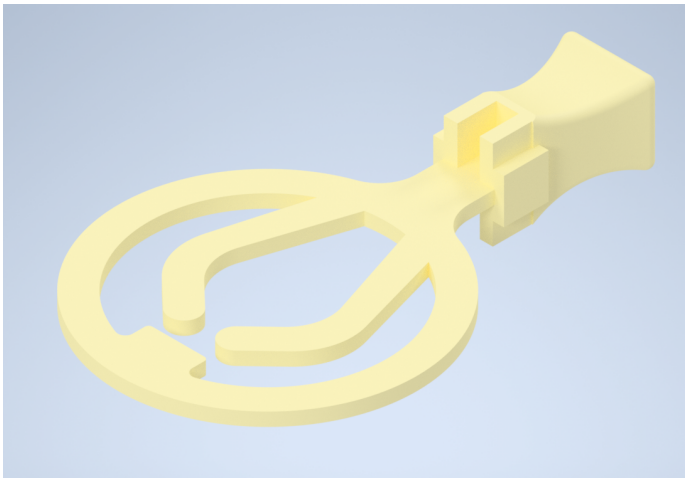


Figure 5.4: Angle limiter.

5.2 Electronics Enclosures

The following sections presents the 3D-printed parts enclosing the electronics. All being removable and adaptable to both right and left leg. The parts were carefully designed and adapted for modularity and accessibility to components and ports.

5.2.1 Main Control Unit Casing

The Raspberry Pi 5 casing is easily removable from the microcontroller/sensors casing, with a sliding mechanism and a stopping pin to secure it in place. The Raspberry Pi 5 is placed into the casing and fastened into the base with four M2.5 screws, the lid is then placed and fastened to the pre-installed M2.5 nuts on the casing with two additional screws. By removing the lid the battery can be changed easily.

The Raspberry Pi 5 casing has two variants. One is designed to fit the PiJuice UPS HAT with its original battery, and the other (which is not displayed more extensively) is designed to fit the PiJuice UPS HAT with a connected 2800 mAh lithium polymer battery. CAD pictures of the Raspberry Pi 5 casings are presented in Figures 5.5 and 5.6 with corresponding lids; see Figures 5.7 and 5.8. Detailed CATIA drawings of the variant with the original battery is provided in Appendix A.5 and A.6 for reference.

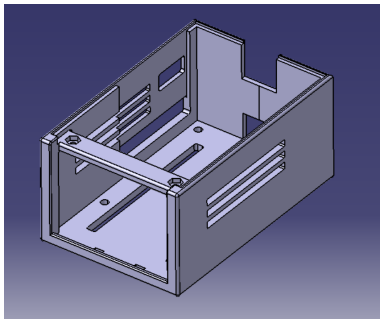


Figure 5.5: Casing, original battery.

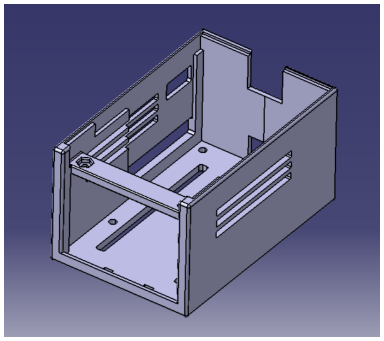


Figure 5.6: Casing, Li-Po 2800 mAh battery.

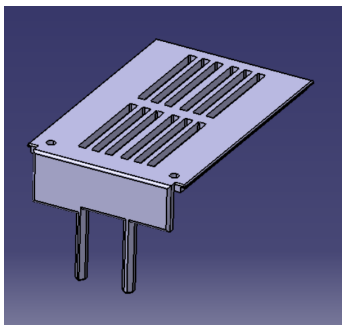


Figure 5.7: Casing lid, original battery.

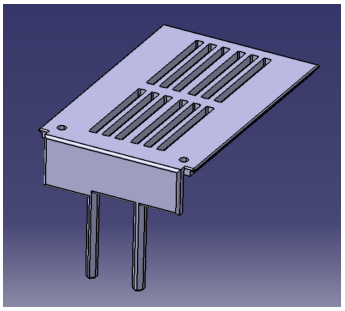


Figure 5.8: Casing, Li-Po 2800 mAh battery.

5.2.2 Embedded Electronics Casing

The Pico 2, ADS1015 and BNO085 was attached into the base of the casing with M2 and M2.5 nuts and screws.

Two similar casing designs were made for the microcontroller/sensors, where the slider for the Raspberry Pi 5 was either placed on the top of, or beside the casing. CAD pictures of the Microcontroller- and sensors casings are presented in Figures 5.9 and 5.10. Detailed CATIA drawings of the variant with the slider on the side, along with its lid is provided in Appendix A.7 and Appendix A.8 for reference.

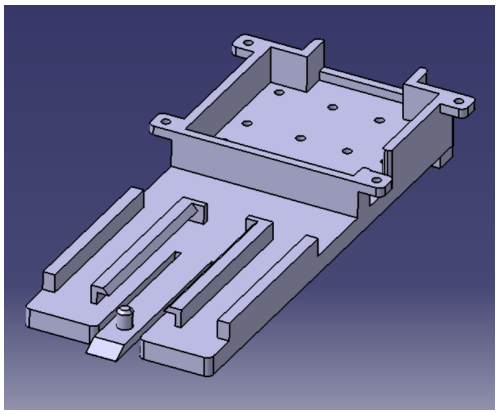


Figure 5.9: Slider beside casing.

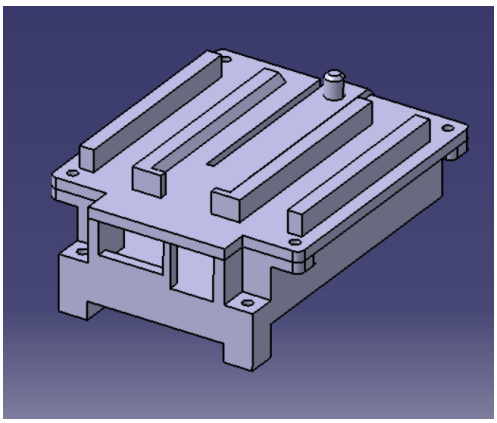


Figure 5.10: Slider on top of casing.

5.2.3 Clamping Piece

The clamping piece was intentionally made smaller than the width of the orthosis, enabling a clamping mechanism with M2.5 screws, holding the casings in place.

CAD pictures of the clamping piece are presented in Figure 5.11. A detailed CATIA drawing of the part is provided in Appendix A.9 for reference.

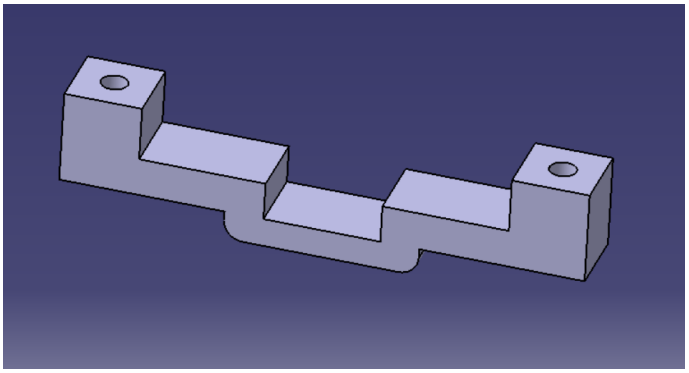


Figure 5.11: Clamping piece.

5.2.4 Encoder Attachment

The encoder housing was securely clamped onto the tibia rail, ensuring stable measurements throughout the full range of motion, see Figure 5.12. The clamping is done by M2.5 screws. The encoder is positioned so that it registers rotation as the tibia rail rotates.

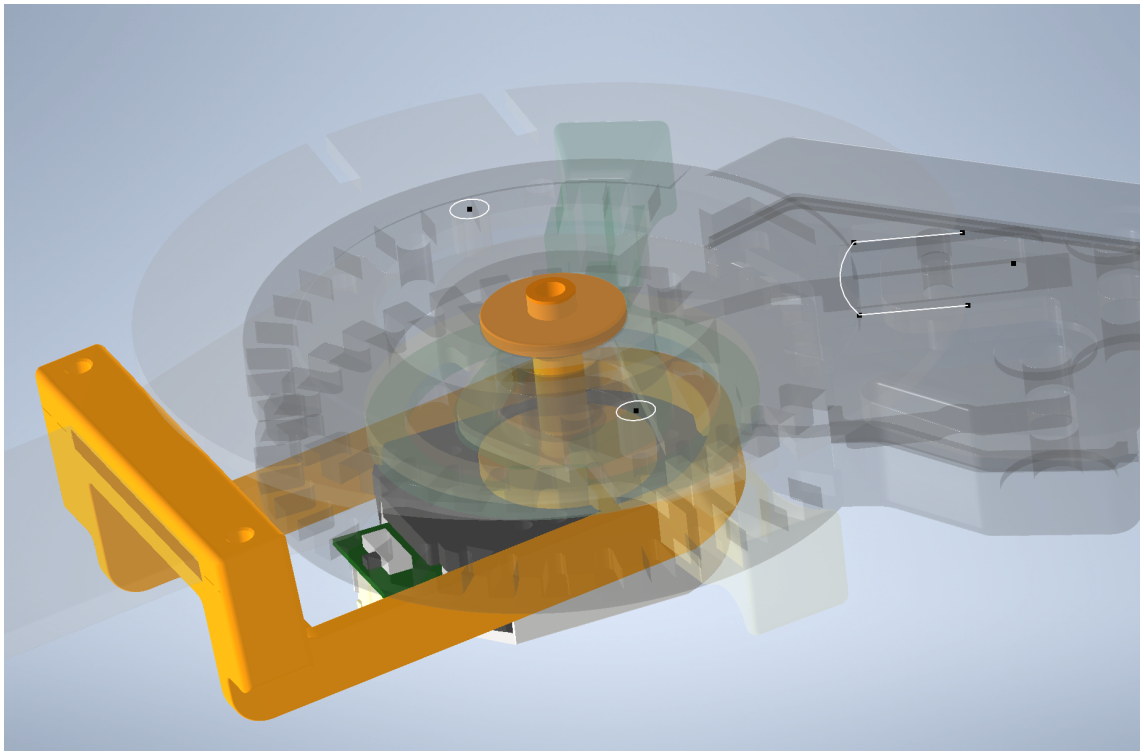


Figure 5.12: Encoder housing and mounting.

5.3 Electronics & Software

In this section the final electric system and schematics are presented.

5.3.1 Electrical System

The electrical system is powered by the Raspberry Pi 5, which contains the only battery in the system and supplies power to all components. It connects to the Pico 2 via USB, providing both communication and power. The Pico 2 acts as a central node for the sensors, handling both their power supply and data communication. The schematic in Figure 5.13 below illustrates the circuit: each 3.3V connection is supplied from the Pico 2's 3V3(OUT) pin, 5V from the VBUS pin, and all grounds are connected to one of the Pico 2's ground pins.

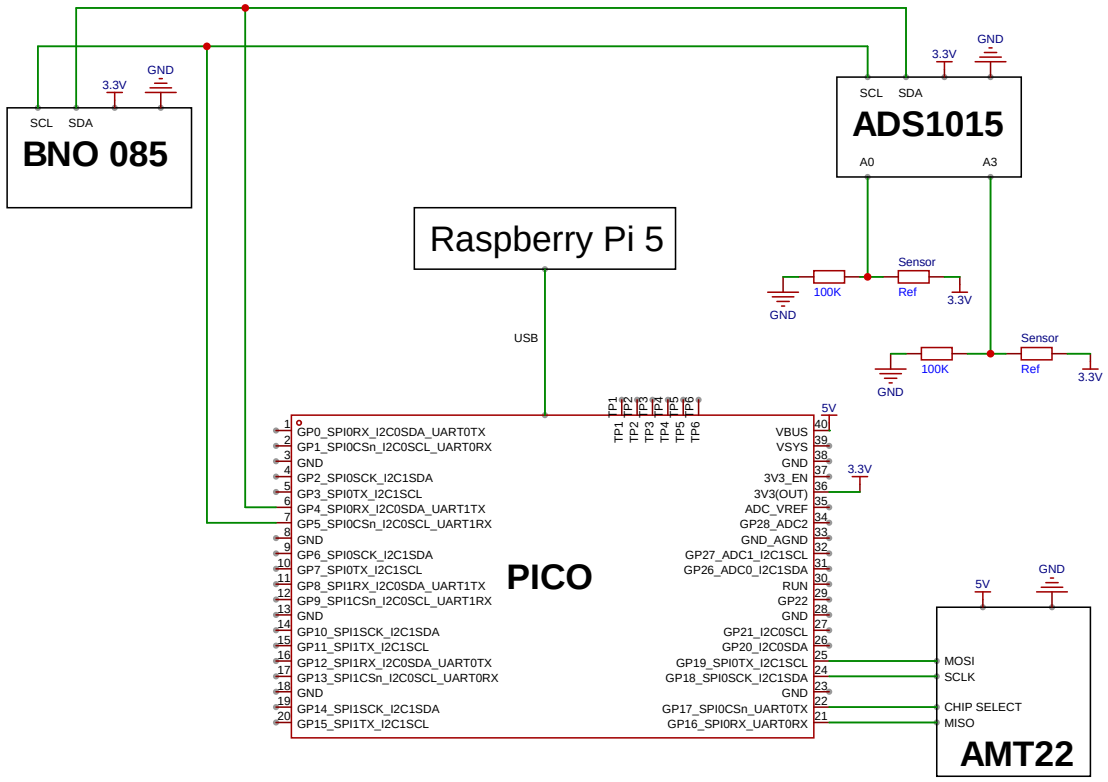


Figure 5.13: Schematic of electrical circuit.

The Pico 2, BNO085 (IMU) and ADS1015 (ADC) are all mounted within the same housing to minimize the amount of exposed cables. This means the only cables that are exposed are those connecting to the encoder as well as two cables for each force sensor.

When the actuator is to be used the electrical circuit includes some additional connections: A USB-CAN connection between the Raspberry Pi 5 and a 24–48V DC power supply to the actuator.

5.3.2 ROS 2 Environment

As previously stated, the software for this project was built using the ROS 2 framework as the underlying infrastructure for sensor data acquisition and control. The ROS 2 system that was built, consists of two primary nodes, called `motor_node` and `pico2_node`.

The motor_node consists of four services and one publisher, all of which use custom message interfaces built for their respective purpose. The node initiates the CAN driver and the myactuator_rmd SDK interface within a reconnection loop. All exceptions that may be raised by the myactuator_rmd API are handled to prevent the node from crashing. Instead, any issues encountered during motor interaction are logged as warning or error messages. In such cases, the node also disconnects from the CAN bus and re-enters the reconnection loop to attempt recovery. This approach

ensures system robustness by keeping the node alive even under faulty conditions and continuously attempting to recover. Additionally, due to the asynchronous nature of ROS 2, a thread lock was implemented to ensure that only one callback at a time could access the CAN bus, thereby preventing message conflicts.

The services in the motor node are called: `set_abs_pos`, `stop_motor`, `set_zero_pos` and `set_torque`. The set_abs_pos service is used to send an absolute position setpoint to the motor's integrated controller. The stop_motor service is used to send a command that disables the motor output and clears the motor's running state, placing it in an idle mode without any active control loop. The set_zero_pos service sends a command to the motor that sets the current encoder position as its zero position, allowing it to be used as a reference for subsequent absolute positioning. Lastly, the set_torque service is used to send a torque setpoint to the motor's internal controller.

The publisher in the motor node are publishing data to the topic called: `motor_status`. This data includes angular position, velocity, applied torque and internal temperature. The data is published continuously through a timer with a standard period of 0.1 seconds. The publishing rate is configurable at runtime through a parameter called `motor_status_pub_rate`, which defines the rate in Hertz and is initially set to 10Hz.

The pico2_node is implemented on the Pico 2 microcontroller, using the open-source micro-ROS project. This node consists of four publishers and two services. During testing, the node successfully integrated with the ROS 2 system through the micro-ROS agent running on the Raspberry Pi 5. Data was exchanged without issues over USB serial, and the reconnection loop worked well to restore communication after temporary disconnections.

The four publishers in the pico2_node publish to separate topics: `ext_encoder_pos`, `imu_data`, `force_upper_data` and `force_lower_data`. The ext_encoder_pos topic carries angular position data retrieved from the external encoder. The imu_data topic provides orientation in Euler angles (yaw, pitch, roll). Finally, the topics force_upper_data and force_lower_data carry filtered analog values read from the force sensors. The filtering and data conversions work well and result in clear and stable values.

The pico2_node provides two services: `set_ext_encoder_zero_service` and `set_imu_offset_service`. The set_ext_encoder_zero_service sets the zero position of the external encoder, enabling accurate joint angular position measurements regardless of initial alignment. The set_imu_offset_service sets the stored IMU offset values, which are initially zero, to the current IMU readings, making it easy to recalibrate the sensor orientation during operation.

The ROS 2 launcher file that launches the ROS 2 environment for this exoskeleton system includes: the motor_node, the micro-ROS agent and a rosbridge WebSocket

server. The rosbridge WebSocket server is an open-source, community-developed ROS 2 utility that enables interaction with the ROS 2 environment from a web application. It is used to facilitate communication between the ROS 2 system and the web interface. Similarly, the micro-ROS agent acts as a gateway for the pico2_node, enabling it to integrate with the existing ROS 2 system.

An overview of the system architecture, illustrating both the hardware and software layers, is shown in Figure 5.14. The diagram depicts the complete exoskeleton system, showing all components, their roles, and how they communicate through various hardware interfaces and ROS 2 communication protocols. The hardware components are marked with solid lines, while the software components are marked with dashed lines. The source code corresponding to the motor_node and pico2_node shown in the figure is provided in Appendix B.1 and B.2, respectively.

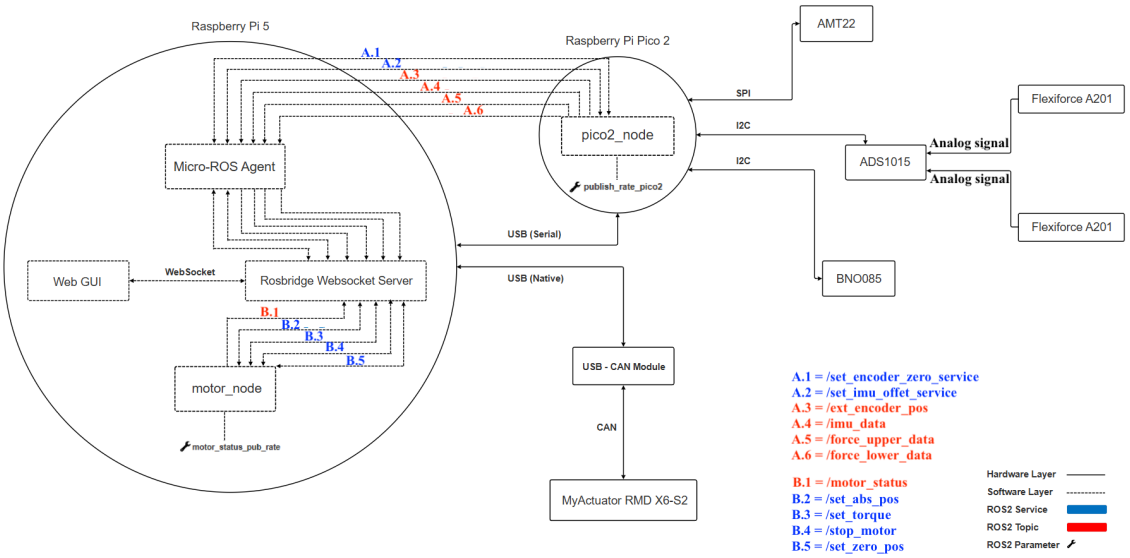


Figure 5.14: System overview showing the electrical hardware and software architecture of the exoskeleton system.

5.3.3 Web-based User Interface

The web application consists of three components: a real-time dashboard, a historical data dashboard, and a settings and calibration dashboard. All pages share a common WebSocket connection to the rosbridge WebSocket server. The connection status is displayed in the top-left corner of each page, as shown in Figures 5.15, 5.16 and 5.17. The functionality of each page is described in detail in the following paragraphs.

The real-time dashboard is designed to provide both real-time data visualization and motor control. An overview of the dashboard is shown in Figure 5.15. On the left side of the page, two vertically stacked sections display the current status of the motor unit and real-time sensor readings. The displayed values reflect the most

recent data published on their respective ROS 2 topics.

The central section of the page contains two time-series plots: one representing the joint angle over time, and the other representing the applied motor torque over time. These charts are updated dynamically whenever new data is received on their respective ROS 2 topics. Below the charts, there are buttons for starting, stopping, and clearing the plots, as well as for downloading a data log of the most recently plotted values.

The right side of the page features a motor control panel, which allows the user to control the actuator through the web interface using a form and two control buttons. The form allows the user to choose between sending a position or torque control command. When position control mode is selected, two sliders are available: one for specifying the motor speed, ranging from 0-100% (where 100% corresponds to 360°/s), and one for setting the desired absolute angular position. When torque control mode is selected, a single slider is used to specify the desired torque. The Submit button sends the specified command to the motor, while the Stop Motor button is used to stop the motor during operation. This functionality is enabled through the use of the ROS 2 services described in the previous section.

The historical data dashboard is intended to provide an interface for the user to view and analyze sensor data collected during daily activities. An overview of this dashboard is shown in Figure 5.16. The interface consists of a panel on the left side, where the user can select a data file from the database based on a specific date. Below the selection dropdown are buttons for displaying and downloading the selected data file. Once a file is selected and displayed, its contents are plotted in the chart area on the right side of the page. The backend for historical data storage is not yet implemented, but the interface is designed to support this functionality in future versions.

The settings and calibration dashboard is designed to allow the user to configure system settings and perform sensor calibration. An overview of the dashboard is presented in Figure 5.17. The dashboard contains three sections: one for setting the sampling frequency, one for calibrating the IMU, and one for calibrating the encoder. The sampling frequency is adjusted by selecting the desired value using a slider and pressing the Set Frequency button. This updates the corresponding ROS 2 parameter, which dynamically changes the publishing rate at runtime. There are also buttons for setting the IMU offset and the encoder zero positions. Each button triggers a corresponding ROS 2 service that perform the respective calibration task.

Testing showed that the web interface and its functionalities worked as intended. The WebSocket connection to the rosbridge server remained stable, enabling reliable communication between the web application and the ROS 2 system. The interface was fast and responsive. Control commands were executed with minimal delay, and sensor data was dynamically updated and visualized continuously.

Figure 5.15: Web Application: Real-time Dashboard.

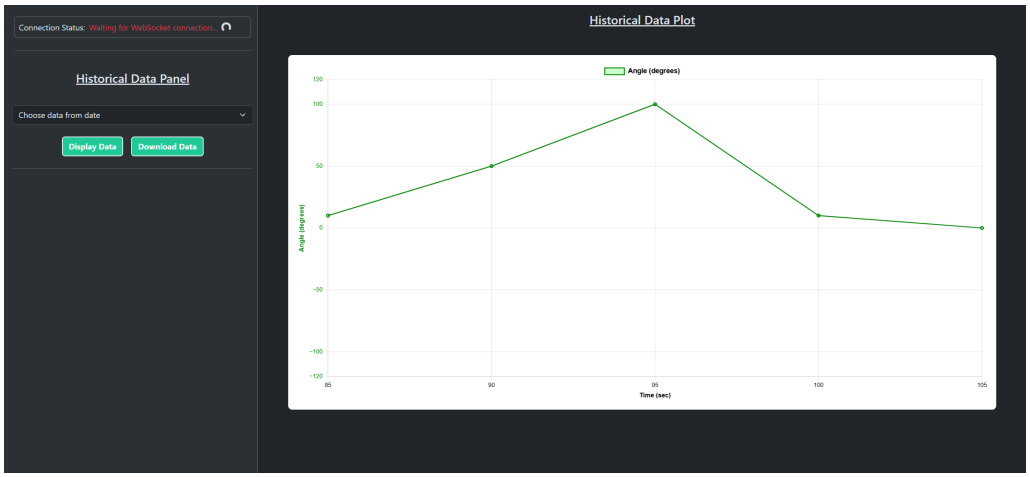


Figure 5.16: Web Application: Historical Data Dashboard.

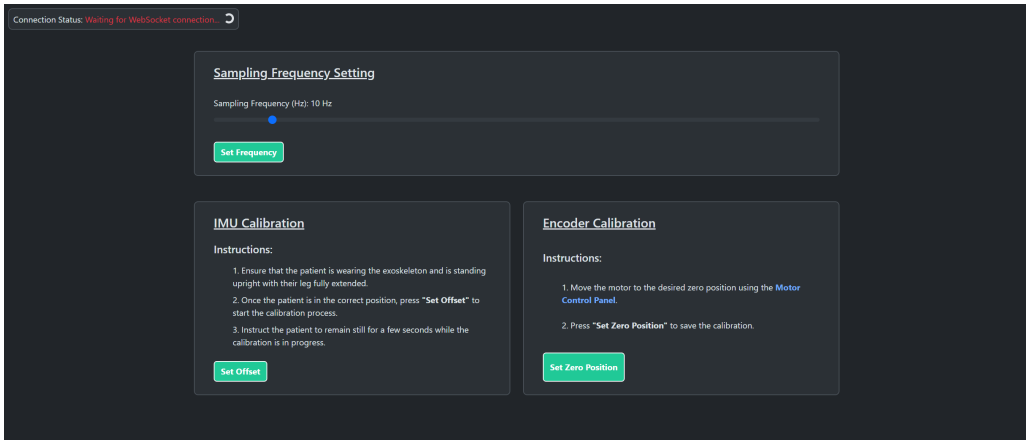


Figure 5.17: Web Application: Settings & Calibration Dashboard.

5.4 Final assembly

The attachment plate, motor housing, angle limiters, encoder housing, force sensors and cables were put together with the original orthosis, creating the final assembly. To show the final assembly and design, pictures were taken to provide a visual representation of the final prototype with the components all put together, to highlight the placement of individual components and to show the complete assembly worn on the leg. These photos illustrates the practical fit and modularity of the final design and are presented in Figures 5.18 to 5.28.

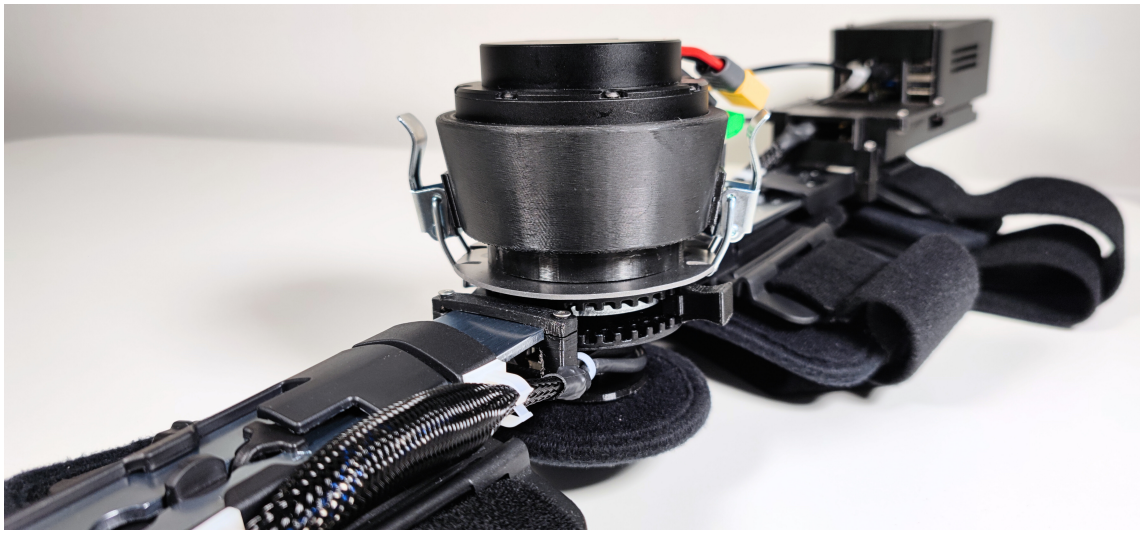


Figure 5.18: Close up of the final assembly showing the two eccentric locks on the side of the motor housing connecting to the attachment plate.

Figure 5.19: The final assembly folded.

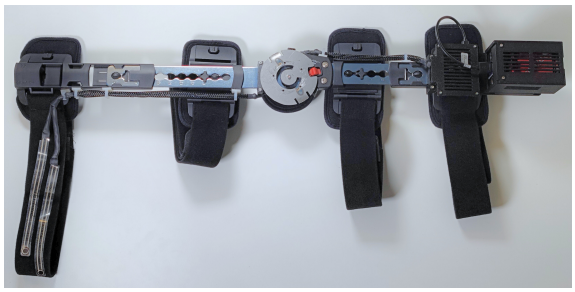


Figure 5.20: The orthosis without the motor and motor housing being fully extended.

Figure 5.21: The orthosis without the motor and motor housing being fully retracted.

Figure 5.22: Full view of the final assembly.

Figure 5.23: The inside of the fully assembled orthosis.

Figure 5.24: The inner leg side (furthest up) and outer leg side (furthest down).

Figure 5.25: Side view of the full assembly with all components (motor, motor housing, attachment plate, angle limiters, electronics housing, encoder housing, force sensors and cables.

Figure 5.26: Side view of assembly without the motor and motor housing, and only the attachment plate, angle limiters, electronics housing, encoder housing, force sensors and cables.

Figure 5.27: The orthosis without the motor worn by a user, showing how the construction fits and aligns with the leg.

Figure 5.28: The orthosis fully assembled worn by a user, showing how the construction fits and aligns with the leg.

5.4.1 Modular Function

The final design consists of 4 independent modules: The actuator, the Raspberry Pi 5, the encoder housing and the embedded electronics. The actuator and Raspberry Pi 5 are very easily detachable by anyone while the embedded electronics and encoder

housing are removable using screws. Additionally the Raspberry Pi 5 and embedded electronics can be easily detached by removing the upper cuff itself from the X-ROM orthosis.

The modification and addition to the original orthosis in no way compromises the function of the orthosis should it be used without the added features.

6

Discussion

This section discusses the challenges that have been encountered and potential future improvements during the project.

6.1 Challenges

This section addresses the main problems that proved challenging throughout the project.

6.1.1 Lack of existing detachable motor solutions

During the course of the project, it proved challenging to find existing solutions specifically designed for detachable motor systems in rehabilitation orthoses. Most available assistive devices are either permanently integrated into the orthosis or designed for use on only one leg, limiting their flexibility in rehabilitation exercises that require alternating support.

6.1.2 Attachment design and assembly challenges

This lack of standardized solutions meant that we had to develop a completely custom design that would allow the motor to be easily removed and reattached on the opposite side. The solution also had to be user-friendly, durable, and compatible with the existing mechanics of the orthosis. The result was a tailored attachment mechanism based on modular design principles, specifically adapted for our prototype system. Since the project is based on a pre-existing orthosis, modifications were necessary to enable integration with the detachable motor system. A custom attachment plate was designed to allow for robust and secure fastening of the motor to the orthosis. The motor was mounted using eccentric locks, chosen for their intuitive handling and ability to enable quick attachment and detachment without the need for tools.

6.1.3 Motor mounting difficulties

Attaching the motor proved to be a challenging and time consuming task. The pegs of the motor housing, combined with the motor's peg had to fit perfectly into their

corresponding holes in the mounting plate and the lower leg rail. Ensuring that all the pegs were aligned simultaneously required repeated small adjustments. The process was very fiddly and frequent slight misalignment made motor attachment more difficult than anticipated.

6.1.4 Prototype manufacturing

Due to the scope of the project being to develop a prototype, the manufacturing methods that would be used in a professional setting are not available during this project. This means that part will not have the same finish or mechanical properties as a finished product. The 3D-printed parts will not have the same structural integrity or finish as an injection molded part. This is because the 3D-printed parts are not solid. The attachment plate that was made by traditional machining, also got a rougher finish than the water jetted one. This means that CNC machining and water jetting would be better for saving time and making the finish look better.

6.1.5 Motor selection

The motor used for this project was recommended by the supervisor Fabian Just. Just had calculated the torque requirements based on 100 % assist, meaning that the motor can hold the leg against gravity. However, after conversations with Sahlgrenska Hospital and getting a better understanding of their needs, it was decided that the motor was too strong. Sahlgrenska Hospital wanted less assist, meaning that the motor can be around half the power and size. This makes the exoskeleton less bulky and easier to wear. The same mechanism used for this exoskeleton can be used for a smaller motor. The location of the cutout in the top cover and attachment plate and the hole on the lower leg rail for the peg from the motor needs to be changed to match the new motor's dimensions.

6.1.6 Structure weakness and safety

A major concern identified during the project is that the current construction is fundamentally too weak to handle the full torque of the motor. If the motor was to operate at full power, the mechanical structure of the exoskeleton, primarily the plastic components would fail. This reinforces the earlier claim of the motor being overdimensioned. To ensure patient and device safety, it is important to implement a stronger construction and a safety precaution in the control algorithm that halts the motor when opposing force is detected. Conducting FEM simulations in future iterations would significantly enhance safety verification.

6.1.7 Ergonomics

The current design of the exoskeleton has some ergonomic problems. Due to the combined volume of the control unit and the integrated battery, they may be considered cumbersome in the context of wearable devices. One concern is that due to the positioning and bulk of the control unit, it may interfere with the patient's nat-

ural arm swing when walking. Hence, reducing the size of the embedded electrical hardware should be a priority in further developments of this exoskeleton.

6.1.8 Sensor accuracy

Since no testing was done regarding the actuators torque accuracy or calibration of the force sensors they cannot be evaluated. Furthermore, no sensors underwent testing to verify the accuracy though the function was verified for all sensors.

6.2 Future improvements

This section addresses some flaws with the design that, with benefit, should be modified in the future.

6.2.1 Improving assembly

To simplify assembly in the future, several improvements could be considered. Designing a motor housing or attachment plate with a guiding mechanism, like tapered alignment holes, or a guiding bracket, could help guide the pegs into their respective holes. Reducing the number of alignment points or consolidating them into a more convenient mechanism could also decrease the risk of misalignment. Additionally, incorporating a sort of flexible or quick release mechanism to fasten the motor housing could make the fastening process more efficient, user-friendly, and quicker. This type of mechanism would be ideal for its ease of use.

6.2.2 Modularity

The modular design of the prototype offers several practical advantages in terms of maintenance, repair, and long-term usability. In the event that a component—such as a sensor, actuator, or microcontroller—fails or wears out, it can be individually replaced without disassembling the entire system. This minimizes downtime and simplifies troubleshooting. Additionally, because the design uses standardized components and symmetrical layouts, spare parts do not need to be side-specific, reducing inventory complexity. This makes the system more cost-effective over time and easier to support in both clinical and home rehabilitation settings. The quick-release features further allow users or support staff to perform replacements or adjustments without technical training.

6.2.3 Embedded hardware improvements

There are several feasible alternatives for changes in the embedded hardware, that won't require major changes to the software architecture or compromise with the system functionality. One proposed change is to replace the current microcontroller with one that offers integrated wireless communication capabilities, such as the ESP32, which supports both Wi-Fi and Bluetooth. With this setup, a host computer at the clinic where the patient performs their exercises could run a ROS 2

environment hosting a micro-ROS agent that listens for data transmitted over Wi-Fi. This implies that the central control unit would be located outside the physical system and would interface with the system wirelessly. For data acquisition during everyday activities, the microcontroller could temporarily buffer the data using an external storage module. Once an internet connection becomes available, the stored data could then be uploaded to a cloud server for further processing or analysis.

This suggested configuration would eliminate the need for a single board computer like the Raspberry Pi 5 and could also allow for a reduction in battery size. Resulting in a significantly more compact control unit, and thereby enhancing both the ergonomics of the device and the overall user experience. Furthermore, designing a custom Printed Circuit Board (PCB) in future iterations would be advisable, as this could further reduce the size of the embedded hardware, optimize power consumption, and facilitate improved integration between the electrical system and the mechanical hardware.

6.2.4 Waterproofing

An additional problem with the current design is that the electronics are not enclosed in a waterproof manner. There are, to some degree, exposed electrical components that are susceptible to water. In fixing this problem, the problem of ventilation and airflow for the electrical components arises.

6.2.5 Aesthetic and patient trust

While mechanical functionality was the primary focus, the ergonomics, comfort and impression of the exoskeleton are still very important. For future designs, the weight-balance needs to be looked over to ensure that the patient does not experience any discomfort during wear. How the exoskeleton looks, should also not intimidate the patient or the therapist. They need to have trust in the exoskeleton in order to maximize rehabilitation.

7

Conclusion

This project set out to develop a motorized, modular exoskeleton design that builds upon and maintains the core functionality of the already existing knee orthosis. Through the design of the actuator attachment and the removable electronics, the system was successfully adapted to allow interchangeability between the left and right leg. The design of these detachable parts was made considering user-friendliness as well as accessibility.

This project also adequately set up a data collection system with 2 force sensors, an IMU and an encoder with relevant software for data acquisition. Additionally, a ROS 2 environment was created to allow effective communication that can be expanded upon in the future when setting up potential system control algorithms for the actuator.

To improve the design in the future, a slightly less powerful and smaller actuator should be considered. A customized PCB fit for the specific needs would reduce the bulkiness and excess electronics. Finally, a more sophisticated solution with non-exposed wiring would enhance both the appearance and durability of the orthosis.

Bibliography

[1] B. Chen, B. Zi, Z. Wang, L. Qin, and W.-H. Liao, "Knee exoskeletons for gait rehabilitation and human performance augmentation: A state-of-the-art," *Mechanism and Machine Theory*, vol. 134, pp. 499–511, 2019. [Online]. Available: https://doi.org/10.1016/j.mechmachtheory.2019.01.016

[2] R. Nordenvall, S. Bahmanyar, J. Adami, C. Stenros, T. Wredmark, and L. Felländer-Tsai, "A population-based nationwide study of cruciate ligament injury in sweden, 2001–2009: Incidence, treatment, and sex differences," *The American Journal of Sports Medicine*, vol. 40, no. 8, pp. 1808–1813, 2012. [Online]. Available: https://journals.sagepub.com/doi/10.1177/0363546512449306

[3] Folksam, "Idrottsforskning," 2025, accessed: 2025-05-05. [Online]. Available: https://www.folksam.se/forsakringar/idrottsforsakring/idrottsforskning/

[4] M. Åman, M. Forssblad, and K. Larsén, "National injury prevention measures in team sports should focus on knee, head, and severe upper limb injuries," *Knee Surgery, Sports Traumatology, Arthroscopy*, vol. 27, pp. 1000–1008, 2019. [Online]. Available: https://doi.org/10.1007/s00167-018-5225-7

[5] R. Yli-Ikkelä, A. Rintala, A. Köyhäjoki, H. Hakonen, H. Korpi, M. Kantola, S. Honkanen, O. Ilves, T. Sjögren, J. Karvanen *et al.*, "Effectiveness of robot-assisted lower limb rehabilitation on balance in people with stroke: A systematic review, meta-analysis, and meta-regression," in *Nordic Conference on Digital Health and Wireless Solutions*. Springer, 2024, pp. 101–116. [Online]. Available: https://doi.org/10.1007/978-3-031-59091-7_7

[6] A. B. Payedimarri, M. Ratti, R. Rescinito, K. Vanhaecht, and M. Panella, "Effectiveness of platform-based robot-assisted rehabilitation for musculoskeletal or neurologic injuries: a systematic review," *Bioengineering*, vol. 9, no. 4, p. 129, 2022. [Online]. Available: https://doi.org/10.3390/bioengineering9040129

[7] Enovis, "X-rom | enovis," 2024, accessed: 2024-03-04. [Online]. Available: https://enovis.com/products/donjoy/x-rom

[8] Oxford English Dictionary, "Anthropometry," 2024, accessed: 2025-05-08. [Online]. Available: https://www.oed.com/dictionary/anthropometry_n

[9] L. Hanson, L. Sperling, G. Gard, S. Ipsen, and C. O. Vergara, "Swedish anthropometrics for product and workplace design," *Applied Ergonomics*, vol. 40, no. 4, pp. 797–806, Jul. 2009. [Online]. Available: https://doi.org/10.1016/j.apergo.2008.08.007

[10] Kollmorgen, "What is a servo motor?" 2024, accessed: 2025-05-11. [Online]. Available: https://www.kollmorgen.com/en-us/resources/technologies-explained/motors/what-is-a-servo-motor

[11] SBG Systems, "Inertial measurement unit (imu) sensor," 2024, accessed: 2025-05-11. [Online]. Available: https://www.sbg-systems.com/glossary/inertial-measurement-unit-imu-sensor/

[12] Omron Corporation, "Introduction to sensors," 2024, accessed: 2025-05-11. [Online]. Available: https://www.ia.omron.com/support/guide/34/introduction.html

[13] XJCSENSOR, "Types of force sensors," 2024, accessed: 2025-05-11. [Online]. Available: https://www.xjcsensor.com/types-of-force-sensors/

[14] MyActuator, "V2-x6-40 details | myactuator," 2024, accessed: 2024-03-09. [Online]. Available: https://www.myactuator.com/x6-40-details

[15] CEVA, Inc., "BNO080/BNO085 Intelligent 9-Axis Absolute Orientation Sensor," 2018, accessed: 2025-05-11. [Online]. Available: https://www.ceva-ip.com/wp-content/uploads/BNO080_085-Datasheet.pdf

[16] Same Sky Devices, "AMT22 Series Modular Absolute Encoder Datasheet," 2024, accessed: 2025-05-11. [Online]. Available: https://www.sameskydevices.com/product/resource/amt22.pdf

[17] Tekscan, Inc., "FlexiForce® Sensors User Manual," 2010, accessed: 2025-05-11. [Online]. Available: https://cdn.sparkfun.com/datasheets/Sensors/ForceFlex/FLX-FlexiForce-Sensors-Manual.pdf

[18] Texas Instruments, "ADS1015 12-Bit Analog-to-Digital Converter Datasheet," 2010, accessed: 2025-05-11. [Online]. Available: https://cdn-shop.adafruit.com/datasheets/ads1015.pdf

[19] Raspberry Pi Ltd, "Raspberry Pi Pico 2 Datasheet," 2024, accessed: 2025-05-11. [Online]. Available: https://datasheets.raspberrypi.com/pico/pico-2-datasheet.pdf

[20] ——, "Raspberry Pi 5 Product Brief," Raspberry Pi Ltd, 2025, accessed: 2025-05-11. [Online]. Available: https://datasheets.raspberrypi.com/rpi5/raspberry-pi-5-product-brief.pdf

[21] Pi Supply, "PiJuice HAT – A Portable Power Platform For Every Raspberry Pi," 2017, accessed: 2025-05-11. [Online]. Available: https://mm.digikey.com/Volume0/opasdata/d220001/medias/docus/2262/PIS-0212_Web.pdf

[22] Open Robotics, "The ros ecosystem," n.d., accessed: 2025-05-07. [Online]. Available: https://www.ros.org/blog/ecosystem/

[23] A. S. Al-Batati, A. Koubaa, and M. Abdelkader, "Ros 2 key challenges and advances: A survey of ros 2 research, libraries, and applications," *Preprints*, October 2024. [Online]. Available: https://doi.org/10.20944/preprints202410.1204.v1

[24] Open Robotics, "Understanding nodes," n.d., accessed: 2025-05-10. [Online]. Available: https://docs.ros.org/en/jazzy/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Nodes/Understanding-ROS2-Nodes.html

[25] ——, "Understanding topics," n.d., accessed: 2025-05-10. [Online]. Available: https://docs.ros.org/en/jazzy/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Topics/Understanding-ROS2-Topics.html

[26] ——, "Understanding services," n.d., accessed: 2025-05-10. [Online]. Available: https://docs.ros.org/en/jazzy/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Services/Understanding-ROS2-Services.html

[27] ——, "Parameters," n.d., accessed: 2025-05-10. [Online]. Available: https://docs.ros.org/en/jazzy/Concepts/Basic/About-Parameters.html

[28] micro-ROS, "micro-ros | ros 2 for microcontrollers," 2024, accessed: 2025-05-10. [Online]. Available: https://micro.ros.org/

[29] Raspberry Pi Ltd, "Raspberry Pi 5 CAD STEP file," 2025, accessed: 2025-05-13. [Online]. Available: https://www.raspberrypi.com/documentation/computers/raspberry-pi.html#schematics-and-mechanical-drawings

[30] T. Flatscher, "Myactuator rmd x-series can driver sdk," 2024, accessed: 2025-05-13. [Online]. Available: https://github.com/2b-t/myactuator_rmd

[31] Robot Web Tools, "rosbridge_suite," n.d., accessed: 2025-05-13. [Online]. Available: https://github.com/RobotWebTools/rosbridge_suite

[32] Wiberger AB, "Eccentric lock el1035-200," 2024, accessed: 2024-04-03. [Online]. Available: https://www.wiberger.se/produkt/el1035-200

Appendix A: CAD Drawings

A.1 Technical drawing of attachment plate

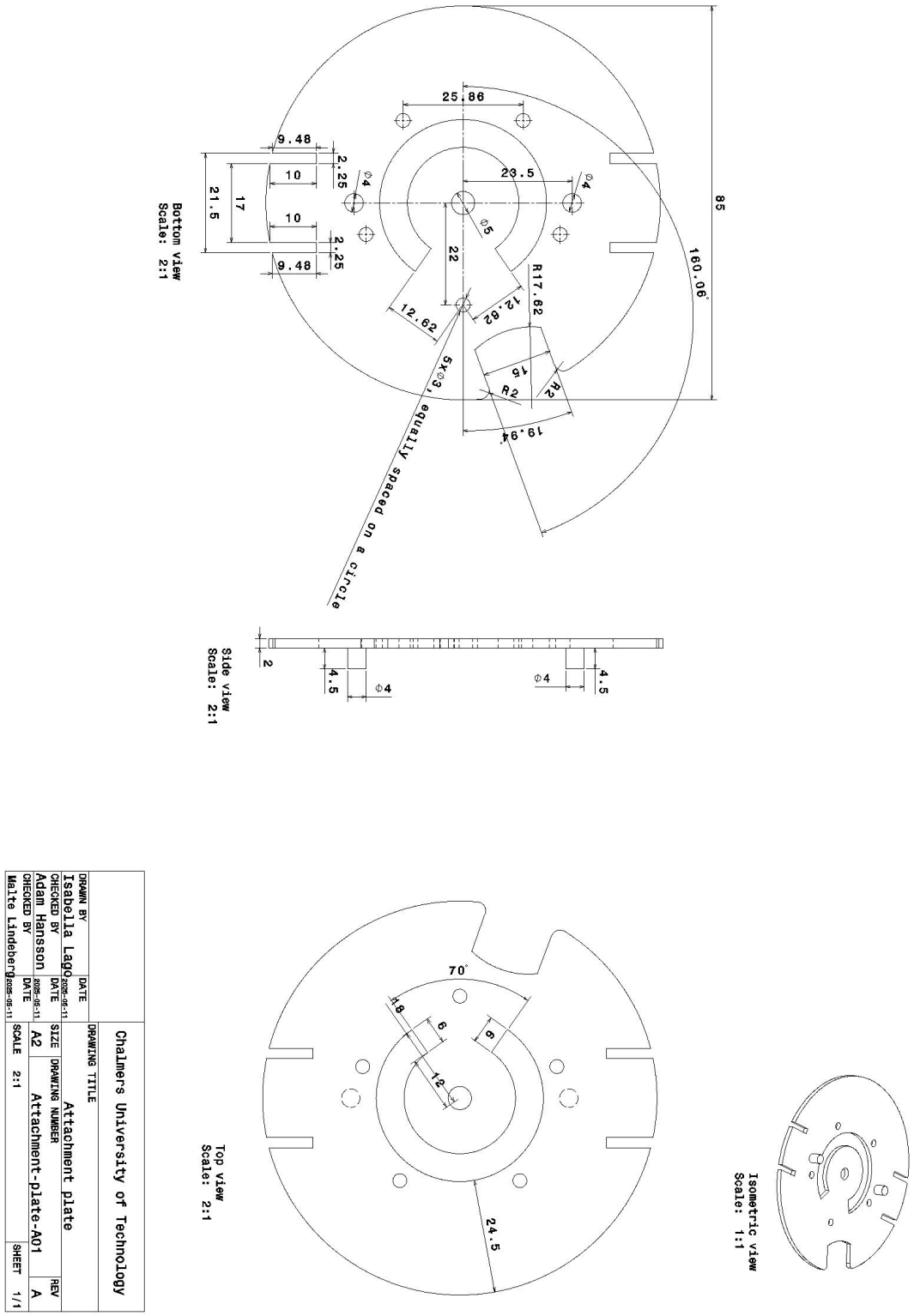


Figure A.1: Attachment plate CAD drawing.

A.2 Technical drawing of motor housing

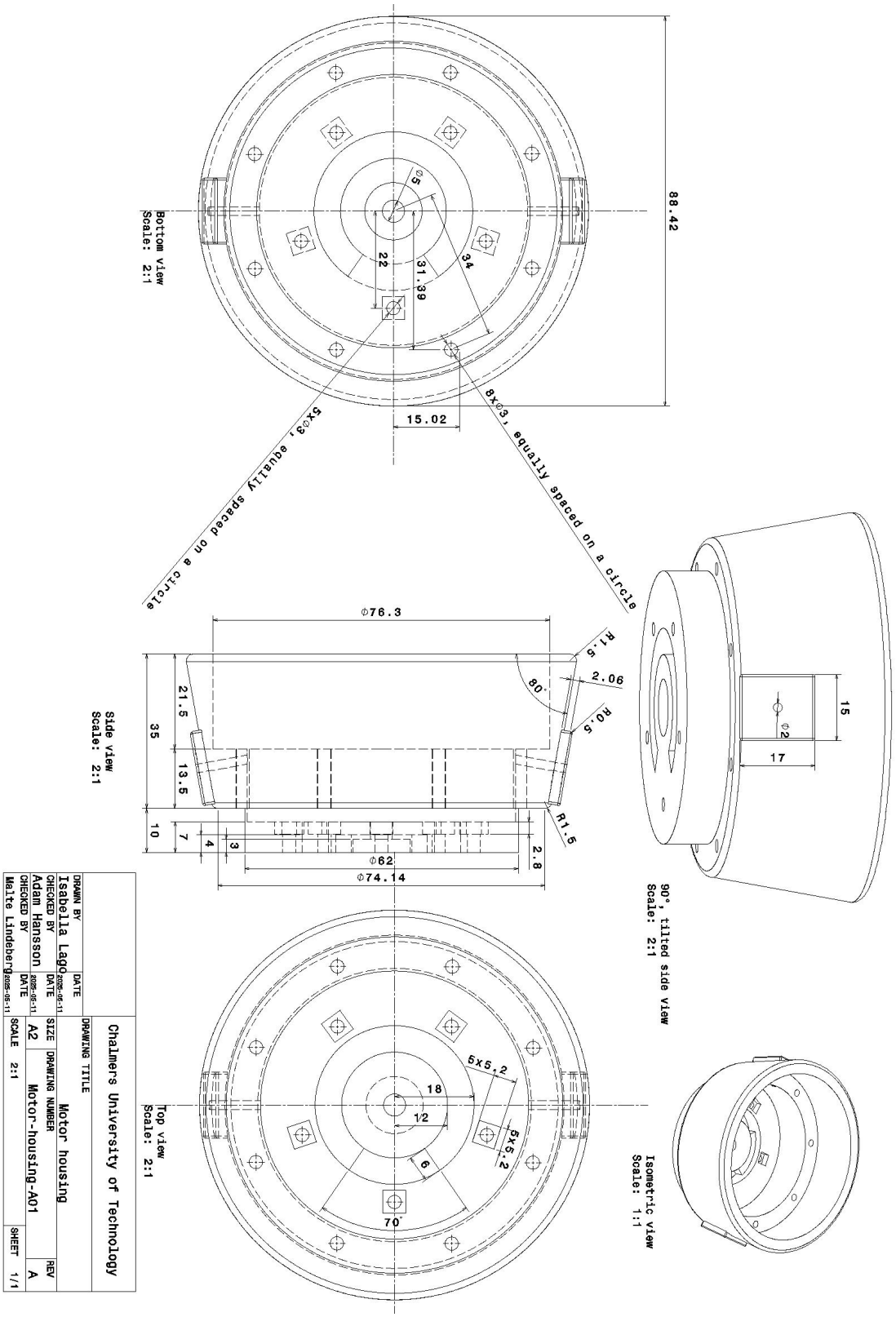


Figure A.2: Motor housing CAD drawing.

A.3 Technical drawing of angle limiter

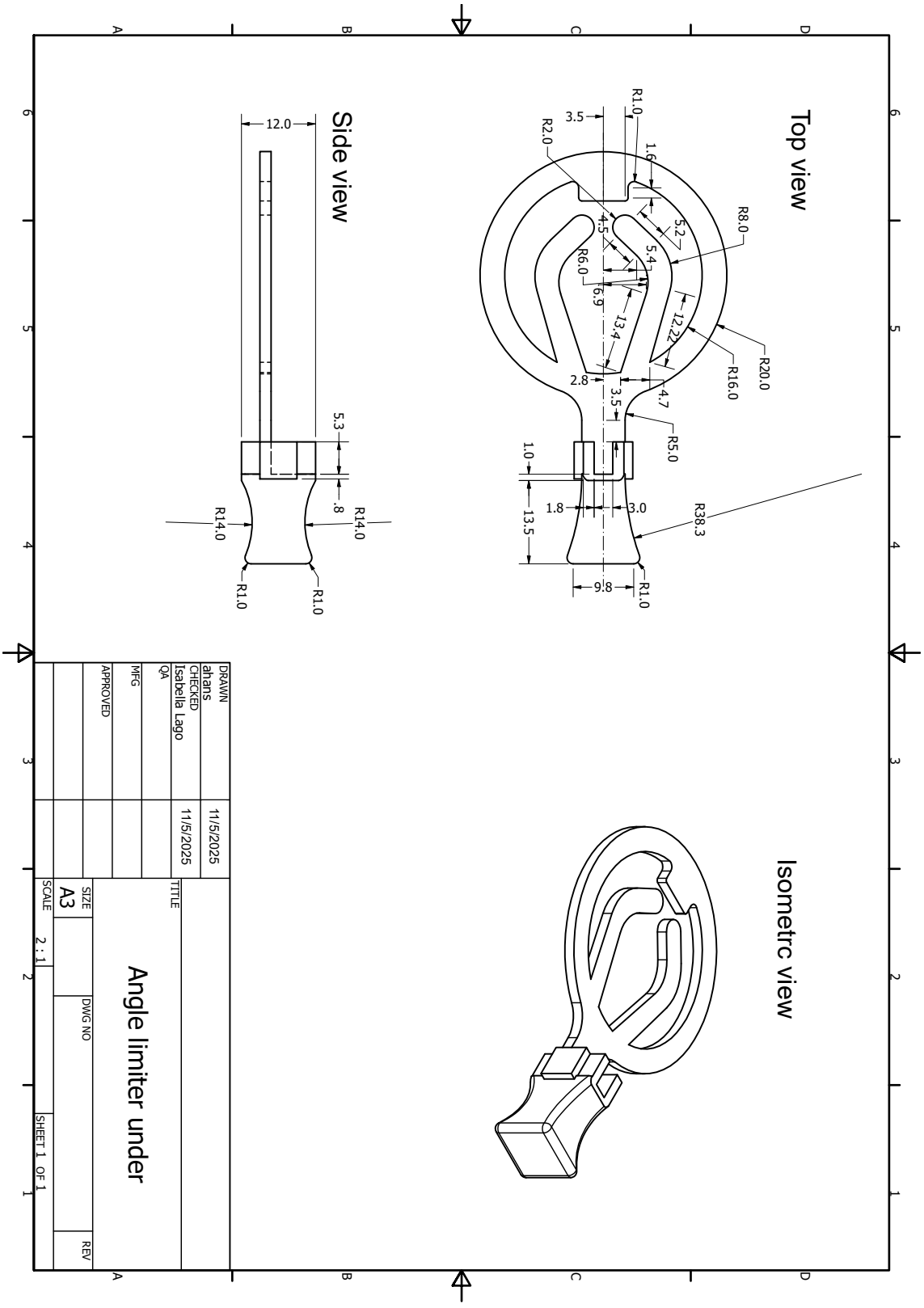


Figure A.3: Angle limiter under.

A.4 Technical drawing of angle limiter

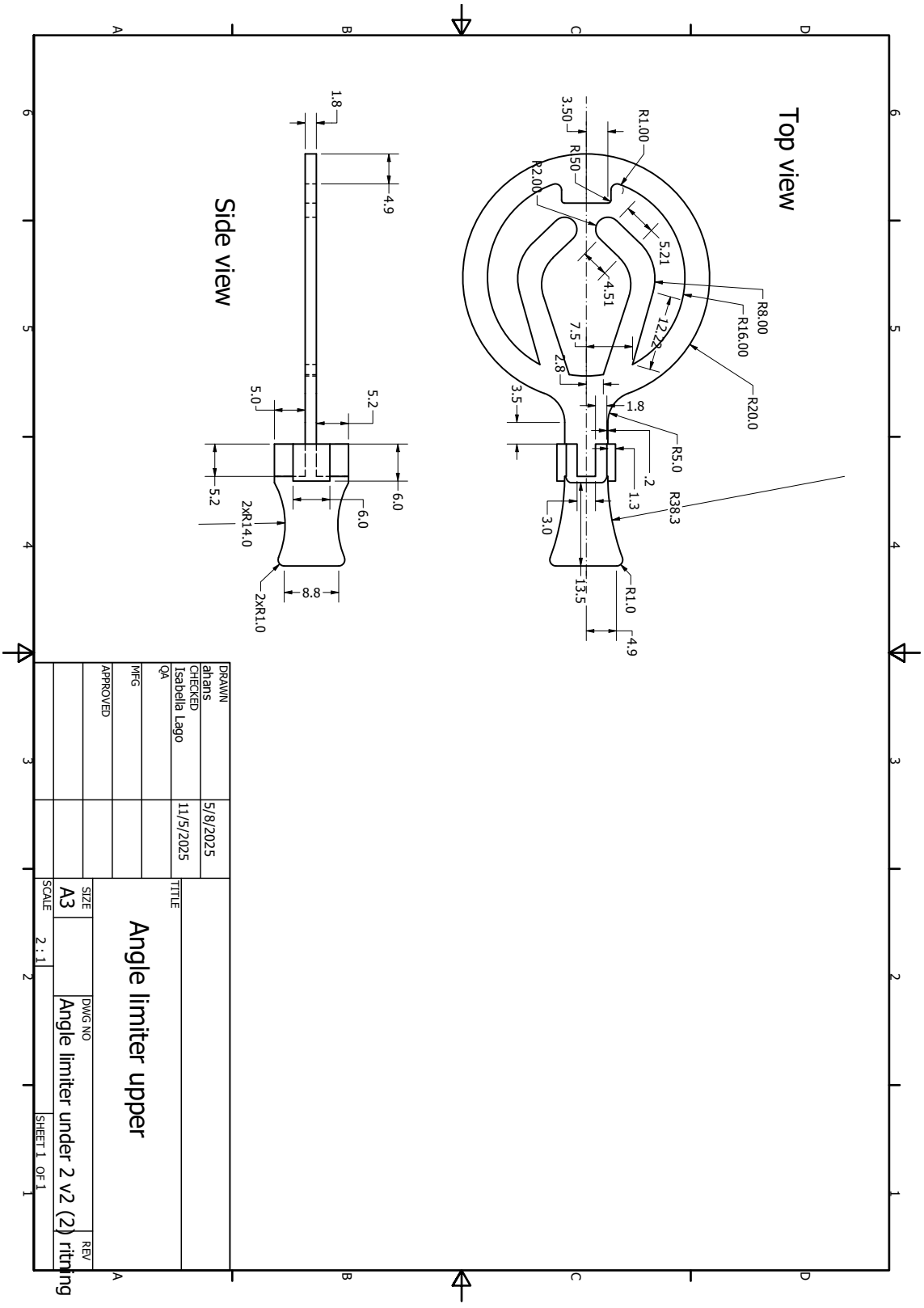


Figure A.4: Angle limiter over.

A.5 Technical drawing of Raspberry Pi 5 casing

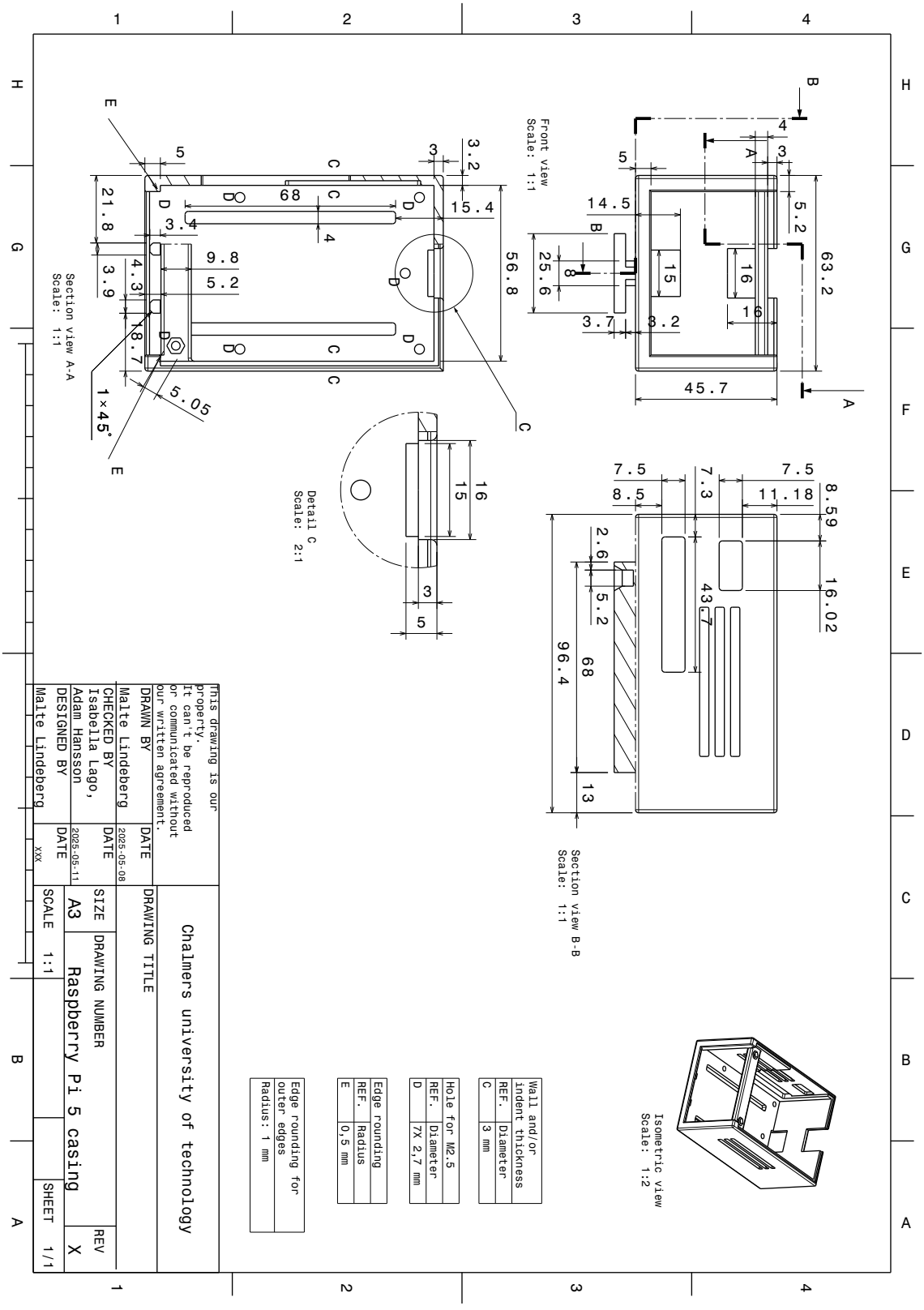


Figure A.5: Raspberry Pi 5 casing CAD drawing.

A.6 Technical drawing of Raspberry Pi 5 casing lid

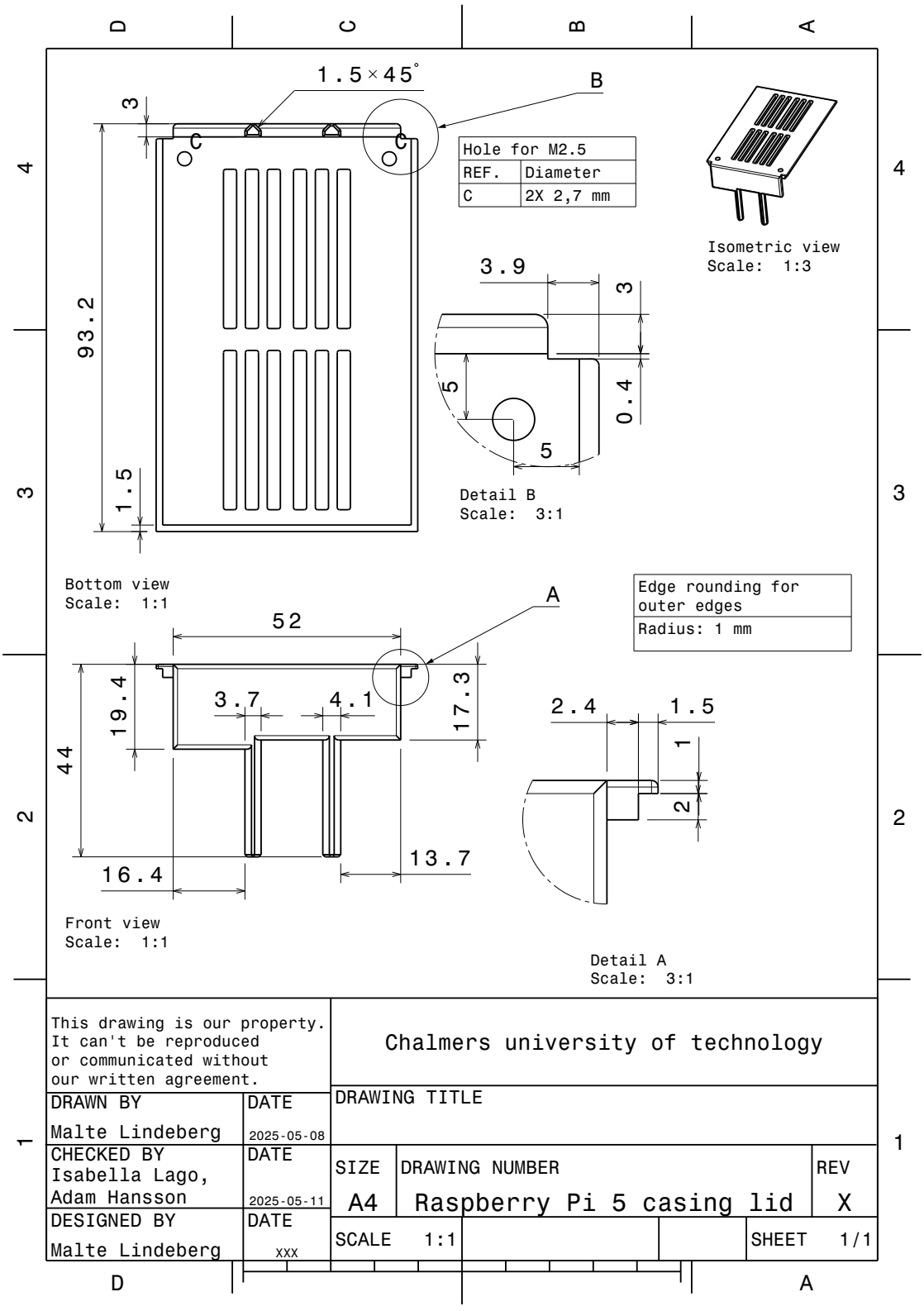


Figure A.6: Raspberry Pi 5 casing lid CAD drawing.

A.7 Technical drawing of embedded electronics casing

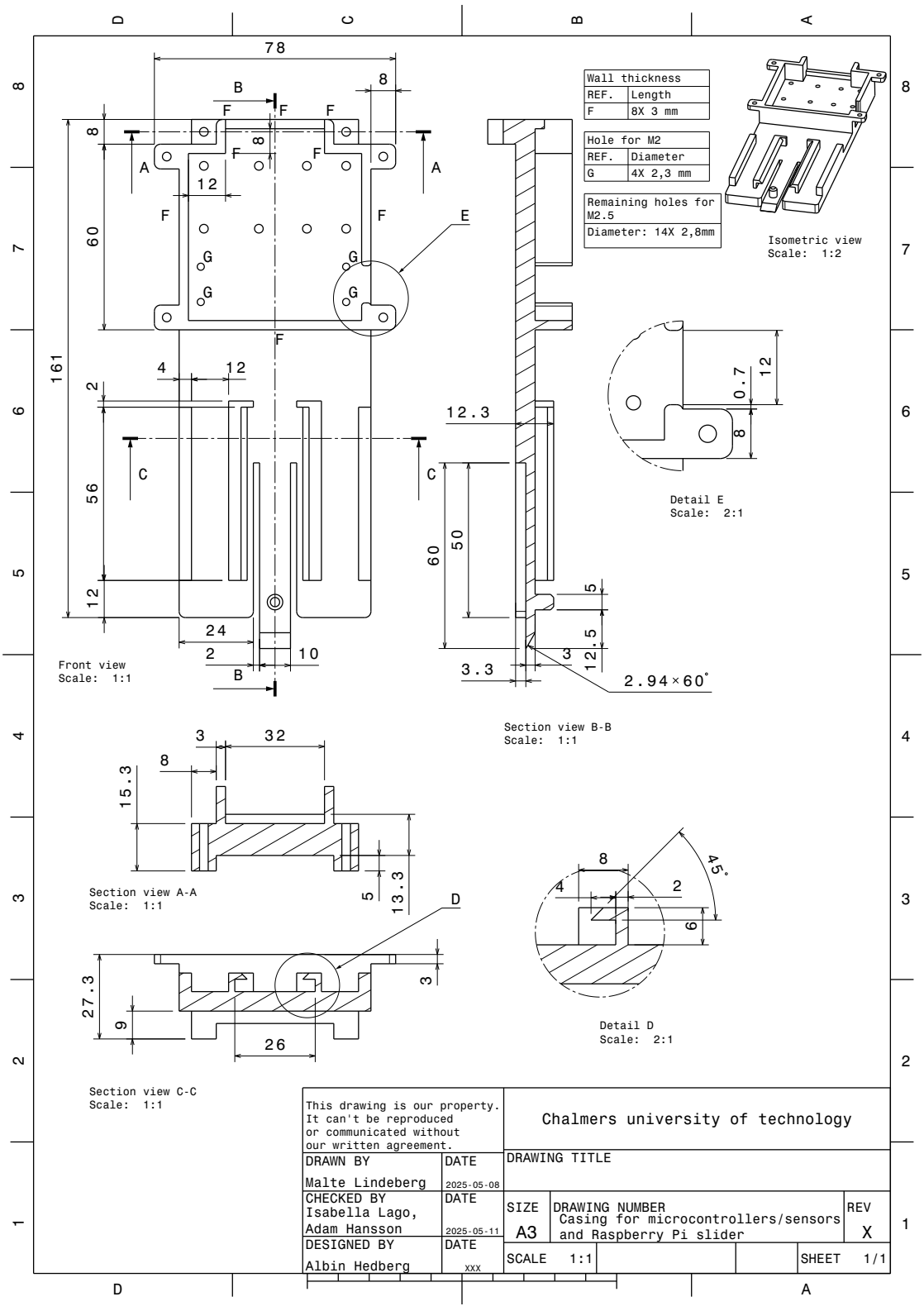


Figure A.7: Microcontroller/sensors casing CAD drawing.

A.8 Technical drawing of embedded electronics casing lid

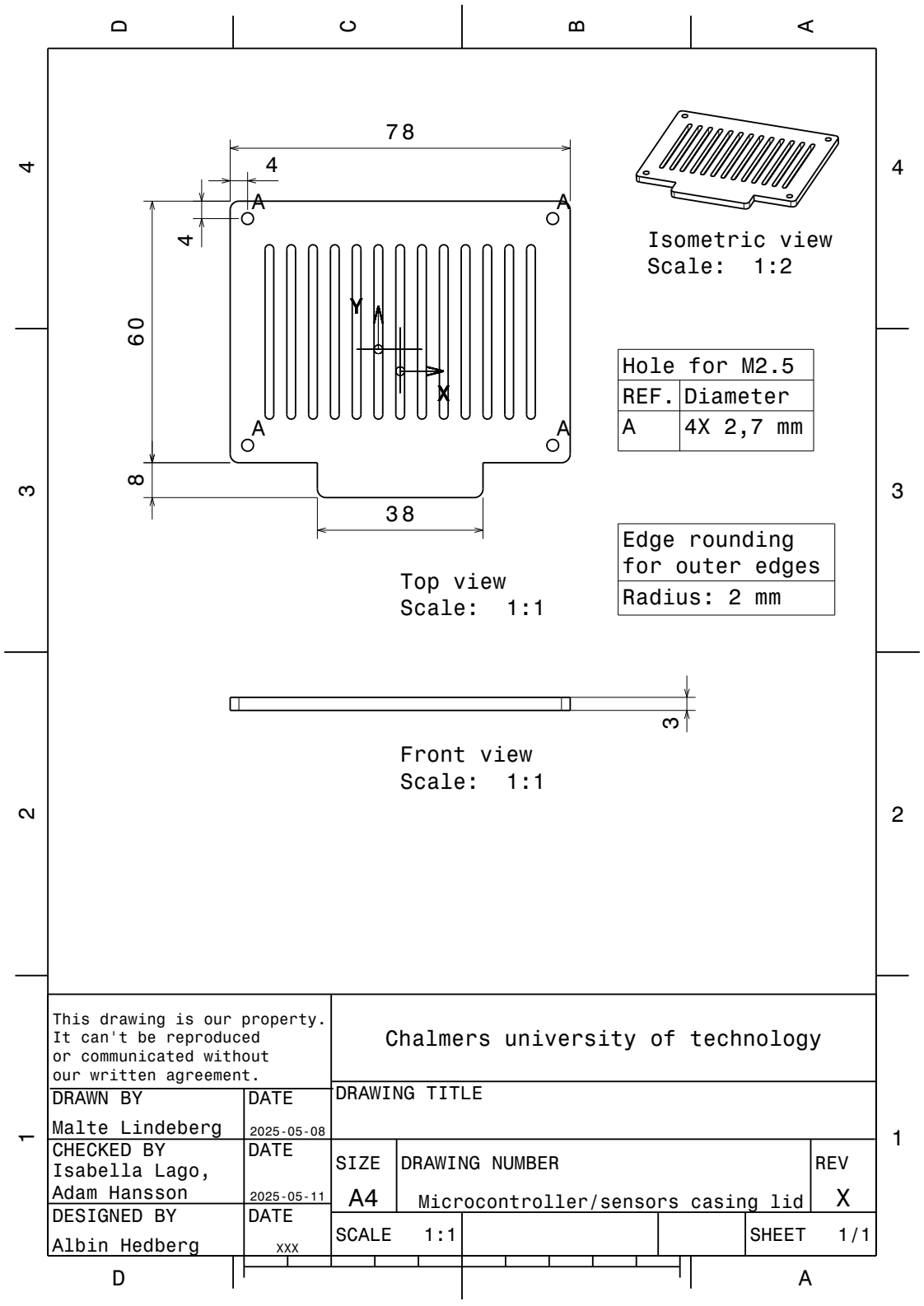


Figure A.8: Microcontroller/sensors casing lid CAD drawing.

A.9 Technical drawing of clamping piece

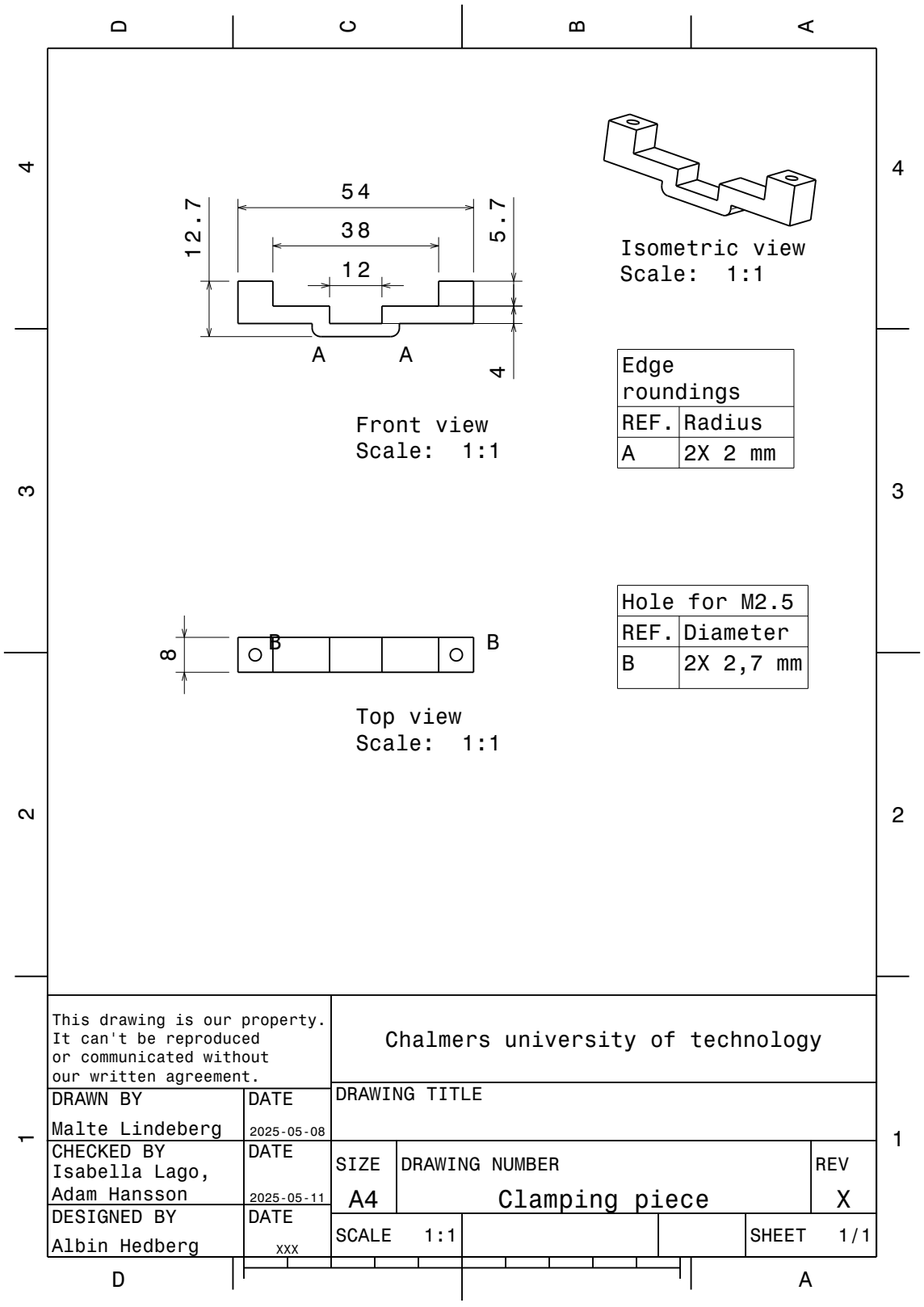


Figure A.9: Clamping piece CAD drawing.

Appendix B: Code

B.1 Motor Node

```
1  #!/usr/bin/env python3
2  import rclpy
3  from rclpy.node import Node
4  from rclpy.parameter import Parameter
5  from rcl_interfaces.msg import SetParametersResult
6  from myactuator_rmd import myactuator_rmd_py as rmd
7  from exo_interfaces.msg import MotorStatus
8  from exo_interfaces.srv import SetAbsPos
9  from exo_interfaces.srv import StopMotor
10 from exo_interfaces.srv import SetZeroPos
11 from exo_interfaces.srv import SetTorque
12 from threading import Lock
13
14
15 class MotorNode(Node):
16     def __init__(self):
17         super().__init__("motor_node")
18
19         self.driver = None
20         self.actuator = None
21         self.can_lock = Lock()
22         self.motor_connected = False
23         self.torque_constant = 6.92
24
25         self.declare_parameter("motor_status_pub_rate",
               10) #Default value is 10Hz
26
27         self.add_on_set_parameters_callback(self.
               parameter_callback)
28
29         self.publisher = self.create_publisher(
               MotorStatus, "motor_status", 10)
30         self.publish_fail_count = 0
```

```
31
32          self.motor_status_timer = None
33          self.reconnect_timer = self.create_timer(2, self.
               try_connect_motor)
34
35          self.server_1 = self.create_service(SetAbsPos, "
               set_abs_pos", self.callback_set_abs_pos)
36          self.server_2 = self.create_service(StopMotor, "
               stop_motor", self.callback_stop_motor)
37          self.server_3 = self.create_service(SetZeroPos, "
               set_zero_pos", self.callback_set_zero_pos)
38          self.server_4 = self.create_service(SetTorque, "
               set_torque", self.callback_set_torque)
39
40      def try_connect_motor(self):
41          if not self.motor_connected:
42              try:
43                  driver = rmd.CanDriver("can0")
44                  actuator = rmd.ActuatorInterface(driver,
                       1)
45                  model = actuator.getMotorModel()
46
47                  self.driver = driver
48                  self.actuator = actuator
49                  self.motor_connected = True
50                  rate = self.get_parameter("
                       motor_status_pub_rate").value
51                  self.reset_motor_status_timer(rate)
52                  self.get_logger().info("Succesfully␣
                       connected␣to␣motor:␣" + model)
53
54              except Exception as e:
55                  self.get_logger().warn(f"Motor␣connection
                       ␣failed␣{e}.␣Reconnecting...")
56
57      def disconnect_motor(self):
58          self.motor_connected = False
59          self.driver = None
60          self.actuator = None
61          self.get_logger().warn("Motor␣connection␣lost.␣
               Attempting␣to␣reconnect...")
62
63      def reset_motor_status_timer(self, rate):
64          interval = 1.0 / rate
65
66          if self.motor_status_timer:
```

```
67              self.motor_status_timer.cancel()
68
69          self.motor_status_timer = self.create_timer(
                interval, self.publish_motor_status)
70          self.get_logger().info(f"Updated publishing rate
                to {rate} Hz.")
71
72      def parameter_callback(self, params):
73          for param in params:
74              if param.name == "motor_status_pub_rate" and
                    param.type_ == param.Type.INTEGER:
75                  try:
76                      rate = param.value
77                      if rate > 0:
78                          self.reset_motor_status_timer(
                                rate)
79                      else:
80                          self.get_logger().warn("
                                Publishing rate must be a
                                positive integer.")
81                          return SetParametersResult(
                                successful=False)
82
83                  except Exception as e:
84                      self.get_logger().error(f"Invalid
                            parameter value: {e}")
85                      return SetParametersResult(successful
                            =False)
86
87          return SetParametersResult(successful=True)
88
89      def publish_motor_status(self):
90          if not self.motor_connected:
91              return
92          try:
93              with self.can_lock:
94                  status = self.actuator.getMotorStatus2()
95              msg = MotorStatus()
96              msg.temperature = float(status.temperature)
97              msg.torque = float(status.current/self.
                    torque_constant)
98              msg.speed = float(status.shaft_speed)
99              msg.angle = float(status.shaft_angle)
100             self.publisher.publish(msg)
101
102             self.publish_fail_count = 0
```

```
103
104             except Exception as e:
105                 error_message = str(e)
106
107                 if "Resource␣temporarily␣unavailable" in
                        error_message:
108                     self.get_logger().warn("CAN␣interface␣
                            busy,␣skipping␣this␣publish␣cycle.")
109                     return
110
111                 self.publish_fail_count += 1
112                 self.get_logger().warn(f"Failed␣to␣publish␣
                        motor␣status␣({self.publish_fail_count}):␣
                        {e}")
113
114                 if self.publish_fail_count >= 3:
115                     self.get_logger().error(f"Maximum␣publish
                            ␣failures␣reached.␣Disconnecting␣motor
                            .")
116                     self.disconnect_motor()
117
118     def callback_set_abs_pos(self, request: SetAbsPos.
            Request, response: SetAbsPos.Response):
119         response.success = False
120
121         if self.motor_connected:
122             try:
123                 with self.can_lock:
124                     self.actuator.
                            sendPositionAbsoluteSetpoint(
                            request.position, request.
                            max_speed)
125                 response.success = True
126
127             except Exception as e:
128                 self.get_logger().error(f"Failed␣to␣set␣
                        motor␣absolute␣position:␣{e}")
129
130         else:
131             self.get_logger().warn("Cannot␣set␣position,␣
                    motor␣not␣connected")
132
133         return response
134
135     def callback_set_torque(self, request: SetTorque.
            Request, response: SetTorque.Response):
```

```
136         response.success = False
137
138         if self.motor_connected:
139             try:
140                 with self.can_lock:
141                     self.actuator.sendTorqueSetPoint(
                            request.torque, self.
                            torque_constant)
142                 response.success = True
143
144             except Exception as e:
145                 self.get_logger().error(f"Failed to send
                        torque setpoint: {e}")
146
147         else:
148             self.get_logger().warn("Cannot send torque
                    setpoint, motor not connected.")
149
150         return response
151
152     def callback_stop_motor(self, request: StopMotor.
            Request, response: StopMotor.Response):
153         response.success = False
154         if self.motor_connected:
155             try:
156                 with self.can_lock:
157                     self.actuator.shutdownMotor()
158                 response.success = True
159             except Exception as e:
160                 self.get_logger().error(f"Failed to stop
                        motor: {e}")
161
162         else:
163             self.get_logger().warn("Cannot stop motor,
                    motor not connected.")
164
165         return response
166
167
168     def callback_set_zero_pos(self, request: SetZeroPos.
            Request, response: SetZeroPos.Response):
169         response.success = False
170         if self.motor_connected:
171             try:
172                 with self.can_lock:
173                     self.actuator.
```

```
                        setCurrentPositionAsEncoderZero()
174                     self.actuator.reset()
175                 response.success = True
176
177             except Exception as e:
178                 self.get_logger().error(f"Failed to set
                        motor encoder zero position: {e}")
179
180         else:
181             self.get_logger().warn("Cannot set zero
                position, motor not connected")
182
183         return response
184
185
186 def main(args=None):
187     rclpy.init(args=args)
188     node = MotorNode()
189     rclpy.spin(node)
190     rclpy.shutdown()
191
192
193 if __name__ == "__main__":
194     main()
```

B.2 Pico 2 Node

```
1  #include <Arduino.h>
2  #include <AMT25.h>
3  #include <Adafruit_BNO08x.h>
4  #include <Adafruit_ADS1X15.h>
5  #include <Wire.h>
6
7  #include <micro_ros_platformio.h>
8  #include <stdio.h>
9  #include <rcl/rcl.h>
10 #include <rcl/error_handling.h>
11 #include <rclc/rclc.h>
12 #include <rclc/executor.h>
13 #include <rclc_parameter/rclc_parameter.h>
14 #include <rmw_microros/rmw_microros.h>
15 #include <std_msgs/msg/u_int16.h>
16 #include <geometry_msgs/msg/vector3.h>
17 #include <std_msgs/msg/float32.h>
18 #include <std_srvs/srv/trigger.h>
19
```

```
20 rcl_node_t node;
21
22 rclc_executor_t executor;
23 rcl_allocator_t allocator;
24 rclc_support_t support;
25 rcl_timer_t timer;
26 bool micro_ros_init_successful;
27
28 rcl_publisher_t encoder_pub;
29 std_msgs__msg__UInt16 encoder_msg;
30
31 rcl_publisher_t imu_pub;
32 geometry_msgs__msg__Vector3 imu_msg;
33
34 rcl_publisher_t force_upper_pub;
35 std_msgs__msg__Float32 force_upper_msg;
36
37 rcl_publisher_t force_lower_pub;
38 std_msgs__msg__Float32 force_lower_msg;
39
40 rcl_service_t set_ext_encoder_zero_service;
41 std_srvs__srv__Trigger_Request encoder_zero_req;
42 std_srvs__srv__Trigger_Response encoder_zero_res;
43
44 rcl_service_t set_imu_offset_service;
45 std_srvs__srv__Trigger_Request set_imu_offset_req;
46 std_srvs__srv__Trigger_Response set_imu_offset_res;
47
48 rclc_parameter_server_t param_server;
49 int publish_rate_hz = 10; //Publishing rate in Hz.
50
51 //Setting encoder SPI CS PIN GPIO 17
52
53 AMT25 encoder(17);
54
55 //Force sensor variables
56
57 Adafruit_ADS1015 ads;
58 const float m = 1.0;  //calibration
59 const float b = 0.0;
60
61 const int windowSize = 5;  //smoothing variables
62 float readings_upper[windowSize];
63 float readings_lower[windowSize];
64 int bufferIndex = 0;
65 float sum_upper = 0;
```

```
66  float sum_lower = 0;
67  bool bufferFilled = false;
68
69  //IMU variables
70
71  Adafruit_BNO08x bno08x(-1);
72  sh2_SensorValue_t sensorValue;
73
74  sh2_SensorId_t reportType = SH2_ARVR_STABILIZED_RV;
75  long reportIntervalUs = 1000000 / publish_rate_hz;
76
77  struct euler_t {
78    float yaw;
79    float pitch;
80    float roll;
81  } ypr;
82
83  struct euler_offset_t {
84    float yaw_offset = 0;
85    float pitch_offset = 0;
86    float roll_offset = 0;
87  } imu_offset;
88
89  #define LED_PIN LED_BUILTIN
90
91  #define RCCHECK(fn) { rcl_ret_t temp_rc = fn; if((temp_rc
        != RCL_RET_OK)){return false;} }
92
93  #define EXECUTE_EVERY_N_MS(MS, X)  do { \
94    static volatile int64_t init = -1; \
95    if (init == -1) { init = uxr_millis();} \
96    if (uxr_millis() - init > MS) { X; init = uxr_millis()
        ;} \
97  } while (0)\
98
99  enum states {
100   WAITING_AGENT,
101   AGENT_AVAILABLE,
102   AGENT_CONNECTED,
103   AGENT_DISCONNECTED
104 } state;
105
106 void quaternionToEuler(float qr, float qi, float qj,
       float qk, euler_t* ypr, bool degrees = false) {
107   float sqr = sq(qr);
108   float sqi = sq(qi);
```

```
109   float sqj = sq(qj);
110   float sqk = sq(qk);
111
112   ypr->yaw = atan2(2.0f * (qi * qj + qk * qr), (sqi - sqj
          - sqk + sqr));
113   ypr->pitch = asin(-2.0f * (qi * qk - qj * qr) / (sqi +
         sqj + sqk + sqr));
114   ypr->roll = atan2(2.0f * (qj * qk + qi * qr), (-sqi -
         sqj + sqk + sqr));
115
116   if (degrees) {
117     ypr->yaw *= RAD_TO_DEG;
118     ypr->pitch *= RAD_TO_DEG;
119     ypr->roll *= RAD_TO_DEG;
120   }
121 }
122
123 void quaternionToEulerRV(sh2_RotationVectorWAcc_t* rv,
       euler_t* ypr, bool degrees = false) {
124   quaternionToEuler(rv->real, rv->i, rv->j, rv->k, ypr,
         degrees);
125 }
126
127 float wrap180(float angle) {
128   if (angle > 180) angle -= 360;
129   else if (angle < -180) angle += 360;
130   return angle;
131 }
132
133 void encoder_zero_callback(const void * req, void * res)
       {
134   (void)req;
135   std_srvs__srv__Trigger_Response * res_in = (
         std_srvs__srv__Trigger_Response *) res;
136
137   encoder.setZero();
138
139   res_in->success = true;
140 }
141
142 void imu_offset_callback(const void * req, void * res) {
143   (void)req;
144   std_srvs__srv__Trigger_Response * res_in = (
         std_srvs__srv__Trigger_Response *) res;
145
146   imu_offset.yaw_offset = ypr.yaw;
```

```
147   imu_offset.pitch_offset = ypr.pitch;
148   imu_offset.roll_offset = ypr.roll;
149
150   res_in->success = true;
151 }
152
153 bool on_parameter_changed(const Parameter *old_param,
      const Parameter *new_param, void *context) {
154   (void) old_param; (void) context;
155
156   if (new_param == NULL) return false;
157
158   if (strcmp(new_param->name.data, "publish_rate_pico2")
        == 0 && new_param->value.type == RCLC_PARAMETER_INT)
         {
159
160     publish_rate_hz = new_param->value.integer_value;
161     int64_t old_period;
162
163     RCCHECK(rcl_timer_exchange_period(&timer,
          RCL_MS_TO_NS(1000 / publish_rate_hz), &old_period)
          );
164
165     reportIntervalUs = 1000000 / publish_rate_hz;
166     bno08x.enableReport(reportType, reportIntervalUs);
167
168     return true;
169   }
170
171   return false;
172 }
173
174 void timer_callback(rcl_timer_t * timer, int64_t
      last_call_time)
175 {
176   RCLC_UNUSED(last_call_time);
177   if (timer != NULL) {
178     //Encoder
179     encoder_msg.data = encoder.getAngle();
180
181     //Force sensors
182     int16_t raw_upper = ads.readADC_SingleEnded(0); //
          Upper force sensor on channel 0
183     int16_t raw_lower = ads.readADC_SingleEnded(3); //
          Lower force sensor on channel 3
184     float force_upper = m * raw_upper + b;
```

```
185     float force_lower = m * raw_lower + b;
186
187     // Smoothing
188     sum_upper -= readings_upper[bufferIndex];
189     sum_lower -= readings_lower[bufferIndex];
190     readings_upper[bufferIndex] = force_upper;
191     readings_lower[bufferIndex] = force_lower;
192     sum_upper += readings_upper[bufferIndex];
193     sum_lower += readings_lower[bufferIndex];
194     bufferIndex = (bufferIndex + 1) % windowSize;
195     if (!bufferFilled && bufferIndex == 0) bufferFilled =
            true;
196
197     float smoothedForce_upper = bufferFilled ? sum_upper
          / windowSize : sum_upper / (bufferIndex + 1);
198     float smoothedForce_lower = bufferFilled ? sum_lower
          / windowSize : sum_lower / (bufferIndex + 1);
199
200     force_upper_msg.data = smoothedForce_upper;
201     force_lower_msg.data = smoothedForce_lower;
202
203     //IMU
204     if (bno08x.wasReset()) {
205       bno08x.enableReport(reportType, reportIntervalUs);
206     }
207
208     if (bno08x.getSensorEvent(&sensorValue)) {
209       if (sensorValue.sensorId == SH2_ARVR_STABILIZED_RV)
              {
210         quaternionToEulerRV(&sensorValue.un.
              arvrStabilizedRV, &ypr, true);
211       }
212     }
213
214     imu_msg.x = wrap180(ypr.roll - imu_offset.roll_offset
          ); //Roll is rotation around x-axis
215     imu_msg.y = wrap180(ypr.pitch - imu_offset.
          pitch_offset); //Pitch is rotation around y-axis
216     imu_msg.z = wrap180(ypr.yaw - imu_offset.yaw_offset);
            //Yaw is rotation around z-axis
217
218     //publishing
219     rcl_publish(&encoder_pub, &encoder_msg, NULL);
220     rcl_publish(&force_upper_pub, &force_upper_msg, NULL)
          ;
221     rcl_publish(&force_lower_pub, &force_lower_msg, NULL)
```

```
        ;
222     rcl_publish(&imu_pub, &imu_msg, NULL);
223   }
224 }
225
226 bool create_entities()
227 {
228   allocator = rcl_get_default_allocator();
229   RCCHECK(rclc_support_init(&support, 0, NULL, &allocator
        ));
230   RCCHECK(rclc_node_init_default(&node, "pico2_node", "",
        &support));
231
232   RCCHECK(rclc_publisher_init_default(
233     &encoder_pub,
234     &node,
235     ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, UInt16),
236     "ext_encoder_pos"
237   ));
238
239   RCCHECK(rclc_publisher_init_default(
240     &imu_pub,
241     &node,
242     ROSIDL_GET_MSG_TYPE_SUPPORT(geometry_msgs, msg,
          Vector3),
243     "imu_data"
244   ));
245
246   RCCHECK(rclc_publisher_init_default(
247     &force_upper_pub,
248     &node,
249     ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, Float32),
250     "force_upper_data"
251   ));
252
253   RCCHECK(rclc_publisher_init_default(
254     &force_lower_pub,
255     &node,
256     ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, Float32),
257     "force_lower_data"
258   ));
259
260   RCCHECK(rclc_service_init_default(
261     &set_ext_encoder_zero_service,
262     &node,
263     ROSIDL_GET_SRV_TYPE_SUPPORT(std_srvs, srv, Trigger),
```

```
264       "/set_ext_encoder_zero_service"
265     ));
266
267     RCCHECK(rclc_service_init_default(
268       &set_imu_offset_service,
269       &node,
270       ROSIDL_GET_SRV_TYPE_SUPPORT(std_srvs, srv, Trigger),
271       "/set_imu_offset_service"
272     ));
273
274     RCCHECK(rclc_timer_init_default(
275       &timer,
276       &support,
277       RCL_MS_TO_NS(1000 / publish_rate_hz),
278       timer_callback
279     ));
280
281     RCCHECK(rclc_parameter_server_init_default(&
          param_server, &node));
282
283     executor = rclc_executor_get_zero_initialized_executor
          ();
284     rclc_executor_init(&executor, &support.context,
          RCLC_EXECUTOR_PARAMETER_SERVER_HANDLES + 3, &
          allocator);
285     rclc_executor_add_timer(&executor, &timer);
286     rclc_executor_add_service(&executor, &
          set_ext_encoder_zero_service, &encoder_zero_req, &
          encoder_zero_res, encoder_zero_callback);
287     rclc_executor_add_service(&executor, &
          set_imu_offset_service, &set_imu_offset_req, &
          set_imu_offset_res, imu_offset_callback);
288     rclc_executor_add_parameter_server(&executor, &
          param_server, on_parameter_changed);
289
290     RCCHECK(rclc_add_parameter(&param_server, "
          publish_rate_pico2", RCLC_PARAMETER_INT));
291     rclc_parameter_set_int(&param_server, "
          publish_rate_pico2", publish_rate_hz);
292     rclc_add_parameter_description(&param_server, "
          publish_rate_pico2", "Publish␣rate␣parameter␣for␣
          Pico2␣publisher␣(Default␣10␣Hz)", "Unit:␣Hz");
293
294     return true;
295 }
296
```

```
297 void destroy_entities()
298 {
299   rmw_context_t * rmw_context =
        rcl_context_get_rmw_context(&support.context);
300   (void)
        rmw_uros_set_context_entity_destroy_session_timeout(
        rmw_context, 0);
301
302   rcl_publisher_fini(&encoder_pub, &node);
303   rcl_publisher_fini(&force_upper_pub, &node);
304   rcl_publisher_fini(&force_lower_pub, &node);
305   rcl_publisher_fini(&imu_pub, &node);
306
307   rcl_service_fini(&set_ext_encoder_zero_service, &node);
308   rcl_service_fini(&set_imu_offset_service, &node);
309
310   rclc_parameter_server_fini(&param_server, &node);
311
312   rcl_timer_fini(&timer);
313   rclc_executor_fini(&executor);
314   rcl_node_fini(&node);
315   rclc_support_fini(&support);
316 }
317
318 void setup()
319 {
320   Serial.begin(115200);
321   delay(1000);
322   set_microros_serial_transports(Serial);
323
324   SPI.begin();
325   encoder.begin(14);
326
327   Wire.setSDA(4);
328   Wire.setSCL(5);
329   Wire.begin();
330
331   ads.begin(0x48, &Wire);
332   ads.setGain(GAIN_ONE);
333
334   for (int i = 0; i < windowSize; i++){
335     readings_upper[i] = 0;
336     readings_lower[i] = 0;
337   }
338
339   bno08x.begin_I2C();
```

```
340
341   bno08x.enableReport(reportType, reportIntervalUs);
342
343   pinMode(LED_PIN, OUTPUT);
344
345   state = WAITING_AGENT;
346 }
347
348 void loop()
349 {
350   switch (state) {
351     case WAITING_AGENT:
352       EXECUTE_EVERY_N_MS(500, state = (RMW_RET_OK ==
             rmw_uros_ping_agent(100, 1)) ? AGENT_AVAILABLE :
              WAITING_AGENT;);
353       break;
354     case AGENT_AVAILABLE:
355       state = (true == create_entities()) ?
             AGENT_CONNECTED : WAITING_AGENT;
356       if (state == WAITING_AGENT) {
357         destroy_entities();
358       };
359       break;
360     case AGENT_CONNECTED:
361       EXECUTE_EVERY_N_MS(200, state = (RMW_RET_OK ==
             rmw_uros_ping_agent(100, 1)) ? AGENT_CONNECTED :
              AGENT_DISCONNECTED;);
362       if (state == AGENT_CONNECTED) {
363         rclc_executor_spin_some(&executor, RCL_MS_TO_NS
              (100));
364       }
365       break;
366     case AGENT_DISCONNECTED:
367       destroy_entities();
368       state = WAITING_AGENT;
369       break;
370     default:
371       break;
372   }
373
374   if (state == AGENT_CONNECTED) {
375     digitalWrite(LED_PIN, 1);
376   } else {
377     digitalWrite(LED_PIN, 0);
378   }
379 }
```

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS