MUSE

Muscular Signal Processing for Exoskeleton Control

Authors

Simon Vinkel — `sivin11`

Thomas Therkelsen — `ththe20`

Supervisors

Dr. Xiaofeng Xiong & Dr. Zhuoqi Cheng

SYDDANSK UNIVERSITET

B.Eng. in Robot Systems

TEK MMMI

University of Southern Denmark

January 2nd 2024

Abstract

This project presents a sensory augmentation-based extension of Xiong *et al.*'s POW-EXO assist-and-resist-as-needed (AAN and RAN) exoskeleton project [1]. POW-EXO is a lightweight exoskeleton with anticipatory and variable-compliant joint control based on impedance adaptation and online iterative learning. Exoskeletons, such as the POW-EXO, serve an important role in boosting human capacities, offering assistance for physical activities, and assisting people with mobility issues in both their daily lives and in potential rehabilitation settings. This technology offers a promising innovation with potential uses in healthcare, rehabilitation, and a variety of sectors, leading to greater productivity and, more importantly, overall quality of life.

This project seeks to address a few shortcomings in the exoskeleton: 1) It cannot switch between AAN and RAN in real time, so the user must select one of the control loops at startup. 2) It can not determine user intent, that is, whether they want to bend their arm or not. This is where the sensory enhancement will be useful, both for switching between control loops and identifying intent when in specific loops.

This study looks at the advantages and disadvantages of several muscle sensing methods, including surface electromyography (sEMG), electrical impedance myography (EIM), and electrical impedance tomography (EIT). Furthermore, as a proposed addition to POW-EXO, To achieve this goal, human-machine interfacing (HMI) is investigated and integrated with the chosen sensor type(s) by developing a sensor interface that collects, filters, and processes the intramuscular data from the sensors in a way that makes it interpretable to a Gated Recurrent Units (GRU) network that will be developed with the ability to predict user intent and integrating it with the exoskeleton to provide both a method to seamlessly transition between AAN and RAN.

***Keywords*—** Exoskeleton, Human-Machine Interface, Intent Determination, Digital Signal Processing, GRU Network, Neuro/Physiological Conditions, Quality of Life Improvement

Preface

Special Thanks

Special thanks go to:

- Dr. Xiaofeng Xiong & Dr. Zhuoqi Cheng for expert supervision during the entire project.
- Dr. Xiaofeng Xiong for letting us work on his previously developed exoskeleton, POW-EXO.
- Dr. Zhuoqi Cheng for supplying us with the Eliko device and interfacing software, along with the EIM electrodes to connect to the device.

Work Distribution

The work during the project has been distributed as follows:

Simon Vinkel:

- Digital Signal Processing (DSP) Unit: Design and implementation of the EIM signal filter.
- Sensor Sleeve: Designing, measuring, cutting, and sewing.
- Kinematics Angle Tracker: Configuration of an open-source MoCap system for obtaining elbow angles throughout the experiment without the use of motion capture markers.
- GRU Network: data manipulation, feature engineering, architecture design, training, and adjusting the model used to predict the user's elbow angle and mass.

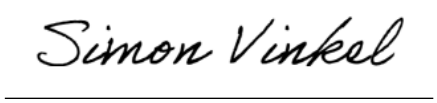
Thomas Therkelsen:

- Software design and architectural philosophy that reduces code complexity while ensuring high levels of maintainability and extendability.
- Development of a C++ interface for the Eliko Quadra device: For collection and processing of electrical bioimpedance data from the EIM electrodes.
- Development of Python Data Visualizer: For data visualization and sanity checks during various data gathering, network training, and network testing experiments.
- Creation of keyboard macros: To achieve time synchronicity between data sets by synchronously starting and ending both data gathering processes.

Both:

- Data pre- and post-processing.
- Interfacing between the GRU network and POW-EXO.
- Report writing.

Simon Chris Vinkel



Thomas Therkelsen

Table of Contents

1 Introduction

A wide range of neurological and physiological illnesses impede upper-body mobility, either by interfering with nerve signals or by reducing skeletal muscle mass. Amyotrophic Lateral Sclerosis (ALS), Parkinson's disease, muscular dystrophy, muscular atrophy, spinal cord injuries, multiple sclerosis, and other comparable conditions exacerbate people's daily struggles.

Over time, many movement-assist exoskeletons have been developed to improve the quality of life (QoL) for individuals suffering from mobility-impeding conditions. One such example is the POW-EXO project, which was intended to assist both patients undergoing rehabilitation and sportsmen in preventing injuries [1]. Because of its relatively simple control loop, which acts on both angle and torque data obtained from an encoder incorporated into the actuator, the exoskeleton can provide assistance (AAN) and resistance (RAN).

In this project, a complete hardware and software system has been developed to collect and analyze data, predict user intent from the data, and control the exoskeleton via its interface using predictions of user intent. The system is made up of the following components:

Hardware:

- Eliko Quadra Impedance Spectroscopy Device with Single Shunt Front End attached [2].
- A custom-made sensor sleeve with four EIM electrodes is attached to the Eliko device.
- The POW-EXO exoskeleton [1].
- A PC running Windows 10/11 is needed to run all the software.

Software:

- Eliko sensor data interface with built-in DSP and statistical post-processing functions for the data.
- Kinematic Angle Tracker, to track elbow angles using computer-vision-based human pose estimation.
- GRU Neural Network that predicts user intent from the processed electrical bioimpedance (BI) data.
- A visualizer for both electrical bioimpedance (BI) and angle data.
- A modified POW-EXO interface with the changes needed to accommodate our data structures.

1.1 State of the Art

This section presents the current state of the art within exoskeletons and electrical muscle-sensor systems.

1.1.1 Real-Time EEG-EMG Human-Machine Interface-Based Control System for a Lower-Limb Exoskeleton [3]

This research presents a rehabilitation technique that employs a lower-limb exoskeleton and a motor imagery-based human-machine interface (HMI). Motor Imagery (MI) Electroencephalograms (EEG) are self-regulated EEG's without an external stimulus, which can be detected by electrodes [4]. They are consistent with changes caused by actual exercise in the motor cortex region [4]. In this context, MI EEG refers to the brain's electrical activity related to the mental imagination of feet and leg movements, along with which leg muscle EMG is used to control the exoskeleton. The exoskeleton is seen in Figure 1.

The term "multi-modal signals" refers to signals that comprise components presented to the same receiver across two or more sensory modalities [5]. Multi-modal signals refer to the combination of MI EEG, and EMG data.

This study demonstrates enhanced accuracy, performance, and dependability when employing multi-modal signals to operate the exoskeleton on healthy participants. Notably, the system supports online real-time operation.

Furthermore, the efficacy of a few EEG electrodes for dependable online control is investigated. It tackles the limits of using EEG or EMG inputs for movement prediction alone, offering a multi-modal method to improve reliability and safety in lower-limb exoskeleton operation.

Figure 1: EEG–EMG controlled lower-limb exoskeleton.

Finally, it proposes possible implications for clinical rehabilitation procedures, with treatments based on neuromuscular disorders being emphasized. Adaptive feedback during rehabilitation and investigating transcranial magnetic stimulation to improve control signal conduction are two future research approaches.

1.1.2 Assistive upper-limb MMI-controlled exoskeleton for severely impaired patients [6]

The exoskeleton, designed for individuals with muscular dystrophy, aims to enhance independence and QoL. Offering direct user control through a sensitive joystick or voice interface, it specifically addresses the needs of severely handicapped patients with limited upper limb muscle force.

Featuring a kinematic model supporting differential inverse kinematics, the system allows users precise control of movements. The exoskeleton is seen in Figure 2. A pilot study with 14 patients with muscular dystrophy demonstrated significant improvements in range of motion and functional benefits, both externally assessed and self-perceived.

Notably, the Bridge exoskeleton stands out as it was purpose-built and tested on severely disabled individuals. The dynamic model optimization focuses on reducing torque requirements, with an antigravity system decreasing actuator torque needs by 46%.

Comparative studies found in the full paper [6] highlight superior functional improvements compared to passive devices. The device proves effective in improving arm mobility, aiding daily tasks, and reducing discomfort from extended postures. User feedback underscores the importance of aesthetics and compactness for enhanced wheelchair compatibility.

In conclusion, the Bridge exoskeleton provides a user-controlled, fully active solution for individuals with muscular dystrophy, emphasizing the significance of upper-limb assistive devices in improving independence and quality of life. Future research should target improvements in actuation units, antigravity systems, and portable power supplies for broader real-world applications.

Figure 2: MMI-controlled assistive upper-limb exoskeleton.

1.1.3 Volitional control of upper-limb exoskeleton empowered by EMG and ML [7]

In bionic assistive robot volitional motion control, machine learning (ML) is applied to interpret many channels of bioelectrical inputs in real-time while addressing issues with noise, artifacts, and individual biovariability. This research focuses on using ML computation to interpret twelve channels of myoelectrical inputs from the shoulder and upper limbs for shoulder motion pattern recognition and real-time upper arm exoskeleton volitional control.

The paper compares the efficiency of three machine learning (ML) techniques for EMG signal processing: support vector machine (SVM), artificial neural network (ANN), and logistic regression (LR). SVM surpasses the other algorithms in offline pattern recognition, obtaining 96% and 90% accuracy in real-time exoskeleton motion control.

The study develops a wearable sensor-controlled exoskeleton system employing ML-based computing, hence offering a platform for evaluating various ML algorithms. Despite limitations, such as the system's ability to detect just four motion patterns, the work shows that ML may be used to interpret EMG inputs for real-time motion identification and control. Future research will look at optimum sensor setups, adaptive motion control, and the effect of machine learning on exoskeleton responsiveness and accuracy.

The research provides important insights into the challenges of deploying an ML model on embedded systems. In the context of robotics, there are constraints in terms of system performance, accuracy, user experience, energy consumption, and processing speed. This also highlights the troubles with EMG sensoring, as it demands high precision in terms of sensory placement and knowledge of the muscular structure. The EMG signals are also noisy, which has an impact on a model's accuracy. An overview of the system is seen in Figure 3 below.

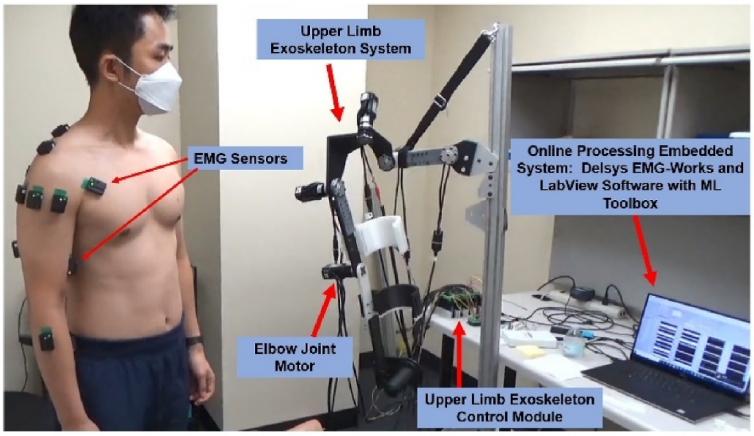


Figure 3: EMG and ML-controlled upper-limb exoskeleton.

1.1.4 A Wearable, Multi-Frequency Device to Measure Muscle Activity Combining Simultaneous Electromyography and Electrical Impedance Myography [8]

This research describes a unique system developed for simultaneous EMG and EIM to improve the robustness of muscle state monitoring. Interconnected EMG, EIM, microprocessor, and communication modules compose the modular, lightweight, wearable, and wireless system.

The EIM module's capacity to measure bioimpedance between 20 and 200 Ω with less than 5% error at $140\ sps$ (Samples Per Second) and a settling time of less than $1,000\ ms$ is a key result. The EMG module collects the EMG signal spectrum between 20 and 150 Hz at $1\ ksps$ with a signal-to-noise ratio (SNR) of 67 dB. Every $1\ ms$, the system achieves data transmission speeds of $500\ kbps$. Preliminary studies on a volunteer using simultaneous EMG and EIM during leg extension, walking, and sit-to-stand indicate the system's potential for examining muscle function.

For EMG module resilience against electromagnetic interference, critical elements include the necessity for high common-mode rejection ratio filters. For successful control, the study underlines the need to maintain an overall SNR of 30 dB or greater and a latency of less than $100\ ms$ between EMG and EIM signals.

The device performs well in muscular contraction during a variety of tests, demonstrating its use as a tool for investigating muscle states and force or torque. The modular architecture enables combinations and simultaneous multi-frequency bioimpedance measurements. Overall, the instrument offers a viable platform for studying muscle states and comprehending muscular force and torque. The setup is seen in Figure 4.

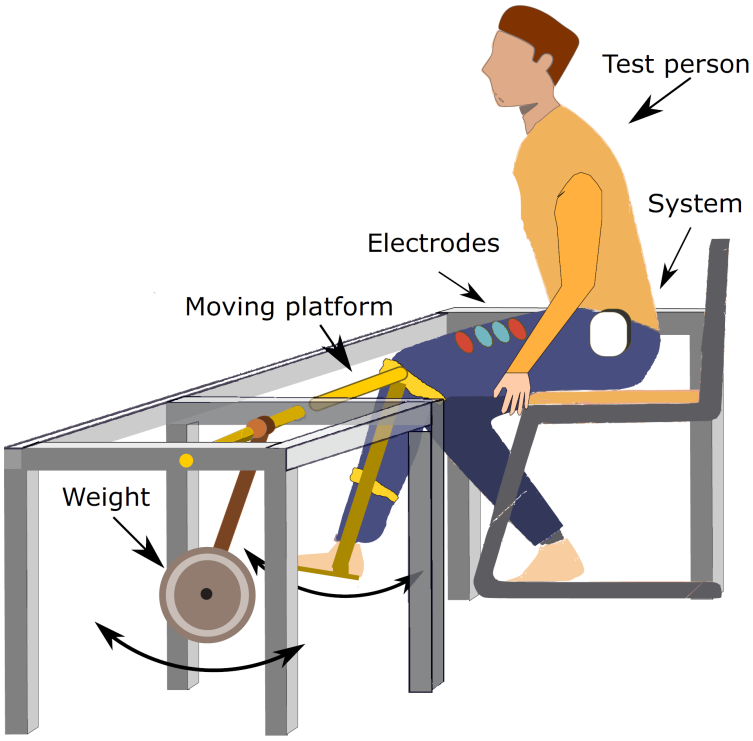


Figure 4: EIM-EMG Setup

1.2 Problem

The main problem to solve in this project is to further improve an already existing exoskeleton with the goal of using intramuscular signals to determine the intent of the exoskeleton wearer, from which to toggle between the aforementioned AAN and RAN modes of the exoskeleton, as well as provide references for the PD controller.

The sensor technologies that will be explored, and one of which will be chosen and implemented, are:

- Surface Electromyography (sEMG).
- Electrical Impedance Myography (EIM).

The development and integration of the MUSE system can be split into the following problems:

1. What are the relevant current-day use cases of sEMG and EIM?
2. What are the pros and cons of using either of the sensor technologies for the problem at hand?
3. Which types of data post-processing techniques are needed to optimize data reliability by maximizing accuracy and minimizing noise in the sensor data to predict and track user intention in real-time?
4. After post-processing the raw sensor data, how can it be transformed in order to use it in the exoskeleton's PD controller?
5. Which considerations are relevant, and why, when making an exoskeleton for end users with neurological, motor, and/or muscular deficiencies?

2 Sensor Technology

This section explores two kinds of sensor technologies in order to pick one to work with in the project.

2.1 Electromyography

Electromyography (EMG) is a technology that measures muscle contraction by outputting an electric signal. This signal is directly proportional to the degree of muscle contraction: the larger the tension amplitude recorded, the stronger the muscle contraction. This is due to the burst of electrical activity generated when a muscle contracts, which can be recorded from nearby tissue [9].

EMG is used to detect variations in current within the motor cortex. Here, brain activity signals the spinal cord, transmitting movement information to the relevant muscle via motor neurons. Upper motor neurons relay signals to lower motor neurons, which trigger muscle activity by innervating the muscle at the neuromuscular junction. This junction is a type of synapse where neuronal signals from the brain or spinal cord interact with skeletal muscle fibers, leading to their contraction [9] [10] [11].

Depolarization, a shift in electric charge distribution resulting in less negative charge inside the cell compared to the outside, is a key process in this interaction [12] [13]. This shift from a negative to a more positive membrane potential occurs during several processes, including an action potential [12].

Depolarization is closely linked to the electrochemical gradient (EG). This gradient, usually for an ion that can move across a membrane, consists of two parts: the chemical gradient (difference in solute concentration across a membrane) and the electrical gradient (difference in charge across a membrane) [14]. When there are unequal concentrations of an ion across a permeable membrane, the ion will move from the area of higher concentration to the area of lower concentration through simple diffusion [14]. This change in the EG is crucial for the function of many cells and the overall physiology of an organism [12].

The noise in EMG signals varies depending on the application. Estimating interaction torques from patient EMG can be challenging due to the irregular EMG-torque connections. EMG sensors can classify upper-limb motions, but the accuracy of the evaluation depends on the user's condition and the selection of the muscle group. Control methods based on EMG are ineffective in cases of tremor or spasticity [15] [16] [17] [18].

2.1.1 Surface Electromyography

Electromyography (EMG) sensors come in two primary types: intramuscular and surface (sEMG). The sEMG sensors, as the name suggests, are applied to the surface of the skin. They employ non-invasive technology, making the procedure painless and commonly used in clinics and sports medicine. While sEMG provides a quick and easy method for measuring EMG signals, it is limited to superficial muscles.

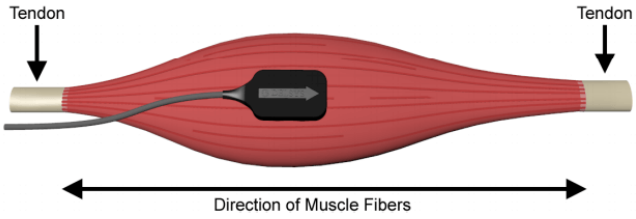


Figure 5: sEMG — Sensor placement (no skin).

The first step of the procedure involves placing the EMG sensors in the area innervating the two tendons on the skin's surface to enhance detection quality (Figure 5). Once in place, the electrodes start detecting electrical activity generated by muscle contractions or movements. [19]

2.1.2 Intramuscular Electromyography

In contrast to sEMG, intramuscular EMG involves the use of a sterile needle electrode to capture electrical activity within the muscle. This method provides valuable insights into muscle function and nerve communication. The needle electrode, which penetrates the skin to reach the target muscle tissue, ensures precise readings (Figure 6). Intramuscular EMG has high selectivity for individual motor unit action potentials and is thus used to measure motor unit activity. It offers a more detailed view of muscle activity, making it a powerful tool for diagnosing neuromuscular diseases and studying kinesiology and disorders of motor control. [19]

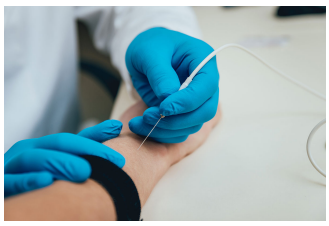


Figure 6: Intramuscular EMG — Needle Electrode.

2.2 Electrical Impedance Myography [20] [21]

Electrical impedance myography (EIM) is a technique that has evolved from estimating tissue volumes and evaluating skin lesions to assessing nerve and muscle conditions. It studies electrical bioimpedance, which is the electrical properties of tissues and body fluids, enabling disease monitoring by detecting changes in passive electrical properties. EIM measurements often use a four-electrode impedance method to minimize potential electrode artifacts. This involves passing alternating current through one pair of electrodes and measuring voltage at the second pair. Advanced approaches analyze voltage amplitude versus applied current to reveal tissue resistive behavior and capacitive/reactive elements. EIM can detect physiological changes like muscle relaxation and contraction. However, measurements beyond isometric contractions can be challenging due to changes in muscle or limb shape. Techniques, such as impedance imaging, have been developed to overcome these challenges. [20]

EIM has proven effective in differentiating between healthy and diseased muscles, as seen in diseases like ALS, SMA, radiculopathy, muscular dystrophy, and traumatic nerve injuries, but is also used for posture estimation. Its diagnostic accuracy becomes more pronounced as diseases progress. EIM has also shown promise in assessing specific conditions like carpal tunnel syndrome and Duchenne muscular dystrophy. Regarding electrode usage, EIM typically employs four electrodes, with silver or silver chloride electrodes being the most common choice. Alternative options include dry electrodes, which prevent inadvertent contact between adjacent electrodes, and textile dry electrodes for long-term EIM monitoring during daily activities. Further research is required to optimize electrode configurations, reduce the impact of subcutaneous fat, ensure effective current penetration into muscles, and capture muscle anisotropy. [21]

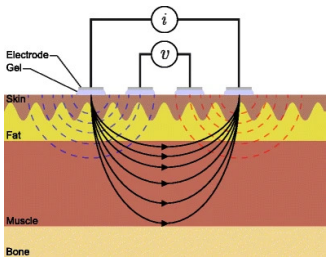


Figure 7: EIM — Four gel-adhesive electrodes to measure electrical impedance and illustrate the potential generated by current flow through tissue.

2.3 Sensor Choice

The decision to use EIM sensors in this project was based on extensive research comparing them with EMG sensors. EIM sensors were chosen primarily for their non-invasive nature, which ensures user comfort and provides stable signals unaffected by skin conditions. They are straightforward to set up and interpret, and they are suitable for long-term use. In contrast, EMG sensors require electrodes to be placed on or in muscles, which can cause discomfort, offer less signal stability, necessitate precise setup, and involve complex signal interpretation.

EIM sensors offer a non-invasive and more comfortable alternative, eliminating the need for skin-penetrating electrodes (see Figure 7). This non-intrusiveness enhances user acceptance and long-term usability. Additionally, EIM sensors have superior spatial resolution, enabling a comprehensive assessment of muscle activity over a larger area. This feature is crucial for capturing complex movements involving multiple muscle groups.

EIM sensors are also less prone to signal artifacts caused by changes in skin impedance or motion artifacts, resulting in a more reliable and accurate representation of the user's intent. The high-quality signal allows for finer control of the exoskeleton, leading to a more natural and intuitive human-machine interface.

In conclusion, the use of EIM sensors aligns with a user-centric approach that prioritizes comfort, accuracy, and ease of use, thereby optimizing human-machine interaction.

3 System Overview

This section explores the architecture, design philosophy, DSP, motion capture, exoskeleton integration, GRU network, and different concrete software components developed for use in this project.

3.1 Architecture

The software architecture has been planned out, with the main design strategy being to have each program take care of its own functionality and then interface with the other programs, as seen in Figure 8. This minimizes code complexity, ensuring high levels of maintainability and extendability, while maximizing modularity (aka the ity's), which helps keep the coupling between components as low as possible.

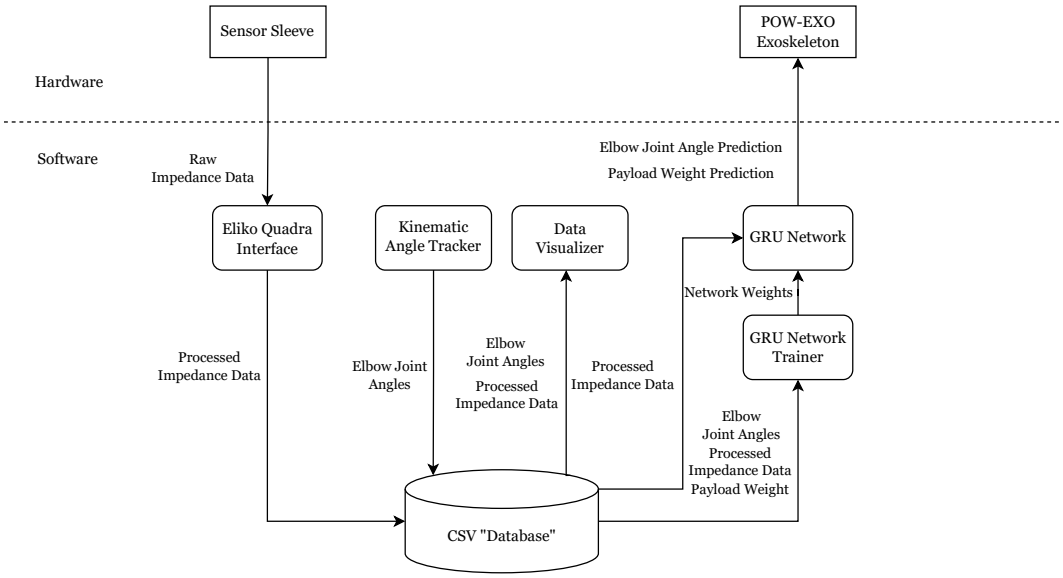


Figure 8: Diagram of Software Architecture.

At a more practical level, we strive to keep the main program file free from bloat, thereby enhancing its efficiency. This is achieved by dedicating it solely to the main status loop and state machine, while file system handling, data processing, and Eliko Quadra interfacing are refactored into their respective classes.

The EIM sensor sleeve, equipped with the Eliko Quadra device, connects and transmits raw electrical bioimpedance data to the CSV file, serving as the system's database. The angle tracker similarly records the joint angles in the database. The visualizer then plots the data collected by these two programs.

The GRU network is trained on all the data from the database to establish a relationship between electrical bioimpedance, elbow joint angle, and payload mass. With sufficient training data, the network should be capable of predicting both elbow joint and payload mass solely from the bioimpedance data. However, the predicted mass needs to be converted to torque, as the control system operates in rotational motion rather than translational motion.

Upon completion, the exoskeleton interface will monitor the converted predictions from the GRU network, effectively predicting and tracking the user's intent in real-time. For a comprehensive guide on setting up and using the MUSE system, please refer to the Experimental Protocol in Chapter 4.

3.2 EIM Sensor Interface

The decision has been made to employ a state-machine architecture for the MUSE program. This was selected for a number of reasons, including easier flow control between different stages in the code, better modularity and extendibility in the form of the ability to add or remove new states on the fly, and the flexibility to partition the code into "initializing", "running", and "shutdown" states for easier setup and tear-down of the sensor interface.

The program smoothly transitions between states, such as **IDLE** for awaiting user input, **OFFLINE** for reading and processing data from a file, **COLLECTING** for real-time data acquisition, **SAVING** for persisting collected data to CSV files, and **STOPPING** for resetting program variables. The state transitions are determined both by user input and by changing context of data collection. Finally, the application manages memory dynamically where needed.

The design of the interface fosters modularity, allowing for the easy integration of new functionalities or adaptations. This structured state machine design ensures a consistent and efficient flow through the program, while also providing flexibility and scalability in sensor interfacing and post-processing tasks. The state machine's flow through the MUSE program is seen in Figure 9.

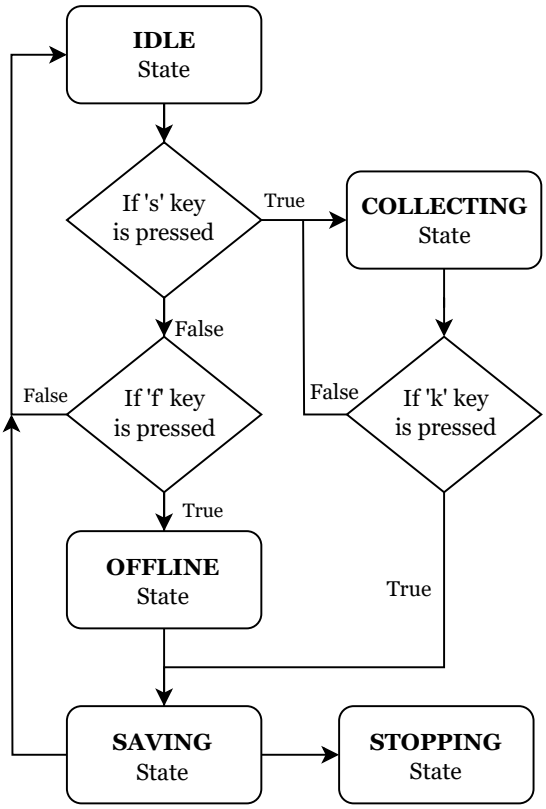


Figure 9: EIM Interface State Machine Flow.

3.3 Digital Signal Processing

This section introduces the filter design used in the signal preprocessing method. It first discusses the findings from the signal analysis, followed by the rationale behind the chosen filter design.

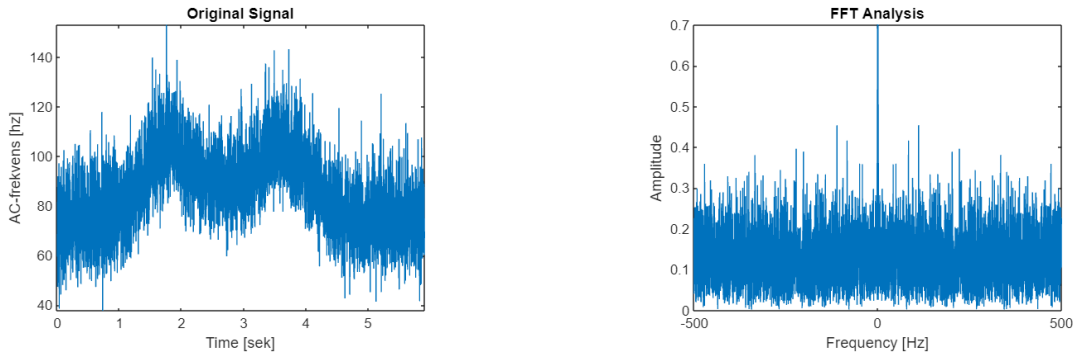
3.3.1 Fast Fourier Transformation Analysis

A Fast Fourier Transformation (FFT) was performed to identify the frequencies to be suppressed in the signal. The formula for conducting an FFT is described as follows [22]:

$$X(k) = \sum_{m=0}^{N-1} x(m) \cdot e^{-\frac{2\pi i}{N} km}, \quad k = 0, 1, \ldots, N-1 \tag{1}$$

Where $X(k)$ is the complex-valued FFR of the sequence $x(m)$ at frequency index k, $x(m)$ is the input sequence, N is the number of samples in the sequence, and i is the imaginary unit.

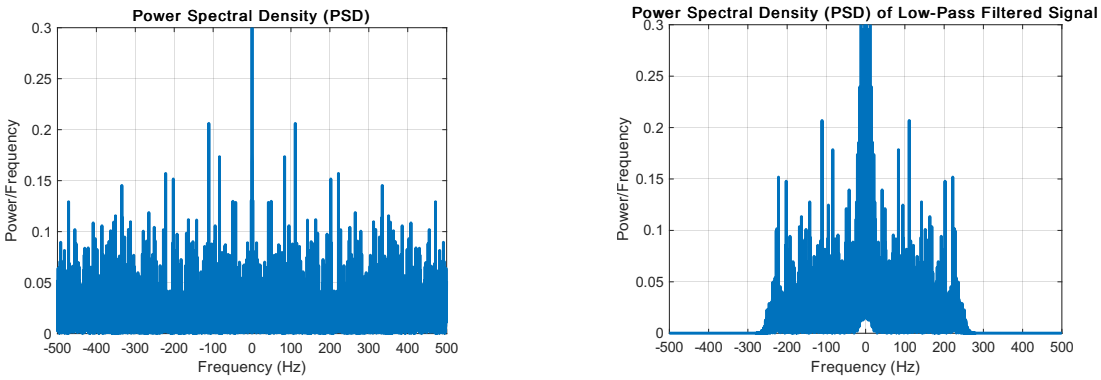
A sample that resembles a full range of motion (FRM) bicep curl with no added mass was selected for the analysis. The FFT analysis was conducted in MATLAB, as shown in Figure 10.



(a) Original signal (b) FFT analysis of the signal

Figure 10: An FRM sample's EIM magnitude time signal and its corresponding frequency spectrum.

The signal was detrended to simplify the interpretation of the frequency spectrum, using the DC component as the reference point and the positive and negative frequencies on either side as symmetric components. It was challenging to determine the dominant frequencies in the signal from the FFT alone. Therefore, a Power Spectrum Density (PSD) plot was conducted to make these frequencies more apparent [23].



(a) PSD diagram of the signal (b) Filtered PSD diagram. Cutoff 250$[Hz]$

Figure 11: The PSD diagram of the signal before and after applying a low pass filter.

As seen in Figure 11, the predominant frequencies are in the lower ranges, with significant periodic spikes at around 100 $[Hz]$ and 200 $[Hz]$.

EIM signals often contain low-frequency components. It is also common to observe spikes at zero frequency in the magnitude spectrum when dealing with physiological signals [8]. Therefore, it is assumed that the lower frequencies are the ones we want to preserve, and the rest are considered noise. A low-pass filter was applied with a cutoff frequency of 250 $[Hz]$ to visualize the isolation of the desired frequencies.

3.3.2 Filter Design

At this point, it was clear that we needed to design a low-pass filter to preserve the low-frequency components of the signal. The next major decision was whether to design a FIR (Finite Impulse Response) or an IIR (Infinite Impulse Response) filter.

FIR Filters: As the name suggests, the filter output response is of finite duration. The input signal is convoluted with the filter coefficients, resulting in a filtered output. FIR filters are inherently stable as their transfer function only contains zeros, and they do not have a feedback loop. They can have a linear phase characteristic, meaning all frequencies are delayed by the same amount. While FIR filters are typically easier to implement than IIR filters, they require a larger number of filter coefficients to achieve a given filter specification, making them computationally more demanding. [24]

IIR Filters: As the name suggests, the filter output response can extend indefinitely in time. The filter output is obtained by recursively applying feedback to the input. IIR filters can be computationally more efficient than FIR filters for achieving the same filter specifications but may exhibit non-linear phase characteristics, meaning different frequencies are delayed by different amounts. IIR filters can have stability issues, and obtaining a linear phase characteristic can be challenging.

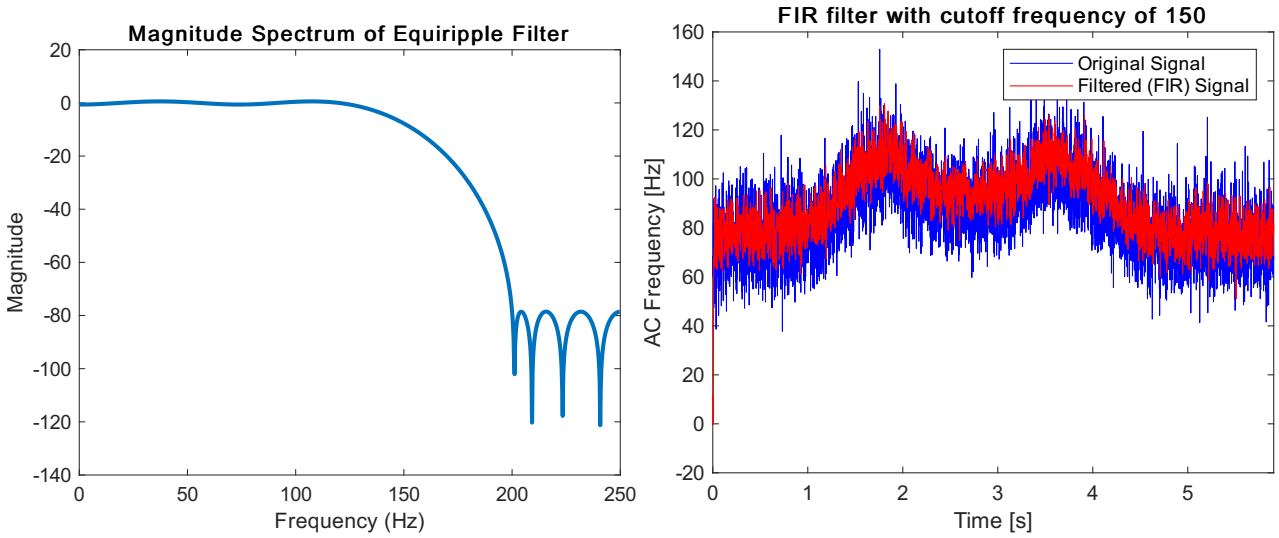
Choosing a FIR filter would increase computation time, but the trade-off of not having to worry about stability and having a guaranteed linear phase characteristic seemed worth the cost. Moreover, the simplicity of implementation was equally attractive. The general equation for the output signal $y(n)$ of a FIR filter with $N + 1$ coefficients ($h(k)$), represented in the discrete-time domain, is as follows:

$$y(n) = \sum_{k=0}^{N} h(k) \cdot x(n-k) \tag{2}$$

Where N is the filter order, $h(k)$ are the filter coefficients, and $x(n)$ is the input signal.

For filter design, MATLAB's $fdatool$ was used to get a visual representation of the magnitude response as we were changing the filter characteristics. For the FIR filter, Equiripple was chosen for its ability to exhibit equal or nearly equal ripples in both the pass band and the stopband, making the response more consistent. Equiripple filters are known for achieving a high level of stopband attenuation for a given filter order. They also allow for a trade-off between the width of the transition band (the frequency range between the pass

band and stopband) and the level of ripple in both the pass band and stopband. Choosing Equiripple also allowed for automating the process of choosing the filter order, which we wanted as low as possible to ease the computational impact on the embedded device. This resulted in a pass-band at $50\ [hZ]$, a stopband at $200\ [hZ]$, and a stopband dampening of $80\ [dB]$.



(a) Magnitude response of the filter.

(b) Original, truncated and filtered signal

Figure 12: Equiripple filter of 16th order's magnitude spectrum and impact on the sample.

The effect of the filter can be seen in Figure 12. We prioritized having as little pass-band ripple as possible, keeping the noise at a minimum, and having a high stopband attenuation.

3.4 Angle Tracker Software

To gather the ground truths of the elbow angles for training the neural network, we implemented angle tracking software.

Initially, we considered a motion capture system using motion trackers. However, due to software issues, we sought an alternative solution that did not require motion trackers. We found our solution in Google's MediaPipe library, using its Pose Landmark Detection model as the foundation for our angle tracking software.

For each prediction, the model returned the coordinates of each predicted joint from the feed, but did not calculate the angle. We implemented a helper function to perform this calculation using simple trigonometry. Given the shoulder, elbow, and wrist coordinates as P_{wrist}, P_{elbow}, and $P_{shoulder}$, we determined the angle θ:

$$\theta = tan^{-1} \frac{P_{shoulder} - P_{elbow}}{P_{wrist} - P_{elbow}}$$

Figure 13: MediaPipe pose estimator, adapted into an angle tracking solution.

We implemented the software using state machine logic for enhanced control. The program starts in the **IDLE** state, awaiting user input. While in **IDLE**, the angle is visualized on stream for target sample adjustment (see Figure 13), before transitioning to the **RECORDING** state. In this state, the angles and Unix timestamps are stored into NumPy arrays, which are then saved into a CSV file once the user gives the **SAVING** command. The program returns to **IDLE** once the sample is saved. The program can be terminated at any time by a keyboard input from the user, resulting in the **EXIT** state.

Concerns were raised about the software's ability to accurately capture the test subject's angle. These concerns were based on the margin of error that deep learning models inevitably introduce. However, it was found that as long as the subject was fully visible in the feed, the tracking would only oscillate approximately 0.1 degrees, and the angle would only update when the subject moved their arm. The greatest inaccuracies occurred when the wrist was obscured by the dumbbell during testing, indicating that hammer curls provided the most accurate tracking. However, this discrepancy in error was not deemed significant enough to exclude the two other types of curls used in the experiment.

3.5 Designing a Neural Network

This section explores the various considerations involved in building a neural network to predict joint angles and mass solely from BI signals.

3.5.1 Choosing a NN Model

The first step in choosing an appropriate model is to identify the type of problem at hand. Machine learning primarily comprises five types of models [25]:

1. **Classification Model:**

   Classification models categorize input data into predefined classes or labels. The goal is to learn a mapping from input features to discrete output categories.

2. **Regression Model:**

   Regression models predict continuous numerical values. They learn relationships between input features and continuous target variables.

3. **Clustering Model:**

   Clustering models group similar data points together based on their intrinsic properties. The goal is to discover inherent patterns and structures within the data.

4. **Dimensionality Reduction Model:**

   Dimensionality reduction models aim to reduce the number of features in a data set while preserving its essential information. This simplifies the data and improves computational efficiency.

5. **Generative Model:**

   Generative models can generate new data samples that resemble the distribution of the training data. They learn the underlying patterns and structures of the data to create realistic samples.

Let's break down the key characteristics of our problem: the target variables (mass and angle) can vary continuously. We aim to predict a numerical measure of these, where the output is not constrained to discrete classes and categories, and we want to predict future joint angles and masses based on prior observations. These characteristics indicate that our problem is a regression problem, which means we need to select a regression model for the task.

There are numerous regression algorithms available to solve this problem. The following is a list of the most common ones and how they would fit our problem [26]:

1. **Linear Regression:**

   Linear Regression (LR) models the relationship between the input features (EIM signals) and the target variable (elbow angle or mass) as a linear equation. It assumes a linear relationship between the features and the target. While it is computationally efficient and easily interpretable, it may not be able to capture complex temporal dependencies in sequential data like EIM signals.

2. **Random Forest Regression:**

   Random Forest Regression is an ensemble learning method that constructs multiple decision trees during training and outputs the average prediction of the individual trees for regression tasks. It can capture complex non-linear relationships in the data and provide insights into feature importance, which might make it a suitable choice for our problem.

3. **Support Vector Machines (SVM) Regression:**

   SVM for regression (SVR) aims to find a hyperplane that best approximates the mapping from input features to the target variable while minimizing errors. It works well in high-dimensional spaces and is effective for non-linear relationships. It allows flexibility with different kernel functions.

4. **Polynomial Regression:**

   Polynomial Regression extends linear regression by considering the polynomial features of the input variables. It allows the model to capture non-linear relationships. However, it can be prone to overfitting, especially with higher-degree polynomials.

5. **K-Nearest Neighbors (KNN) Regression:**

   KNN regression predicts the target variable by averaging the values of its k nearest neighbors in the feature space. It can adapt to complex patterns and is effective if there are localized relationships in certain regions of the feature space. However, it may struggle with high-dimensional data.

6. **Recurrent Neural Networks (RNNs):**

   RNNs are designed for sequential data, like time series or EIM signals, over time. They naturally capture temporal dependencies, making them suitable for problems where the order of data matters. They can handle variable-length sequences, adapting to different patterns and lengths in EIM signals, and can automatically learn relevant features from sequential input data. This is beneficial for tasks where manually engineered features may not capture the full complexity of the relationships. The drawback of using an RNN over other regression algorithms is that the interpretability is somewhat of a black box in comparison with LR or SVM, since the recurrent model optimization is stored in the hidden layers.

Given the sequential nature of our data (EIM signals over time), selecting an RNN for the task is a natural choice. The next step is to narrow down which RNN unit to use for building the model architecture. Of these, the following were considered [27]:

1. **LSTM (Long Short-Term Memory):**

   LSTM mitigates the vanishing gradient problem through a gating mechanism and can capture long-term dependencies in sequences. However, it is more complex than a simple RNN and requires more computational power and proper regularization to avoid overfitting.

2. **GRU (Gated Recurrent Unit):**

   The GRU is simpler than LSTM with fewer parameters, making it computationally more efficient. It is still effective in capturing long-term dependencies and performs well with less available training data. However, it might not capture relationships as precisely as LSTM.

3. **Bidirectional RNN:**

   A Bidirectional RNN processes input sequences in both forward and backward directions, allowing it to capture information from both the past and the future. This is useful for tasks where context from both directions is important (e.g., natural language processing). However, it requires more computational resources as the entire sequence must be analyzed before making a prediction. This makes it unsuitable for real-time applications where immediate predictions are needed. While bidirectional processing can improve accuracy for certain tasks, it's computationally intensive and not ideal for real-time scenarios.

4. **Deep RNN (Stacked RNN):**

   A Deep RNN is essentially multiple RNN units stacked on top of one another to capture more intricate patterns and dependencies in data. The drawback is increased computational complexity and training time, as well as susceptibility to vanishing and exploding gradient problems in deep sequences.

5. **Hierarchical RNN:**

   A Hierarchical RNN is designed to capture hierarchical structure in sequences and is suitable for tasks where information at different levels of abstraction is important. However, its complex architecture requires careful tuning and may not necessarily perform significantly better than simpler models.

To address the vanishing gradient problem evident in classic RNNs while keeping the model complexity at a minimum, the choice was between LSTM and GRU. Because GRU is simpler than LSTM, it was chosen for this task. Note that the final structure is a deep RNN, consisting of two GRU units (see Figure 28).

In GRU, the memory cell in LSTM is replaced by a candidate activation vector, which is updated using two gates: the reset and the update gates, which are determined by:

$$\begin{aligned} z_t &= \sigma(W_z \cdot [h_{t-1}, x_t] + b_z) \\ r_t &= \sigma(W_r \cdot [h_{t-1}, x_t] + b_r) \\ \tilde{h}_t &= \tanh(W_h \cdot [r_t \odot h_{t-1}, x_t] + b_h) \\ h_t &= (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \end{aligned} \tag{3}$$

Where z_t is the update gate, determining how much of the past information should be combined with the new information, t denotes the current time step, h_t is the hidden state at time t, and x_t is the input at time t. W_z and b_z are the weight matrix and the bias term for the update gate, and σ represents the sigmoid activation function.

The reset gate, denoted by r_t, determines how much of the past information should be forgotten, where W_r and b_r are the weight matrix and bias term for the reset gate. The candidate hidden state, denoted by $\tilde{h}_t$, represents the new information that could be added to the hidden state, where W_h and b_h are the weight matrix and bias term for the candidate hidden state, and $\odot$ denotes element-wise multiplication. The new hidden state (h_t) is a combination of the past hidden state and the candidate hidden state, controlled by the update gate.

3.5.2 Feature Extraction

This section justifies the extraction and utilization of various features from the EIM signal for the neural network's input vector. Each feature, derived from the EIM magnitude and phase, is detailed in table 3.5.2.

List of features:			
Median	**Mean**	**Standard Deviation**	**Variance**
Kurtosis	**Sliding Window Average**	**Rate of Change**	**EIM Magnitude, EIM Phase**

The neural network is trained on a total of 16 features, with the elbow joint angle and mass as targets.

3.5.2.1 Sliding Window Averaging

Sliding Window Averaging, also known as a moving average, is a data analysis method that calculates local averages within a moving window. The size of this window determines the number of averages and, consequently, the smoothness of the resulting data set. This method effectively acts as an additional low-pass filter, eliminating the need for frequency domain analysis. Its purpose is to smooth out signal variations and highlight underlying trends or patterns. The method requires a number of input variable iterations equal to the buffer size to take effect. Until then, the data is typically left raw or processed through a regular running average.

In this project, a regular running average was chosen to fill out the data points before the sliding window averaging takes over. The implemented code is explained with equations 4 and 5.

$$k = \frac{N}{c} \tag{4}$$

$$y_i = \begin{cases} \frac{1}{i+1} \sum_{i=0}^{N} (x_i) & \text{if } i < k \\ \frac{1}{k} \sum_{i=N-k+1}^{N} (x_i) & \text{if } i \geq k \end{cases} \tag{5}$$

Where x_i represents the input variable at index i. y_i denotes the output variable, the averaged signal at index i. N denotes the length of the input signal. c is a user-determined constant. k is the computed buffer size, determining the size of the moving window.

This feature was implemented in the early stages of the project to further retrend the signal after applying the FIR filter, thereby facilitating data analysis. See Figure 14 for more details.

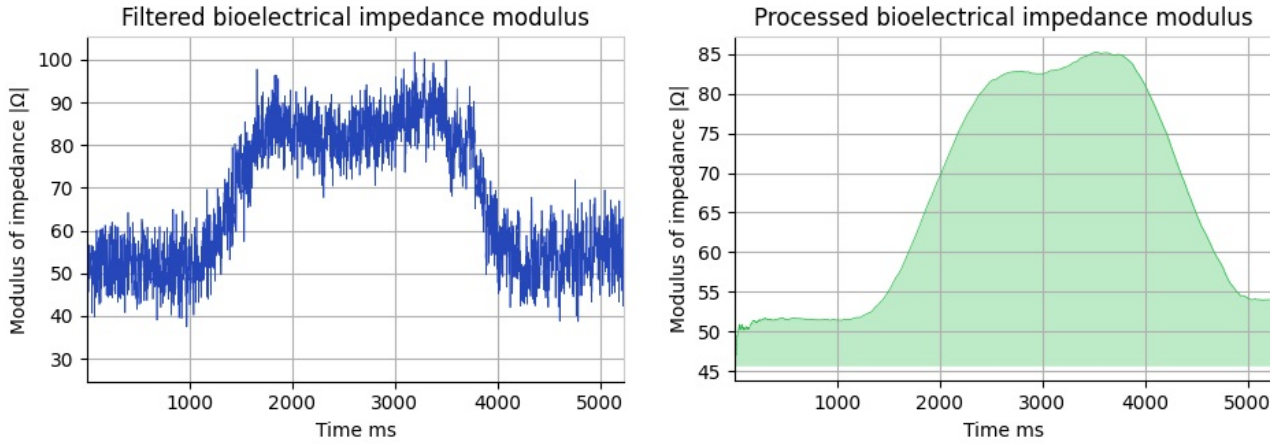


Figure 14: Image of a full range motion curl (FRM), with jitters, with a lifted mass of $6[kg]$. Left: Filtered EIM magnitude. Right: Corresponding sliding window averaging with $c = 5$.

As illustrated in Figure 14, the sliding window averaging assists in discerning whether a given signal change is real or noise. In the image, $c = 5$, which was later adjusted to $c = 75$ to avoid excessive data smoothing and potential information loss. The EIM data set, including both the magnitude and phase, underwent sliding window averaging. Subsequently, it was incorporated as a feature for neural network training.

3.5.2.2 Rate of Change

A fractional rate of change was calculated for both magnitude and phase at each time step, based on the sliding window averaging. This was done to more accurately detect a deliberate rise or fall in signal, which could help the model determine whether a subject is holding the arm still or moving. The formula is given by:

$$\text{ROC} = \frac{x_i - x_{i-1}}{x_{i-1}} \tag{6}$$

A negative Rate of Change (ROC) of the BI modulus indicates that the subject is extending their arm, while a positive ROC indicates that they are flexing their arm.

3.5.2.3 Mean and Median

The mean and median were calculated for both phase and magnitude at each sample point. The mean was calculated under the assumption that it would provide the NN model with an insight into the "balance point" of each sample, thereby helping it distinguish between a full range of motion curl and a half range of motion curl. Given that the data in each sample is roughly normally distributed and not heavily influenced by outliers, the mean is a suitable choice as a feature.

Analyzing the median in conjunction with the mean provides insight into the skewness of the sample distribution. In a symmetric distribution, the median will be fairly close to the mean. In a right-tailed distribution, the mean is typically greater than the median, while in a left-tailed distribution, the mean is typically less than the median. Both are given by:

The mean:

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{N} x_i \tag{7}$$

The median for an odd-sized sample is:

$$\tilde{x} = x[\frac{N}{2}] \tag{8}$$

The median for an even-sized sample is:

$$\tilde{x} = \frac{x[\frac{N}{2}] + x[\frac{N}{2}+1]}{2} \tag{9}$$

Where $\bar{x}$ is the mean, $\tilde{x}$ is the median, N is the number of observations, and x_i represents each individual observation.

3.5.2.4 Standard Deviation and Variance

The standard deviation, denoted as σ, and variance, denoted as σ^2, are measures of variability or dispersion within a data sample. They provide insights into the spread of values from the mean, offering a sense of the distribution's "width".

σ quantifies the average distance of data points from the mean. A low standard deviation indicates that the data points tend to be close to the mean, suggesting low variability. Conversely, a high standard deviation suggests that the data points are spread out over a wider range. In our context, a static sample would have a lower σ than a dynamic sample.

Similar to σ, a low σ^2 indicates that the data points are close to the mean, while a high variance indicates a wider spread of data points. They are calculated as follows:

Sample Standard Deviation (σ)::

$$\sigma = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2} \tag{10}$$

Sample Variance (σ^2):

$$\sigma^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2 \tag{11}$$

Here, σ is the sample standard deviation, σ^2 is the sample variance, n is the number of observations, x_i represents each individual observation, and $\bar{x}$ is the sample mean.

Extracting the standard deviation can aid in outlier detection, as outliers often lie several standard deviations away from the mean. Observing how many data points fall outside certain multiples of the standard deviation can help identify outliers.

3.5.2.5 Kurtosis

Kurtosis is a statistical measure that describes the shape of a data set's distribution. It measures the "tailedness" of the data distribution, indicating whether the data is heavy-tailed (more extreme values) or light-tailed (fewer extreme values).

Positive kurtosis indicates a distribution with heavy tails and a sharp peak, suggesting more outliers or extreme values. Negative kurtosis indicates a distribution with light tails and a flat peak, suggesting fewer outliers and a distribution more concentrated around the mean.

Zero kurtosis indicates a distribution similar to a normal distribution. Kurtosis is often defined as:

$$\text{Kurtosis} = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{x_i - \bar{x}}{s}\right)^4 - 3 \tag{12}$$

Here, s is the sample standard deviation. The constant 3 is subtracted to provide a measure relative to a normal distribution's kurtosis.

The kurtosis as a feature should accomplish two things:

1. Help the model distinguish between a "quick" curl, resulting in a quick rise and fall in magnitude.
2. Help the model detect when the subject pauses at the top before lowering the arm again.

3.5.3 Optimization of the GRU Network

Given that the target hardware is a compact embedded device, it was crucial to minimize the model's memory usage while still effectively addressing the problem. Consequently, the initial training was conducted with a relatively simple architecture, comprising a single hidden layer with 32 neurons, a ReLU activation function, and an output layer with a linear activation function. All 16 features (refer to Figure 3.5.2) were utilized in the training, with a sequence length (SL) of ten. The loss function employed is mean-squared error (MSE), and the optimizer used is ADAM.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 Input-Layer (GRU)           (None, 32)                4800

 dense (Dense)               (None, 2)                 66

=================================================================
Total params: 4866 (19.01 KB)
Trainable params: 4866 (19.01 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
```

Figure 15: The first GRU model architecture.

The purpose of using an adaptive optimizer is to eliminate the need for adjusting the learning rate for each training session and to prevent the optimizer from getting stuck on a plateau or local minimum, which is a risk when using stochastic gradient descent (SGD). ADAM is one of the most effective adaptive optimizers, maintaining an exponentially decaying average of past gradients. Similar to momentum, the trade-off is slightly lower expected generalization compared to training with SGD. SGD requires more intensive training than ADAM to achieve superior results. Another reason for choosing an adaptive optimizer was the project's time constraint, which did not permit extensive training experimentation. [28]

The rationale for selecting MSE as the loss function is its suitability when the distribution of the target variable is Gaussian, which is similar to the distribution of the elbow angle from biceps curl samples.

MSE is a common choice for regression problems. In our case, we aim to predict the continuous values of the joint angle and mass, making MSE advantageous. It calculates the squared difference between the predicted and actual values. Since the error is squared, the result is always positive. This also means that larger errors contribute more to the total error than smaller ones, imposing a severe penalty on the model for making mistakes. [29]

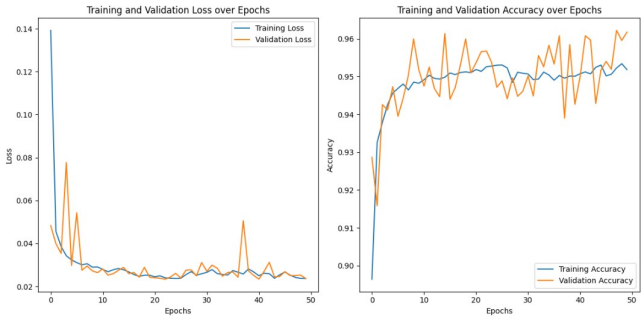


Figure 16: Convergence of Loss and Accuracy for the first GRU model.

The initial training was conducted over 200 epochs, but the trend became evident after just 50. Therefore, to save time, the number of epochs was reduced to 50.

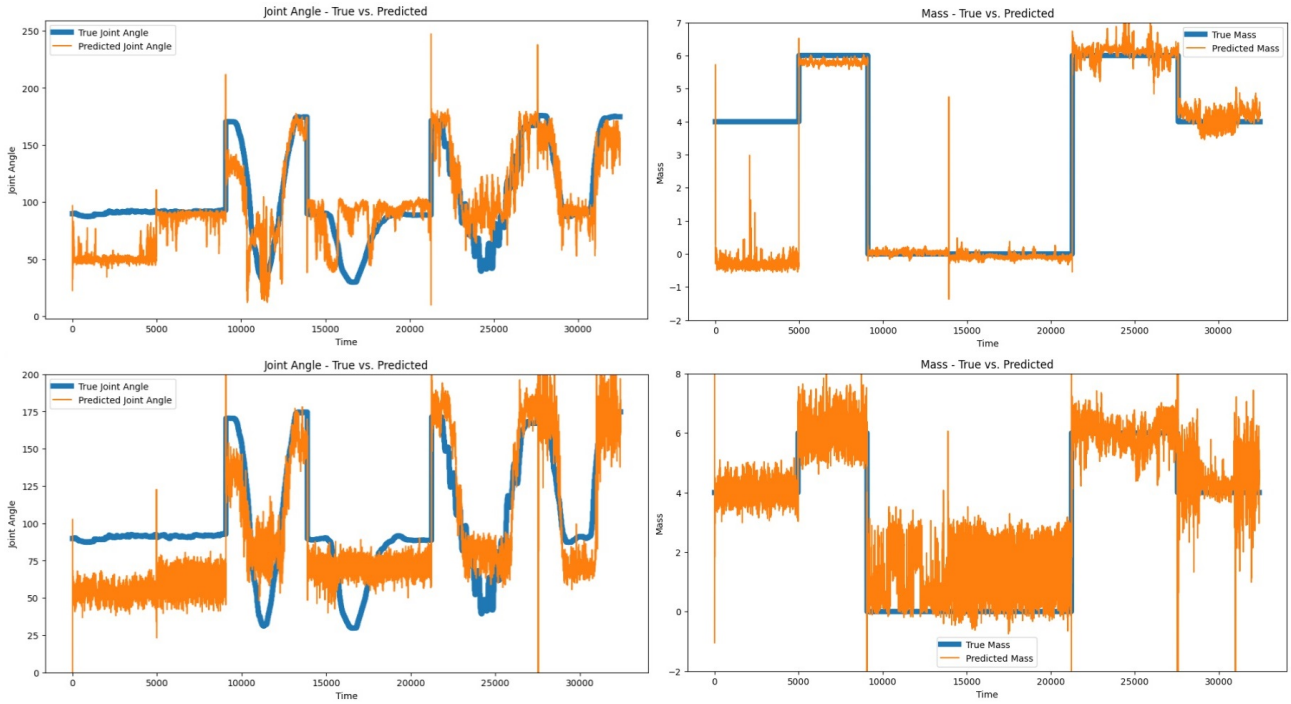


Figure 17: GRU models with sequence lengths (SL) of 10 and 20 predict six test samples.

The model stabilized at a loss of 0.03 and an accuracy of 96% during training, which seemed positive initially. However, when the model was tested with six manually selected samples from the test set, the accuracy dropped significantly to 75.27%, and the loss increased to 0.36, suggesting overfitting. Increasing the sequence length (SL) to 20 did not significantly improve the situation, achieving a maximum accuracy of 95% and a loss of 0.035, but the accuracy dropped to 84.37% at the end of training. On the test set, however, the accuracy increased to 80.64%, and the loss decreased to 0.33. This suggests that increasing the SL might improve generalization.

Training with an SL of 20 appears to generalize better in terms of mass, but it seems to generate significantly more noise than training with an SL of ten. To test this hypothesis, a final training session with an SL of 30 was conducted before modifying the model architecture to mitigate overfitting.

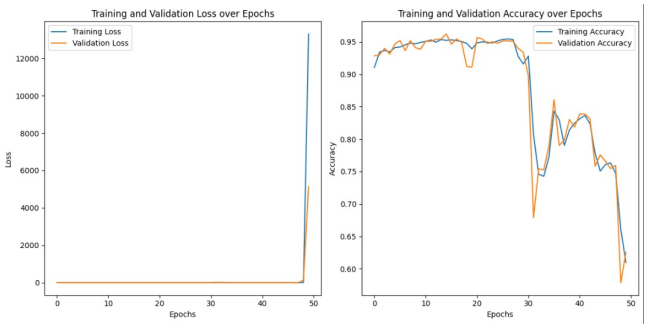


Figure 18: GRU Model with an SL of 30, showing signs of exploding gradient after 30 epochs.

While the Gated Recurrent Unit (GRU) solves the vanishing gradient problem by applying gating techniques that allow the network to selectively update information, the results reveal that it is not immune to the exploding gradient problem (EGP). The rapid increase in losses and unstable training behavior are indicators of this. See Figure 19. This might be due to insufficient weight initialization or an excessively high learning rate. The use of overly large weights might result in significant gradients during training. Using an extremely high learning rate may cause the model to update weights with excessively large gradients, resulting in instability and divergence.

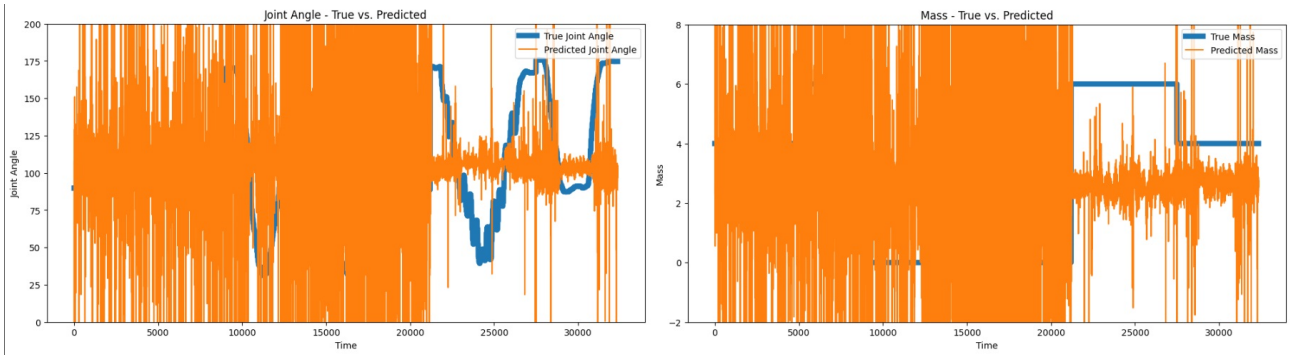


Figure 19: GRU model trained with SL of 30, showing the effect of EGP.

As seen in Figure 19, we have identified the source of the noise. In fact, the model completely loses its ability to generalize as training progresses.

Gradient clipping was used to address this issue by limiting the magnitude of the gradient during backpropagation. The clipping threshold was set to a value of ± 0.5. Additionally, the initial learning rate was reduced from 0.001 to 0.0001. Finally, the weights were initialized using the standard (Gaussian) He initialization method. This is because the He initialization method is designed for ReLU activation, where it initializes weights from a normal distribution. The decision was made to use a normal distribution over a uniform distribution, as the normal distribution is less susceptible to statistical outliers and noise.

Another GRU layer with 32 neurons was added to the model in an attempt to improve accuracy, and dropout was applied on both the recurrent connections and each layer to address overfitting.

A final training session was conducted with an SL of 30 and a dropout rate of 20% on all parameters to evaluate the effect of these modifications.

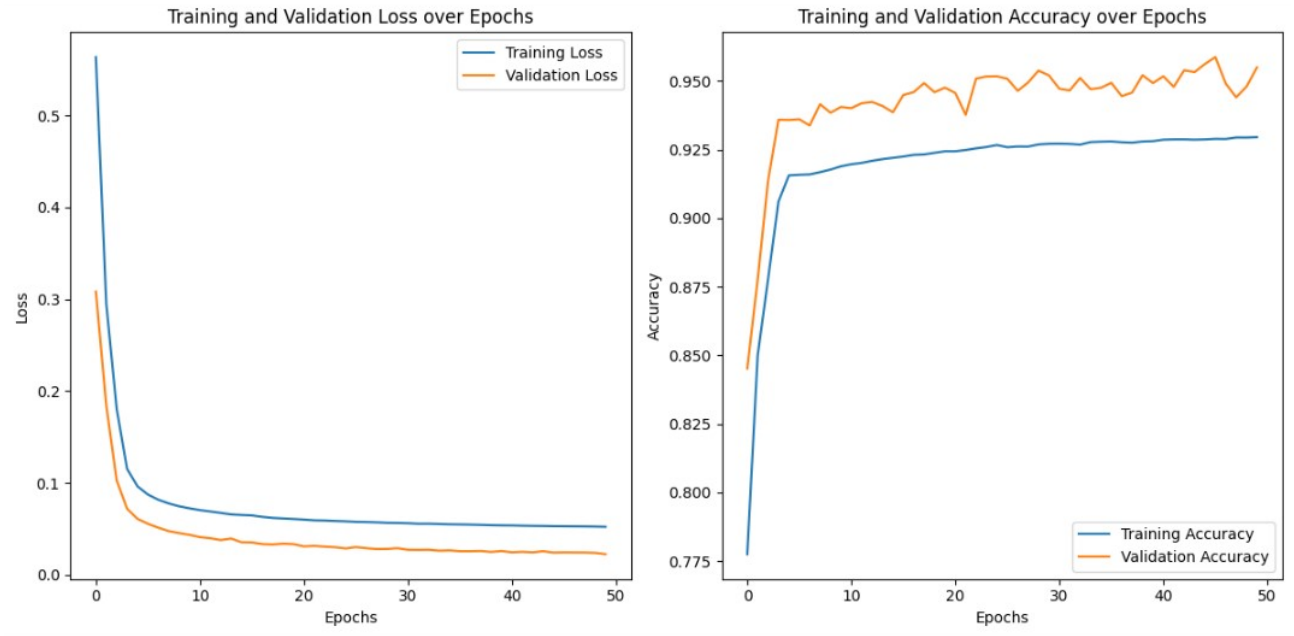


Figure 20: New model with changes to address EGP, with an SL of 30 and Dropout of 20% across all layers.

The smooth convergence in Figure 20 suggests that the EGP is currently stable, with an accuracy of 93% and a loss of 0.052. However, on the test data, the accuracy drops to 86% and the loss increases to 0.167, indicating potential overfitting despite this being the best generalization observed so far.

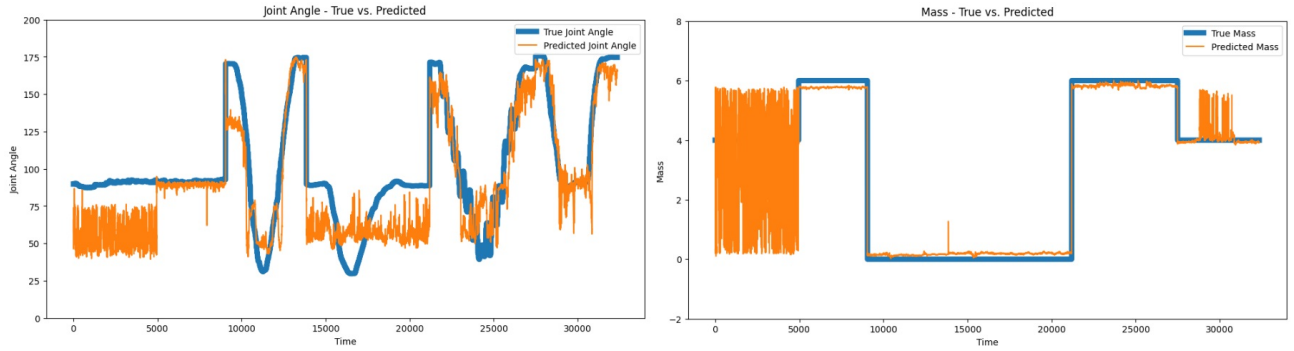


Figure 21: Second model with an SL of 30. EGP has been addressed, but unanticipated behavior is still occurring between time steps 0 and 5,000 and at 30,000.

Figure 21 shows promise, particularly in terms of mass prediction, but the joint angle prediction needs significant improvement. Some noise is also detected, which is expected to reduce as generalization improves.

To mitigate overfitting and enhance model accuracy, batch normalization was introduced on each layer, along with L2 regularization. Additionally, the number of neurons in the second layer was increased to 64, and the dropout rate was raised to 40%.

The aim of implementing batch normalization is to normalize the activations of each layer, allowing for faster convergence during training and stabilizing the optimization process. [30]

L2 regularization is favored over L1 regularization because L1 (also known as Lasso in regression problems)

tends to nullify features with insignificant contributions to the correlation between features and target. As we are still uncertain about the most important features for estimating angle and mass, L1's complete elimination of data is not desirable at this stage. In contrast, L2 (or Ridge) allows for minor contributions from less important features. However, it should be noted that L2 performs poorly on outliers. [31]

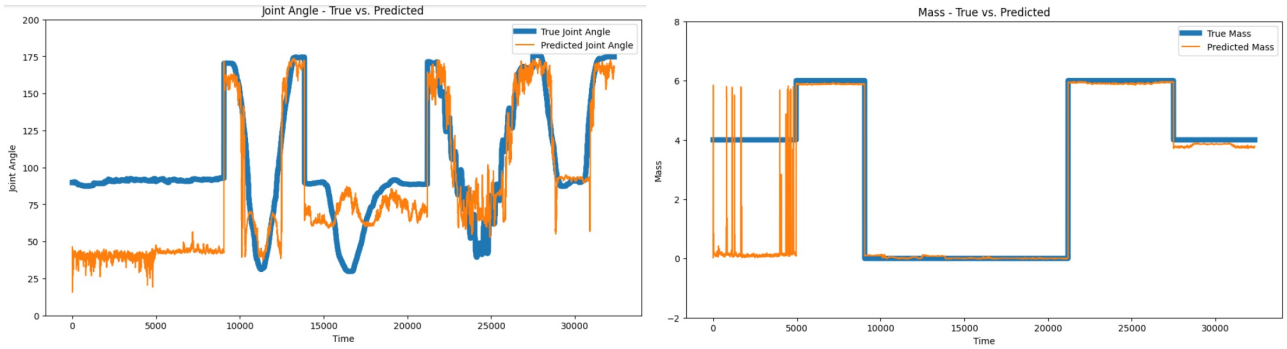


Figure 22: Second model with an SL of 30, L2 regularization (0.01), and 40% dropout on second layer.

The modifications resulted in a decrease in training accuracy to 90% and an increase in loss to 0.096. However, on the test set, the accuracy improved to 87% and the loss increased to 0.297. Figure 22 shows a better match on the mass test data, and the angle prediction seems to align more accurately with the ground truth curvatures. As expected, the gap between training and test accuracy has narrowed, but overfitting remains a concern. Further improvements will likely require hyperparameter tuning.

3.5.4 Hyperparameter Tuning

Hyperparameter tuning is not an exact science, but it revolves around a set of best practices depending on the problem one is trying to solve, and what the results of the trained model are telling you. When done manually, as we have done up until now, hyperparameter tuning can be a time-consuming and difficult task. Fortunately, there are methods that can assist in automating this process. The most popular of these are **Grid search**, **Random search**, and **Bayesian optimization**. The main difference is that while grid search exhausts all possible finite combinations of the given hyperparameter values, random and Bayesian require only a range for each hyperparameter and the distribution it should use to pick the values for each run. While grid search ends when all combinations are exhausted, random and Bayesian can continue indefinitely. [32]

In terms of value selection, Bayesian search optimizes its pick for each iteration to rule out values that are unlikely to result in a strong metric score, whereas Random does not. However, Bayesian search requires a solid grasp of one's area and might be more difficult to implement than random search, while not always offering superior results.

We opted to eliminate grid search for the time being, since we do not have a strong idea of which fixed settings we want to try out, such as learning rate or batch size. Furthermore, we may be uninterested in Bayesian optimization because it optimizes based on the measure provided to it. Although we want high accuracy, we do not want it to come at the expense of loss because there is still some overfitting going on, hence we have settled on a random search.

Weights and Biases (WandB), a third-party library and platform, was utilized to tune the hyperparameters since it provides a wide variety of handy tools for assessing the runs.

When choosing hyperparameters for optimization, it is generally a good practice to only pick a few at a time, in order to control the importance of each hyperparameter and its correlation to the metric one is trying to optimize. Since we believe that we have reduced the level of overfitting to a satisfactory point for now, we want to focus on accuracy while still keeping an eye out for the loss. It makes the most sense to start with a set of hyperparameters that have a significant impact on the stability of the model.

3.5.4.1 Learning Rate

Although an optimizer with an adaptive learning rate was chosen, tuning it is still beneficial. This is due to the modification of the update rule in adaptive optimizers. To illustrate this, let's briefly delve into the mathematics behind it:

$$\begin{aligned} w_{t+1} &= w_t - \alpha \cdot \nabla J(w_t) \\ w_i &= w_i - \alpha \cdot \frac{\partial J}{\partial w_i} \end{aligned} \tag{13}$$

Equation 13 depicts the general update rule for gradient descent, where w represents the model parameters (weights and biases), α is the learning rate, and $J(w)$ is the loss function, which measures the discrepancy between the predicted and actual values. In our case, the mean squared error. $\frac{\partial J}{\partial w_i}$ is the partial derivative of the loss function with respect to the parameter i. This changes slightly when implementing an adaptive learning optimizer, as demonstrated below with ADAM as an optimizer:

$$\begin{aligned} m_t &= \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot \nabla J(w_t) \\ v_t &= \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot (\nabla J(w_t))^2 \\ \hat{m}_t &= \frac{m_t}{1 - \beta_1^t} \\ \hat{v}_t &= \frac{v_t}{1 - \beta_2^t} \\ w_{t+1} &= w_t - \alpha \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \end{aligned} \tag{14}$$

Here, $\nabla J(w_t)$ is the gradient of the loss function with respect to the parameters at time step t. m_t and v_t are moving averages of the gradient and the squared gradient, respectively. $\hat{m}_t$, $\hat{v}_t$ are bias-corrected estimates of m_t and v_t. β_1, β_2 are the exponential decay rates for the moving averages. ϵ is a small constant to prevent division by zero. While the bias-corrected estimates control the weight of the learning rate rather than the partial derivative of the loss function directly, allowing the optimizer to attentively scale the learning rates for different parameters and dimensions in the model, it does not render the careful selection of the learning rate α unnecessary.

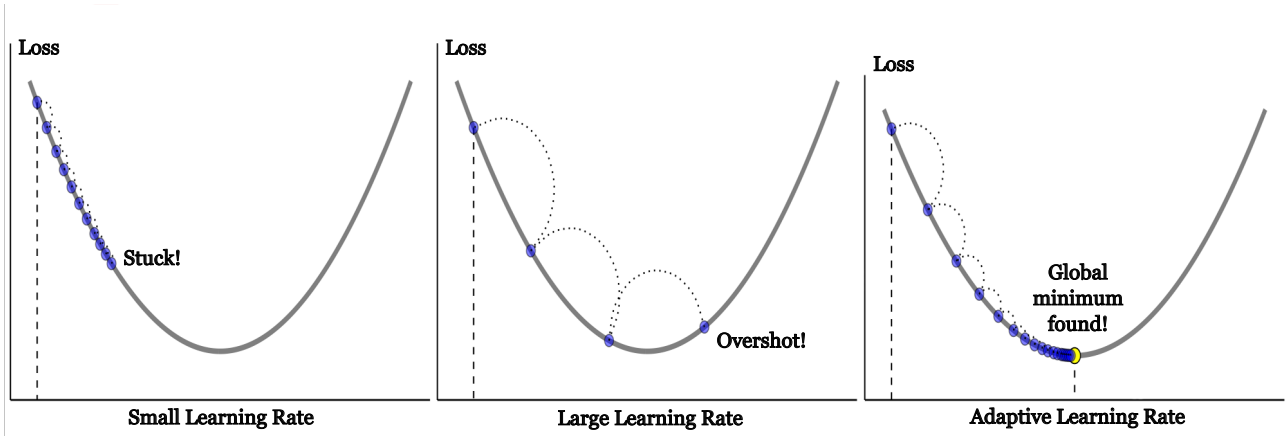


Figure 23: A small learning rate will result in a slow but more stable descent with risk of getting stuck in a local minimum. A large learning rate will converge faster, but has the risk of overshooting the global minimum or even diverging completely. An adaptive learning rate will take previous gradients into account, but can still fall prone to a local minimum.

Figure 23 illustrates the potential pitfalls of not carefully selecting the learning rate, which remains a concern even when using an adaptive optimizer. A too large learning rate risks the model overshooting or even diverging completely from the global minimum. While a small learning rate generally results in more stable training, a too small learning rate risks getting stuck in a local minimum or at a saddle point, if the loss-landscape is non-convex [33]. Therefore, the learning rate is our first chosen hyperparameter.

3.5.4.2 Batch Size

The batch size refers to the number of training examples used in one iteration. During each iteration of the training process, the model parameters are updated based on a subset of the entire training data set, the size of which is the batch size. This process of using batches instead of the entire data set is known as mini batch training [34].

Training in batches is less computationally expensive than using the entire data set, particularly when dealing with large data sets that might not fit into memory. The choice of batch size can influence the model's generalization performance. Larger batch sizes may lead to faster convergence but could result in overfitting, especially if the data set lacks diversity. Smaller batch sizes might offer better generalization, as the model updates more frequently and encounters a wider variety of examples.

When each mini batch is randomly sampled from the data set, it introduces a level of stochasticity during training. Each mini batch provides a noisy estimate of the true gradient, introducing randomness into the gradient estimates and, consequently, into the parameter updates during each iteration. This can aid the optimization process in escaping local minima and exploring the loss landscape more efficiently. The mathematical explanation for this is as follows:.

In mini batch training, the update rule in stochastic gradient descent (SGD), which is also used by adaptive optimizers, is given by:

$$w_{t+1} = w_t - \alpha \cdot \nabla J(w_t; \mathscr{B}_t) \tag{15}$$

Here, $\mathscr{B}_t$ is the gradient computed with respect to the samples in the mini batch. The stochasticity arises from the fact that $\mathscr{B}_t$ is a random subset of the entire data set. Different mini batches will yield different gradient estimates at each iteration, introducing variability in the direction and magnitude of the gradient at each step.

Adaptive optimizers can adapt to this stochasticity and update the learning rate accordingly. The ability of the adaptive optimizers to adjust the learning rate can be influenced by the stochastic nature of the optimization problem. For instance, if the gradients exhibit high variance, the learning rate adaptation might need to strike a balance between adapting to variations and maintaining stability.

Stochasticity in mini-batch optimization can provide a form of implicit regularization, which may help the model generalize better to unseen data by preventing overfitting.

The choice of batch size is interrelated with other hyperparameters, such as the learning rate. Changes in batch size may necessitate adjustments to other hyperparameters to maintain optimal training dynamics. Furthermore, the optimal batch size can depend on the characteristics of the data set. For example, if the data exhibits a high degree of variability, a smaller batch size might be more appropriate. In our case, it is unclear whether the different ways of curling the biceps constitute high or low variability. Considering this, along with the strong interrelation with the learning rate, batch size was chosen as the second one for tuning.

3.5.4.3 Choice of Optimizer

Until now, all training has been conducted with ADAM. However, as previously mentioned, it is possible that SGD as an optimizer could offer better generalization, albeit likely requiring more extensive training. There is also a risk for SGD to get stuck on a plateau or a local minimum. This can be mitigated by using momentum. The update rule for SGD with momentum is given by:

$$\begin{aligned} v_t &= \beta \cdot v_{t-1} + (1-\beta) \cdot \nabla J(w_t) \\ w_{t+1} &= w_t - \alpha \cdot v_t \end{aligned} \tag{16}$$

Here, β is the momentum term, typically close to 1, and v_t is the momentum term, representing the moving average of past gradients. The first equation computes the momentum term v_t based on the weighted sum of the previous momentum v_{t-1} and the current gradient $\nabla J(w_t)$. The momentum term is used to accumulate information about the direction of the gradients over time. The second equation updates the model parameters w_t based on the momentum term.

SGD with momentum forms the foundation for more advanced adaptive optimizers, such as ADAM and NADAM, which incorporate additional adaptive learning rate components. While we have already explained the update rule for ADAM in equation 14, NADAM is simply a variation of ADAM with Nesterov momentum, which is given by:

$$w_{t+1} = w_t - \alpha \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} + \beta_1 \cdot \frac{(1-\beta_1) \cdot \nabla J(w_t)}{1-\beta_1^t} \tag{17}$$

In this equation, m_t, v_t, $\hat{m}_t$, and $\hat{v}_t$ are computed in the exact same way as in equation 14, with the symbols having the same meanings. Nesterov momentum allows the optimization process to "look ahead" and adjust the parameter updates accordingly. Nesterov momentum, or Nesterov accelerated gradient (NAG), is a modification of the traditional momentum method in optimization. The key idea is to use the gradient information not at the current position, but at an adjusted position that takes into account the momentum term. In NADAM, Nesterov's momentum is represented in the last contribution in equation 17.

The performance difference between NADAM and ADAM is not significant, however, the "look ahead" behavior of Nesterov momentum could help to further reduce the chances of getting stuck in a local minimum. It might also aid in the convergence speed during training, as the momentum will assist in the acceleration at areas of small curvature in the loss landscape. One thing worth considering is that NADAM is more computationally intensive than ADAM, resulting in longer training times. This should not affect the rate at which the trained model can make predictions. Other factors like model size, layer complexity, and hardware acceleration (CPUs, GPUs, or TPUs) play a much larger role during the inference. Since we have made the executive decision to keep the max epochs at 50 to be able to test out as many combinations as possible within the time constraint of this project, the extra training time is deemed acceptable if NADAM would perform better.

The final optimizer that we want to test out is RMSprop (Root Mean Square Propagation). It is an optimization algorithm that adapts the learning rates for each parameter individually based on the average of recent squared gradients. The update rule for RMSprop is given by [35]:

$$\begin{aligned} E[g^2]_t &= \rho \cdot E[g^2]_{t-1} + (1-\rho) \cdot (\nabla J(w_t))^2 \\ w_{t+1} &= w_t - \frac{\alpha}{\sqrt{E[g^2]_t + \epsilon}} \cdot \nabla J(w_t) \end{aligned} \tag{18}$$

In this equation, $E[g^2]_t$ is the exponentially weighted moving average of squared gradients at time t, ρ is the decay rate for the moving average (typically close to 1), and ϵ is a small constant to prevent division by zero. The other variables represent the same values as in the other optimizers. The algorithm uses the moving average of squared gradients to scale the learning rates. If the gradient for a particular parameter has been consistently large, the corresponding learning rate will be reduced, and vice versa.

RMSprop is commonly used in non-convex optimization problems, particularly in training deep neural networks. While our network is not particularly deep in terms of layers, the sequence length adds complexity to the model, making it difficult to determine whether the loss landscape is convex. RMSprop is also effective in dealing with sparse data or when the gradients for different parameters have varying scales. It is often chosen for its stability and ability to converge in a variety of settings, at the cost of speed.

RMSprop adapts the learning rates for each parameter individually based on the history of squared gradients. This adaptability can be useful in scenarios where different parameters have different sensitivities to learning rates. Moreover, unlike SGD with momentum, RMSprop does not use a momentum term. The absence of momentum can sometimes be advantageous in specific optimization landscapes. The reason for this is that

momentum relies on accumulating gradients over time, where the presence of noisy gradients could lead to oscillations or undesired movements in parameter space. RMSprop's focus on the magnitude of gradients rather than their accumulated history can provide stability in such scenarios. Moreover, if the objective function ($J(w_t)$, "loss" or "cost" function) is rapidly changing or non-stationary, a momentum term can sometimes hinder adaptation. The accumulated momentum from past gradients may cause the optimizer to overshoot or respond too slowly to sudden changes. RMSprop's ability to adapt the learning rates without relying on a momentum term can offer more responsiveness to rapid changes in the optimization landscape.

RMSprop may be a suitable choice to increase stability in training, and minimize the likelihood of diverging or oscillating.

3.5.4.4 Results from Weights and Biases sweeps

The sweeps were first done with the second architecture, with the choice of optimizer, batch size, and learning rate as the only variables. The runs were done on the full data set, with the same seed split as done before for reproducibility. For the sake of limiting the time for each run, the number of epochs was set to 10 per iteration. Apart from SGD with momentum, all chosen optimizers have some form of adaptive scaling to the learning rate, which usually results in quick convergence, which is why it was deemed that we would still be able to properly access the effect of each optimizer after only 10 epochs.

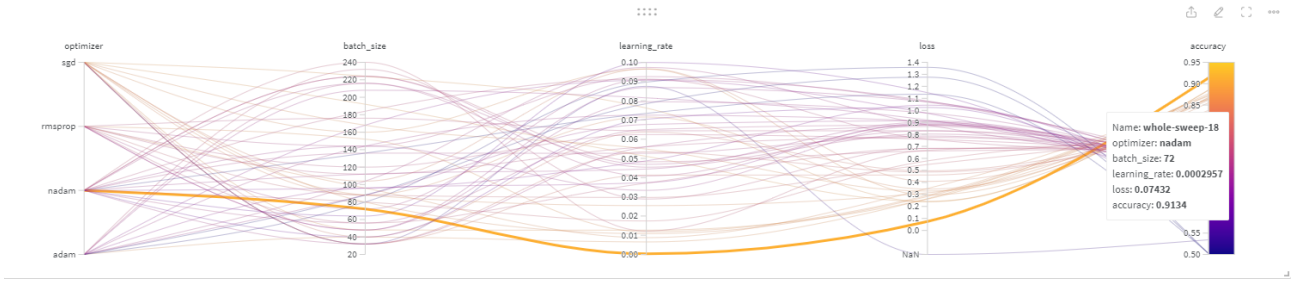


Figure 24: 51 runs with 4 optimizers. NADAM (batch size 72, learning rate 0.0002957) had best results.

Figure 24 shows the results of 51 sweeps done with the settings. As it can be seen, NADAM ended on top with a loss of 0.07 and an accuracy of 91%. The next best was SGD, with a batch size of 224, learning rate of 0.076 and an accuracy of 88% and a loss of 0.23. While RMSprop and ADAM scored about equally well, with their best runs landing at 87% and 0.29 loss. RMSprop had a batch size of 152 and a learning rate of 0.01, while ADAM had a batch size of 40 and a learning rate of 0.0065.

The results might seem to indicate that NADAM performs better than the rest, but if we take a look at the importance of the parameters, it becomes quite clear which impact each hyperparameter has on our metrics.

Figure 25 shows the parameter importance chart from Weights and Biases. It captures linear relationships between hyperparameters and metrics, but not polynomial relationships. It indicates association, not causation, and is sensitive to outliers. A high learning rate negatively impacts accuracy, while a larger batch size has a small positive impact. Optimizer choice is of lesser importance.

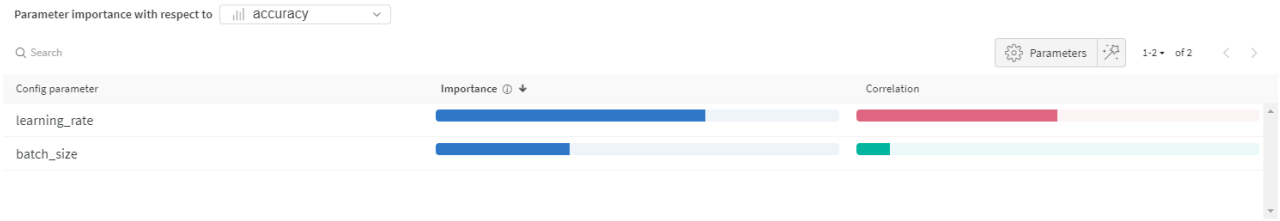


Figure 25: Parameter importance chart.

Nevertheless, NADAM was chosen with the best-performing hyperparameters for the next run, where we wanted to test the effect of changing the layer sizes and dropout rate.

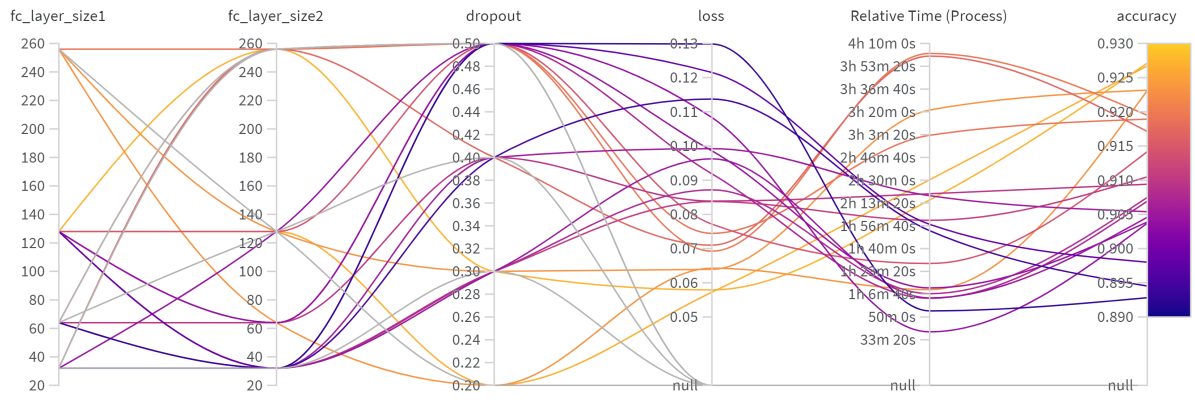
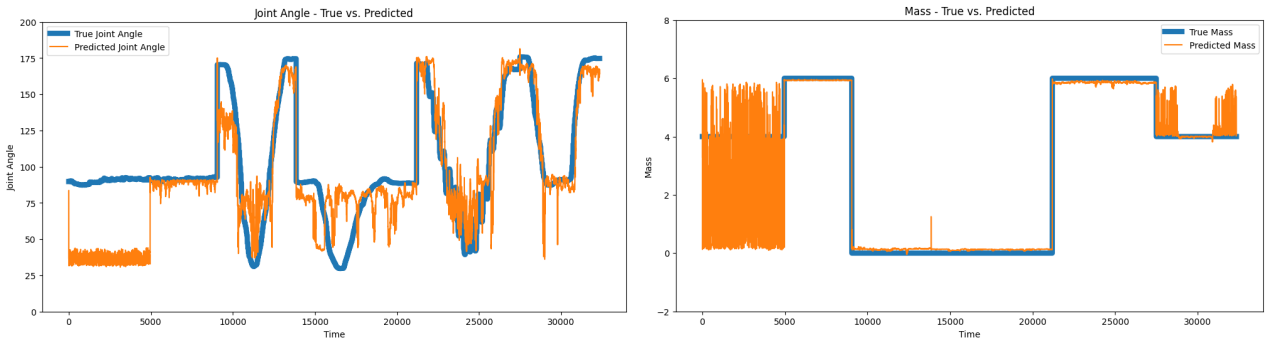


Figure 26: Parallel coordinates from 24 runs of different layer sizes ranging from $32 - 256$ with different dropout rates ranging from $20 - 50\%$.

It quickly became evident that while increasing the layer sizes had a positive impact on accuracy, it made the training time excruciatingly slow. Curiously, it seemed to be more the combination of different sizes that made the training slow than simply increasing everything to the max. Here the slowest run took four hours, with parameters of $(64, 256, 50\%)$ representing "layer1", "layer2", and "dropout rate", while another run with parameters of $(256, 256, 50\%)$ only took three hours. The best run with parameters of $(128, 256, 30\%)$ had an accuracy of 92% and a run-time of $2h15m$, which is not that far from the results of a significantly faster run with parameters of $(256, 64, 20\%)$ scoring 92.3% in only $1h2m$. Granted, the dropout rate is lower, but it still gives an indication that it will not be necessary to dramatically increase the training time in order to achieve better results. Intuitively, we would also want a wider layer at the beginning of the architecture, to give the model a higher capacity for capturing complex patterns and dependencies in the input data. With more units, the GRU layer can potentially learn hierarchical representations of the input sequence, capturing both short-term and long-term dependencies. It is also a general good practice to decrease subsequent layer sizes to reduce the risk of overfitting [36].

The dropout rate was a more difficult hyperparameter to analyze, since we also had promising accuracy and loss convergence on the training set before introducing regularization, where the evidence of the effect first came when testing the model on the test data. Moreover, the dropout rate will naturally have less of a negative impact on accuracy when the layers are much larger. While there were more possible hyperparameters to tune, it was decided to train yet again with the updated hyperparameters, where the layer size and dropout would be set at $(256, 64, 40\%)$ along with the other acquired hyperparameter values. The layer size combination is set to minimize training time, and the dropout is set relatively high to counter the increased risk of overfitting from increasing the layer sizes. The results are seen in Figure 27.



(a) Final model angle predictions. (b) Final model mass predictions

Figure 27: Final trained model with a learning rate of 0.0002957, batch size of 72, optimizer is NADAM, dropout of 40%, first layer size of 256 units, second layer size of 64 units.

With the final model, we were able to squeeze the accuracy up to 90% on the test data with a loss of 0.24. It is noticeable, though, that the model still seems to struggle at the same places, namely predicting the mass when the subject is staying still, specifically with 4 $[kg]$. This could indicate ambiguity in the impedance signal, or simply that the model has not seen enough samples of that type. The final model architecture can be summarized as follows:

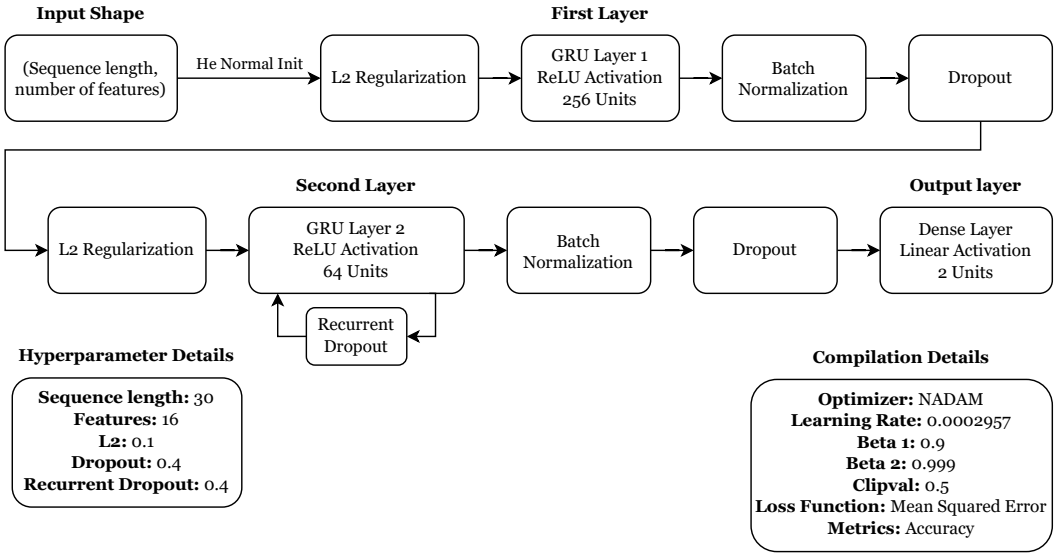


Figure 28: Final model architecture, scoring an accuracy of 90% on test data

4 Experimental Protocol

This experimental protocol serves as a structured plan outlining the step-by-step procedures and guidelines for recreating our experiments, ensuring consistency and reproducibility.

4.1 Physical Setup

For the purpose of this experiment, you will need the hardware and software combination previously mentioned in Chapter 1, and repeated here concisely:

- Eliko Quadra Impedance Spectroscopy device with Single Shunt Front End attached. [2]
- Four EIM electrodes.
- The POW-EXO exoskeleton.
- A PC running Windows 10/11.
- The entire MUSE software stack, found in the MUSE GitHub Repository [37].

The EIM electrodes are sewn to a sleeve made from whichever fabric you have available. This is done to ensure proper contact between the electrode surface area and the bicep, as well as reduce the variability of the sensor placement between subjects.

We designed the sensor sleeve depending on the subject's bicep head length, which was around 6 cm long. Thus, the electrodes were placed $\approx 1.5\,cm$ apart along the bicep, with the lowermost electrode also $\approx$ 1.5 cm from the elbow joint. The elbow joint should be aligned with the lines, as shown in Figure 29. It is important that the two middle sensors are as close to the middle of the bicep as possible. If your subject has a longer or shorter bicep, then the space between electrodes becomes $l_b/4$ where l_b is the bicep length.

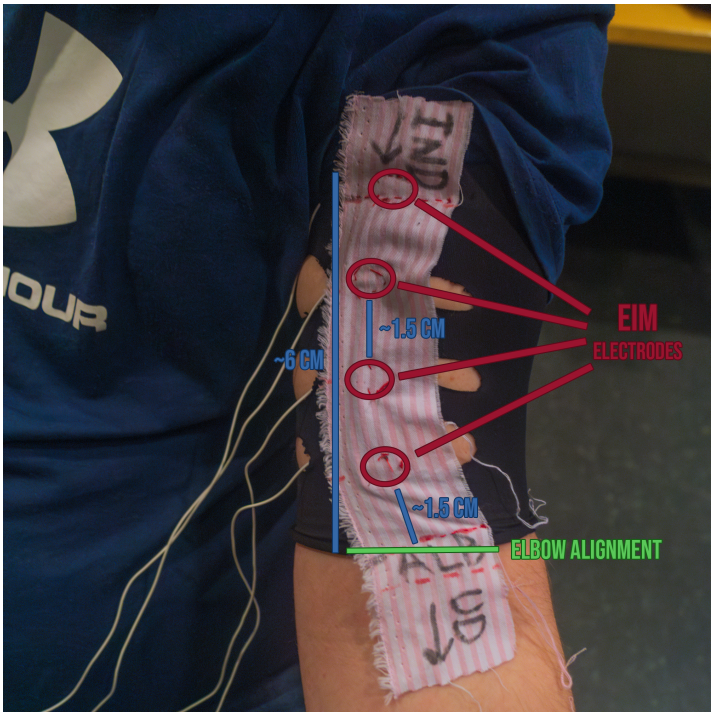


Figure 29: Visual guide for sensor sleeve construction.

4.2 Purpose

The purpose of this experiment is to gather data for the GRU neural network, train the network, process the data output, and finally interface the network and control system.

The purpose of the neural network is to be able to predict the assumed payload weight that the user is lifting and the desired angle of the elbow joint. This is to be able to calculate the feedback and feedforward torque for the control system. The following equations estimate the feedback and the feedforward torque, along with their associated functions:

$$\tau_{ff} = \tau_g \cdot a \tag{19}$$

Where τ_{ff} is the feedforward control signal, τ_g is the torque generated from the gravitational pull, and a is a scalar that the user puts in, ranging from 0 to 1.

$$\tau_g = m \cdot g \cdot l \tag{20}$$

Where m is the mass of the payload, g is the gravitational acceleration ($\approx 9.82\ [m/s^2]$), and l is the length of the arm link.

$$\tau_{fb} = k\left(\theta_r - \theta\right) + B\left(\dot{\theta}_r - \dot{\theta}\right) \tag{21}$$

Where τ_{fb} is the feedback control signal, k and B are tuning variables that the control loop is self-tuning, θ is the current angle, θ_r is the desired angle reference, and finally $\dot{\theta}_r$ and $\dot{\theta}$ are the first-order derivatives of the two former parameters.

$$\dot{\theta}_r = \frac{\Delta\theta}{\Delta t} = \frac{\theta_r - \theta}{t_{i+1} - t_i} \tag{22}$$

Where t_i is the discrete time at the data point i.

4.2.1 Training Data

Training is then done on the impedance input, with the elbow angle and payload weight as ground truths. Impedance and elbow angle are logged for each time step. The payload remains constant throughout the sample, only to be changed between samples. The training data consists of the following:

- Electrical Bioimpedance Ω (From EIM electrodes connected to Eliko device)
- Elbow Angle θ (0 - $180\,\mathrm{deg}$, aka full extension $\longrightarrow$ full flexion, gathered from elbow angle tracker)
- Payload weight m (Weight in hand of, e.g. 0, 4, $6\ kg$)

Where the GRU network uses Ω to predict θ_r and m to pass onto the exoskeleton control system after some post-processing, the exoskeleton can determine whether to assist or resist as needed. Once again, refer to 8 for a visualization of data flow through the program.

4.3 Materials & Software Setup

Materials needed are:

- Eliko interface data collection program with the drivers installed
- Elbow angle tracker program
- Macro to start / stop recording samples

For an installation and usage guide for the entire software system, please refer to the README in the MUSE GitHub repository [37], where it is elucidated in thorough detail.

4.4 Sample Generation Process

This section explains the process of generating samples to train the network with. Below are the variable parameters for the samples; feel free to include more or restrict them, but be aware that a decrease in parameters will likely lead to a decrease in accuracy.

1. **Weights:** Choose weights (e.g., $0\ kg$, $4\ kg$, $6\ kg$ in our case).
2. **Grips:** Consider at least three grips: palm up (i.e., regular bicep curls), palm inward (i.e., bicep hammer curls), and palm down (i.e., forearm curls).
3. **Samples:**
    - **Static:** No movement.
    - **Dynamic:** Predetermined motions.

4.4.1 Procedure

Here is the procedure itself:

- For each participant, weight, grip, and arm:
    - Execute static samples.
    - Execute dynamic predetermined samples.
    - Include any additional useful samples.

Static Samples

- **SampleIndex_X**participant_**Y**kg_180deg_**Z**grip
- **SampleIndex_X**participant_**Y**kg_135deg_**Z**grip
- **SampleIndex_X**participant_**Y**kg_90deg_**Z**grip
- **SampleIndex_X**participant_**Y**kg_45deg_**Z**grip
- **SampleIndex_X**participant_**Y**kg_0deg_**Z**grip

  Examples:

  - 2_ththe_4kg_135deg_hammer
  - 4_sivin_0kg_45deg_palm-down

Dynamic Samples

- **SampleIndex_X**participant_**Y**kg_FullRangeOfMotion_**Z**grip
  - Start out with a relaxed arm, fully bend your arm slowly and squeeze the bicep at the top and wait there for 5 seconds, then slowly relax your arm again.
- **SampleIndex_X**participant_**Y**kg_LowerHalfRangeOfMotion_**Z**grip
  - Start out with a relaxed arm, bend your arm slowly close to 90 degrees and wait there for 5 seconds, then slowly relax your arm again.
- **SampleIndex_X**participant_**Y**kg_UpperHalfRangeOfMotion_**Z**grip
  - Start out at 90 degrees, fully bend your arm slowly and squeeze the bicep at the top and wait there for 5 seconds, then slowly relax your arm again.
- **SampleIndex_X**participant_**Y**kg_FullRangeOfMotionWithStop_**Z**grip
  - Start out with a relaxed arm, bend your arm slowly close to 90 degrees and wait there for 5 seconds, fully bend your arm slowly and squeeze the bicep at the top and wait there for 5 seconds, bend your arm slowly close to 90 degrees and wait there for 5 seconds, then slowly relax your arm again.
- **SampleIndex_X**participant_**Y**kg_FullRangeOfMotionWithJitters_**Z**grip
  - Start out with a relaxed arm, while jittering (as if you had Parkinson disease), fully bend your arm slowly and squeeze the bicep at the top and wait there for 5 seconds, then slowly relax your arm again while still jittering.
- **SampleIndex_X**participant_**Y**kg_FullRangeOfMotionWithPulsing_**Z**grip
  - Start out with a relaxed arm, while pulsing (a bit more aggressively than jittering), fully bend your arm slowly and squeeze the bicep at the top and wait there for 5 seconds, then slowly relax your arm again while still pulsing.

  Examples:

  - 46_ththe_6kg_FullRangeOfMotion_hammer
  - 55_sivin_4kg_FullRangeOfMotionWithJitters_palm-down

This systematic approach creates a diverse data set for training the network, covering various weights, grips, and movements to enhance predictive accuracy from EIM data.

The static samples are there to supplement the GRU network with a significantly higher amount of ground truth data than what the dynamic samples provide by themselves. The dynamic samples themselves do provide some static data, around a second before the main movement and about a second after. This was deemed insufficient for the network to draw strong enough relationships between the inputs and outputs.

The dynamic samples, on the other hand, were chosen for a variety of reasons, which can be classified into two major categories: the ones that represent what movements done by healthy people look like, and the ones that represent what movements done by people with muscular deficiencies look like. The various chosen movements will be described below, and visualizations of them will be present in Appendix A.

4.4.2 Healthy Movement Samples

4.4.2.1 Full Range of Motion

This is one of the major movements that the exoskeleton must be able to do successfully, that is, go from full bicep extension to full bicep flexion. All samples are derivatives of this main movement.

4.4.2.2 Lower Half Range of Motion

This movement was chosen due to its resemblance to the movement of lifting something with your arms perpendicular to the body, for example, a heavy box with no handles.

4.4.2.3 Upper Half Range of Motion

The upper half range of motion movement was chosen for its resemblance to moving your hand to head level. For example, picking up your phone when it is ringing, putting on a beanie, etc.

4.4.2.4 Full Range of Motion with Stop

This simulates a scenario where the user might need to adjust their grip or posture while lifting or lowering an object. The stop also adds a challenge for the GRU network to predict the next state of the muscle activation, as it has to account for the transient change in the EIM signal.

4.4.3 Unhealthy Movement Samples

4.4.3.1 Full Range of Motion with Jitters

This movement was picked to represent patients suffering from extremely invalidating muscle tremors, shown in this case by rough shaking while doing a full range of motion movement.

4.4.3.2 Full Range of Motion with Pulsing

The final movement is the lesser of the two unhealthy ones. It still represents invalidating muscle tremors, but at a lower intensity. This separation in intensity was done to help the GRU network better find the relationship between shaking and actual user intent.

4.5 Sample Collection Process

This section walks through the sample collection process in depth.

1. Wear the sensor sleeve on your upper arm, ensuring the electrodes are all on your bicep and the middle of the sleeve is in the middle of your bicep surface area. Refer to Figure 29 for a visual guide.
2. Connect it to the computer, run the **MUSE** Eliko data interface, and create a throwaway sample (Press **S**, **K**, **Y** and **Enter**) so the startup sequence of the Eliko device does not cause a mismatch in time.
3. Start the **visualizer.py** program so you can visualize your data.
4. Start the **angle_calculator.py** elbow tracker program
5. Start the **MacroRecorder** and **PhraseExpress** Windows apps.
6. For each generated sample:
    - Using the **start_recording.mrf** macro, start sampling in each program simultaneously.
    - Perform the sample.
    - Use the **stop_recording.mrf** macro to stop sampling.

It is recommended to train the network with as many samples as possible; you may even include samples that you believe may aid the network in discovering a relationship between EIM, weight, and elbow angle. Refer to Appendix A for examples of how samples will look in the visualizer.

4.6 Post-processing

In addition to using the macro, Unix time stamps were used to ensure complete synchronization between EIM data and joint angles. During post-processing, the EIM data was truncated to match the timestamps of the joint angles, because of the low sampling rate of the angle tracker software relative to the Eliko device.

The difference in sampling rate between the Eliko device ($1000\ [Hz]$) and the angle tracker software ($18\ [Hz]$), made it necessary to either down-sample the EIM signal, or stretch out the joint angle data points, where the latter was chosen to preserve as much information about the EIM data as possible. The kinematic data points were stretched across the length of the EIM data points, where linear interpolation was applied by averaging between the kinematic data points. To see the implemented interpolation process, see Appendix B.

4.7 Data Interpretation and Analysis

4.7.1 The Effect of Grips

The choice was made to record each sample with **regular**, **hammer**, and **palm down** grips, so the effect of the grip on the data is analyzed. To analyze the effect of the grip on the data, each sample was recorded with three different grips: **regular**, **hammer**, and **palm down**. Figure 30 presents three variations of a FROM sample. The top row contains the modulus of each electrical bioimpedance, while the bottom row displays the phase. The hammer curl exhibits a higher overall range (50 to 100 $[|\Omega|]$), whereas the regular curl and the palm-down grip show lower ranges (40 to 90 $[|\Omega|]$). The phase appears to be unaffected by the grip variation, which is why it is not present in the figure.

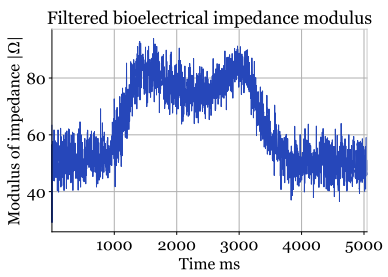
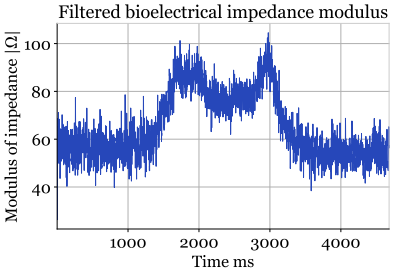
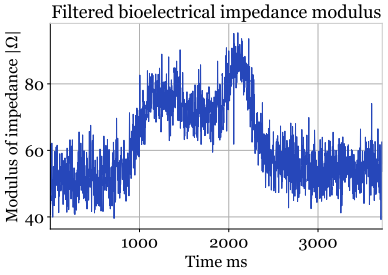


Figure 30: Regular curl grip



Hammer curl grip



Palm down grip

4.7.2 Various Payload Weights

Recording each sample with three different weights, being 0, 4, and 6 $[kg]$, the effect of this will be analyzed below. As depicted in Figure 31, the impedance modulus ranges of the 0 and 4 $[kg]$ samples are quite similar (45 to 80 $[|\Omega|]$), with the 6 $[kg]$ sample exhibiting a slightly higher range (45 to 85 $[|\Omega|]$). The phase range of the 0 $[kg]$ sample (1 to 13 $[\phi\,(\Omega)]$) is slightly lower than that of the 4 and 6 $[kg]$ samples (1 to 15 $[\phi\,(\Omega)]$).

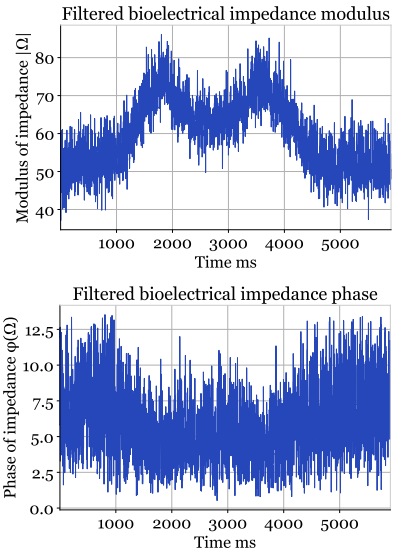
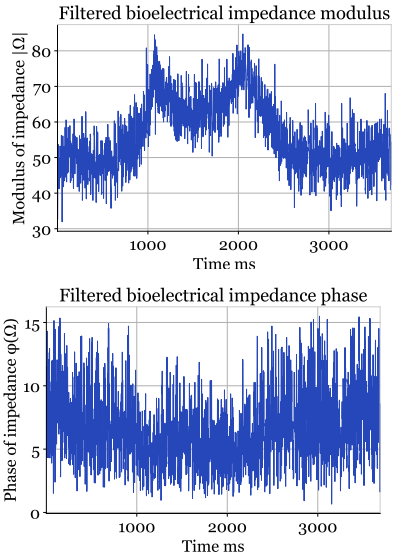
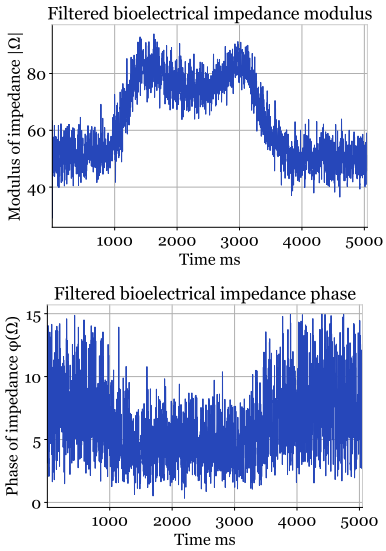


Figure 31: 0 $[kg]$



4 $[kg]$



6 $[kg]$

4.7.3 Potentially Difficult Samples

This subsection explores various potentially difficult samples, the cause of this, and what effect it might have on the accuracy of the predictions from the GRU network that are passed into the exoskeleton interface.

4.7.3.1 Half Range of Motion (Upper)

Three versions of the HRMU movement are seen in Figure 32. As seen in the figure, the movement shows a limited climb from resting position to peak in the impedance modulus, while the middle dip is still present like in the other movements. The phase, however, seems unusable in this type of movement. The limited range in the modulus could make it challenging for the network to establish a proper relationship between the data and the sample, even more so for the almost nonexistent variance of the phase, which once again is not shown as it is almost indistinguishable between samples.

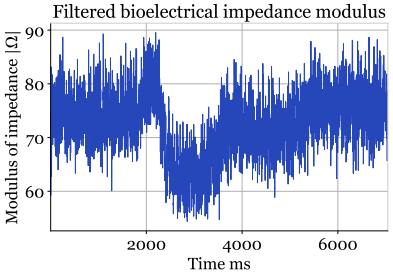
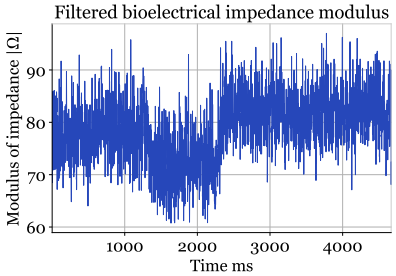
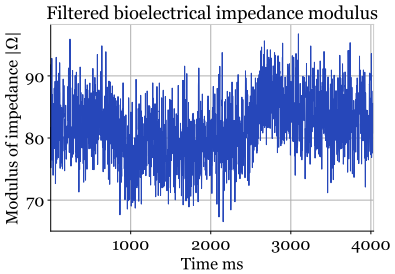


Figure 32: 0 $[kg]$



4 $[kg]$



6 $[kg]$

4.7.3.2 Variance in Static Samples

In Figure 33, three variants of the same static sample are shown: 0, 4, and 6 $[kg]$ versions of the 90° sample. The main concern with these samples, is that the tendencies are barely, if at all, visible in the data. Furthermore, the electrical bioimpedance ranges are almost identical between the 0 and 4 $[kg]$ samples (60 to 80 $[|\Omega|]$), whereas the 6 $[kg]$ one has a slightly larger range (70 to 90 $[|\Omega|]$). Once again, the phases are indistinguishable between samples, at least with the naked eye.

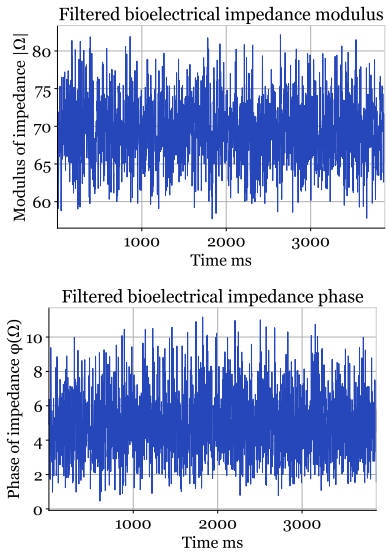
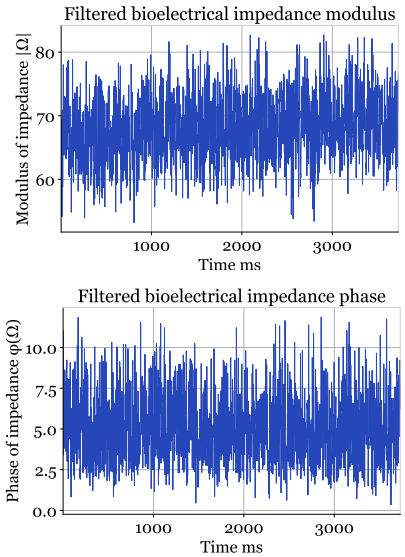
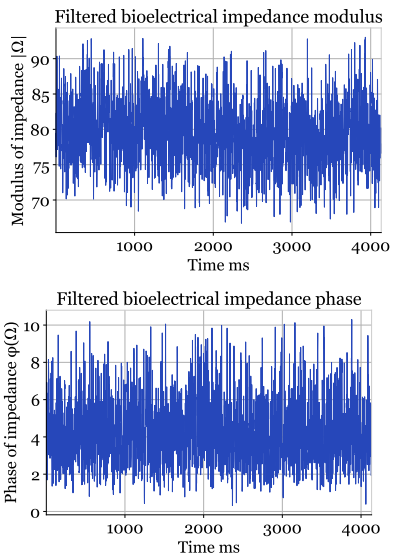


Figure 33: 0 $[kg]$



4 $[kg]$



6 $[kg]$

4.8 Subject Calibration

We retrieved the mean and standard deviation of each feature for normalization. Due to the expected variance in impedance data between individuals, calibrations are conducted for each subject. This involves collecting data, calculating the difference from training data, and adjusting for proper scaling:

Training set:

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i, \quad s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2} \tag{23}$$

New data set:

$$\bar{x}' = \frac{1}{m}\sum_{j=1}^{m} x'_j, \quad s' = \sqrt{\frac{1}{m-1}\sum_{j=1}^{m}(x'_j - \bar{x'})^2} \tag{24}$$

Calculate difference:

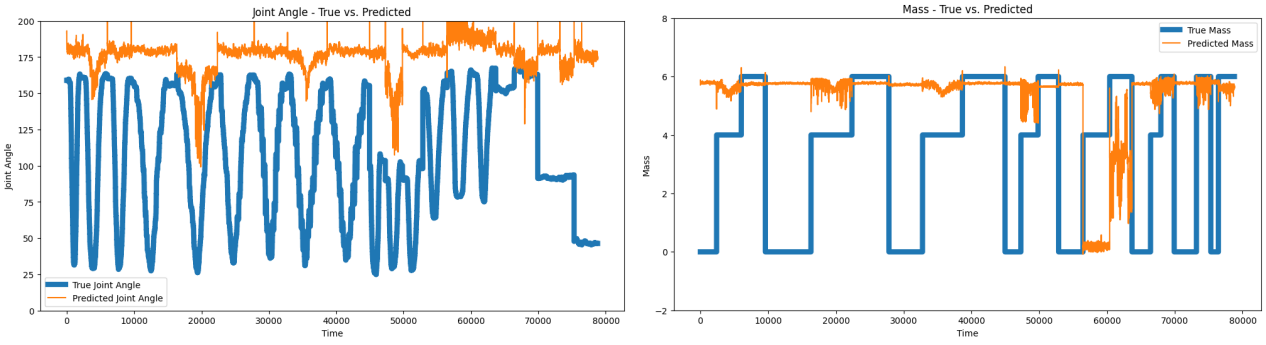
$$\Delta\bar{x} = \bar{x}' - \bar{x}, \quad \Delta s = s' - s \tag{25}$$

Apply the difference to the new data set:

$$\text{Adjusted } \bar{x}' = \bar{x}' - \Delta\bar{x}, \quad \text{Adjusted } s' = s' - \Delta s \tag{26}$$

Where $\bar{x}$ and $\bar{x}'$, are the means, s and s' are standard deviations, n and m are sample sizes, and x_i and x'_j are the input variables to the time i and j.

The model was tested on another able-bodied subject, where the impedance data varied vastly from the training subject. The result of the test was quite poor accuracy at only 54% and a staggering 2.28 loss. The visualization can be seen in Figure 34:



(a) Different subject angle predictions. (b) Different subject mass predictions

Figure 34: Visualization of the model performance on a different subject.

The reason for the poor performance is most likely due to the fact that the model only was trained on a single individual, and the new impedance data proved to be quite different. This is expected to be improved by more samples from different subjects.

5 Discussion

In this section, we will discuss the meaning, importance, and relevance of the results of our project and explore future work options for the project.

5.1 Sensor Considerations

5.1.1 Sensor Placement

The wet sensors, attached to the sleeve and Eliko device, needed to be tightly fastened to the subject in order to give a proper reading. Moreover, the length of a subject's biceps could vary considerably from subject to subject, making the sleeve suboptimal for tracking on other subjects than the sleeve was made for. This is an important observation that can be addressed for the future design of the exoskeleton, so the spacing of the sensors can be modified easily from subject to subject while assuring proper connection to the skin.

5.1.2 Sensor Amount

The project sought to explore what was possible in terms of predicting subject intention with a relatively simple setup, using only four sensors for an EIM signal on the biceps. But particularly for helping in detecting the direction of the motion, or presumably the mass as well, it could be beneficial to add electrodes to the triceps as well. This is because of the subtle stabilizing effect that the triceps have during a curl, which presumably would result in a more prominent signal, the higher the mass being lifted. During the downward phase of the curl, the triceps role is more pronounced, as it helps in controlling the deceleration of the forearm [38].

Other than the biceps, it would be interesting to apply sensors to some other more active stabilizers, namely the shoulder. This way, it would presumably be possible to distinguish between alternating postures of the upper body of the subject during the curl.

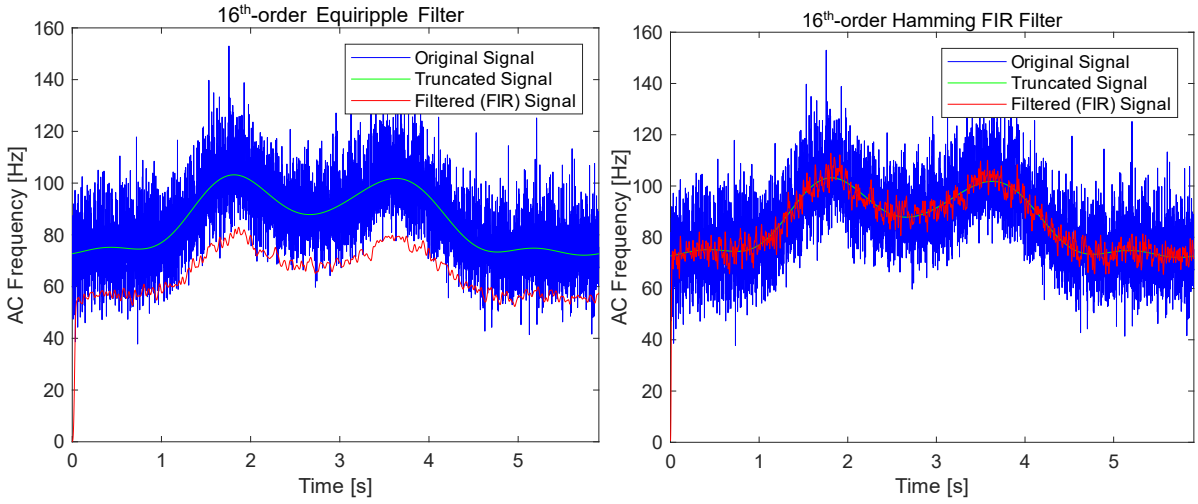
5.2 Database & Internal Communication

The MUSE software system currently employs a "database" of several CSV files to facilitate data exchange among its components, including the Eliko data interface, kinematic angle tracker, data visualizer, and exoskeleton control interface. This strategy, however, is not the most efficient for managing data interfaces across different applications and needs a superior replacement. The system lacks inter-application connectivity, with synchronous data collection between the Eliko data interface and kinematic angle tracker achieved through a keyboard macro — a solution as suboptimal as the database selection.

5.3 Filter Considerations

One could argue that employing a more aggressive filter, specifically one that completely filters out the AC component, might have yielded better results. However, if the goal is to maintain a low filter order, equiripple may not be the optimal choice.

As illustrated in Figure 35, equiripple introduces a dampening effect in the signal, causing the filtered output to fall significantly below the desired truncated reference. Tests with equiripple revealed that an order of 181



(a) Equiripple, order 16, band-pass 1 $[Hz]$, band stop 150 $[Hz]$

(b) Hamming window FIR, order 16, scaled band-pass between 0 and 1, cutoff at 0.002

Figure 35: Signal comparisons with the original, a desired truncated signal at cutoff frequency 1 $[Hz]$ and the corresponding filtered signals.

was required to approach the reference, albeit without ever reaching it. In contrast, achieving satisfactory results with a low order was comparatively straightforward using a Hamming window.

The use of a more aggressive filter would also allow us to eliminate the running window averaging feature, which essentially performs a similar function.

5.4 Neural Network Considerations

5.4.1 Model Performance

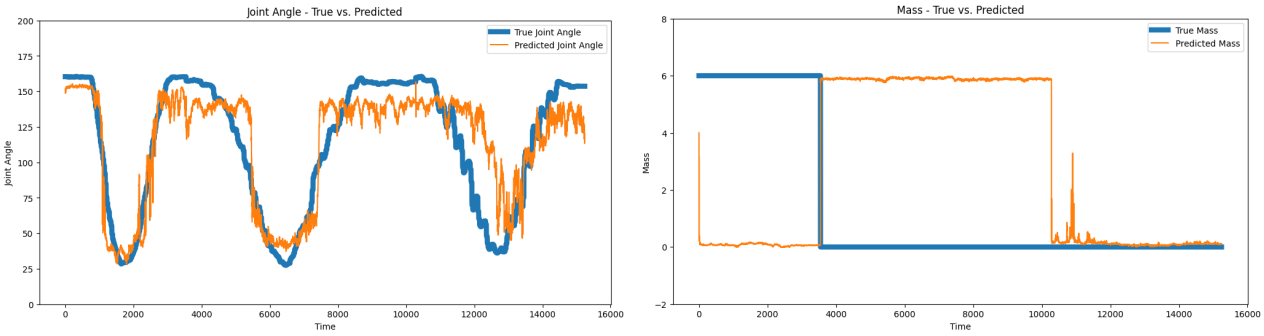
The model's subpar performance on the new subject is unsurprising, given that it was trained solely on data from one subject. The initial plan was to gather data from multiple subjects to enhance generalization. However, due to delays and issues during experimentation, we were unable to collect additional data and retrain the model within the given timeframe.

We attempted a modification to transfer learning to the new subject data set by freezing the model parameters. When this did not yield an improvement in accuracy, we unfroze GRU Layer 2 (see Figure 28). This resulted in a more promising angle prediction on the new subject's test set than the original test data, although the mass prediction was less successful.

However, this rendered the model ineffective on the original test data, suggesting catastrophic forgetting. While adding a new layer, as done in traditional transfer learning, might resolve this issue, concerns about increasing model complexity deterred us from this approach. The optimal strategy would likely involve incorporating the new data into the original data set and retraining the model.

5.4.2 Choosing a NN Model

We chose GRU because we thought that its ability to consider long-term dependencies would come in handy for predicting future targets. That way, we would also distinguish our approach from the paper on "Volitional



(a) Angle prediction on new data after transfer learning. (b) Mass prediction on new data after transfer learning.

Figure 36: Predicted joint angles and mass vs. Ground truth on the test set of a new subject.

control of upper-limb exoskeletons empowered by EM and ML". However, it would have been interesting to see a comparison between our GRU model and SVM, which performed the best on their application, if the optimization process had not taken as long as it did.

It would also have been interesting to see a comparison with LSTM to see if its added complexity would be able to improve the model's performance. The key difference between a Gated Recurrent Unit (GRU) and an LSTM is that a GRU has two gates (reset and update gates) whereas an LSTM has three gates (namely input, output, and forget gates) [39]. This additional gate in LSTM provides more control on the network, making it more flexible [40]. LSTMs are preferred when sequence lengths are more and some good context is there. LSTMs, when trained with more data, give better results than GRU networks [39]. Furthermore, LSTM's more sophisticated memory separates the internal cell state from the cell output, allowing it to output features useful for a task without needing to memorize those features [40].

5.4.3 Hyperparameter Tuning

Time constraints limited the hyperparameter tuning process, but potential improvements could include selecting an activation function other than ReLU or experimenting with different loss functions.

For instance, Mean Absolute Error might be less sensitive to outliers, as it does not square the error. Leaky ReLU could address the "dying ReLU" problem, where neurons become inactive and cease to learn during training. The slight negative slope introduced by it could facilitate learning even with negative input.

L1 regularization, which is more stringent with non-contributing features, could also be beneficial. The number of features included is another consideration. However, if L1 regularization is employed, there would be less concern about identifying contributing features.

Due to the "black box" nature of neural networks, it was unclear which features were contributing to learning. A Weights and Biases grid search sweep could provide insight into this, but testing all combinations would be time-consuming.

5.5 Exoskeleton Interfacing

Integrating with the exoskeleton interface, a complex piece of software not originally authored by us, was a challenging task that occurred late in the process. Despite these challenges, we successfully implemented a model that continuously resets the angle reference based on its predictions.

However, we had to adjust the zero-point reference due to the exoskeleton's design limitations, which would cause a collision at the actual zero point. We limited the range of motion to 180° to prevent this. The actuator's aggressive jerk caused constant overshooting of the target reference, an issue we did not have time to rectify. Consequently, we were only able to implement the angle predictions.

5.6 Future Work

This section explores both solutions to problems unveiled in the first part of the Discussion section 5, as well as the next natural steps for MUSE if we had more time to work on it.

5.6.1 Exoskeleton Actuator Performance

Something that was realized too late in the project was the limiting factor of the actuator installed in the exoskeleton. From the data sheet [41], the maximum torque of the motor is given as $4.1\ [Nm]$ and using a measuring tape, the length of the exoskeleton arm is measured to $0.21\ [m]$.

Making the assumption that the only force affecting the payload is gravity, we can use the formulae for force and torque to calculate the estimated maximum payload that can be lifted by the exoskeleton.

$$F = m \cdot a \tag{27}$$

Where F is the force, m is the mass being lifted, and a is the applied acceleration.

$$\tau = F \cdot l \tag{28}$$

Where τ is the torque and l is the length of the arm lifting.

In our case, the formula becomes:

$$\tau_{max} = F_{max} \cdot l_a = m_{max} \cdot g \cdot l_a \tag{29}$$

$$4.1\ [Nm] = m_{max} \cdot 9.82\ [N] \cdot 0.21\ [m] \tag{30}$$

Reducing and isolating m_{max} on the left-hand side

$$\frac{4.1\ [Nm]}{2.06\ [Nm]} = \frac{m_{max} \cdot 2.06\ [Nm]}{2.06\ [Nm]} \longrightarrow m_{max} = 1.99\ [kg] \tag{31}$$

From equations 29 through 31, we can see that the maximum payload that the exoskeleton can lift is roughly a third as much as the maximum weight that the network was trained with, which was $6\ [kg]$. The right way forward would be to install a motor with at least triple the torque rating of the one currently installed.

5.6.2 Database

The most apparent modification to improve the database problem would be to utilize a relational database management system (RDBMS), such as one of the several SQL-based RDBMSes. These operate well with structured data and are used to organize data and establish links between key data points.

5.6.3 Internal Communication

There are several modifications to make that would add proper internal communication, of which the two discussed in this section are TCP sockets [42] and Robot Operating System (ROS) [43].

5.6.3.1 Transmission Control Protocol

The easiest technique to accomplish intercommunication amongst data gathering applications would be to use a socket, such as Transmission Control Protocol (TCP). It may be as easy as running a TCP socket in a different thread alongside the state machine in the main MUSE program. This would allow MUSE to activate the kinematic angle tracker, collect data via the socket, and then post-process and record the data with the BI data collected via the Eliko interface from the EIM electrodes.

TCP sockets provide a dependable and connection-oriented communication method that ensures data integrity and order. They provide a reliable option for applications requiring precise and sequential data transmission, such as file transfers and real-time systems. TCP manages error recovery and acknowledgment, which helps to provide a reliable communication route [42].

Some disadvantages to consider are that TCP's stability comes at the expense of additional overhead, potentially making it slower than alternative protocols. TCP's connection-oriented structure necessitates a more complicated setup, which introduces delays during connection formation. Furthermore, in cases where network conditions are sporadic or unstable, TCP may not be the most efficient solution since it prioritizes dependability above speed [42].

5.6.3.2 Robot Operating System

Another method is to write the various applications in the system as ROS nodes that exchange data over topics and are activated by custom service calls from the MUSE state machine, acting as a node manager.

There are various reasons to prefer a refactor into ROS nodes over simply using a TCP socket. First, ROS provides robust abstractions that allow developers to concentrate on their goals rather than the complexities of low-level TCP/IP protocol sockets. This abstraction is very useful for managing several processes in a network. ROS's fundamental attractiveness is its large collection of libraries and pre-implemented algorithms, which provide a rich environment for testing ideas, experimenting with concepts, and learning the coordination of robotic actions via message-based communication [43].

Moreover, ROS's flexibility for programming nodes in C++ or Python interchangeably inside the same system is a remarkable feature, encouraging language abstraction. This flexibility allows developers to select languages based on their individual needs; for example, real-time processing tasks can be performed in C++,

while neural network implementation might be done in Python. Furthermore, ROS's open-source nature and large community contribute to a plethora of pre-written nodes for a variety of applications, complemented by extensive documentation, supporting its position as a strong robotic framework [44].

However, while ROS has many advantages, it also has certain drawbacks. The open-source nature of ROS may present challenges outside the Debian and Ubuntu Linux distributions. However, numerous choices, like Docker containers, provide for greater freedom when experimenting with ROS in diverse contexts [43].

5.6.4 Further Data Gathering and Training

Whether or not GRU is the best regression solution to the problem, gathering data from more subjects are needed in order to assure proper generalization of the model. This includes able-bodied individuals with healthy muscle tissue, but especially also individuals with any form of muscular deficiency, as EIM is particularly efficient at capturing ill muscle tissue, which must mean that there is a significant-expected difference from our gathered data of two able-bodied subjects, and these samples [20].

An effort was made during this project to get people with Parkinson's disease involved as test subjects, but the time required for the medical committee to decide on allowing us to perform experiments on these individuals, proved to be outside the range of this project.

6 Conclusion

In conclusion, we have made significant strides in enhancing the capabilities of the POW-EXO exoskeleton. By integrating sensory augmentation, we have addressed key limitations in the existing system, namely the inability to switch between AAN and RAN modes in real time and the difficulty in discerning user intent.

Our exploration of various muscle sensing methods, including sEMG and EIM, and subsequent choice of EIM for the human-machine interfacing, has provided valuable insights into their respective advantages and disadvantages. The development of a sensor interface that collects, filters, and processes intramuscular data has enabled us to make this data interpretable to a GRU network. This has the potential to predict user intent and facilitate seamless transitions between AAN and RAN modes.

However, the project was not without its challenges. The complexity of the exoskeleton's software architecture and the need for precise sensor data processing techniques presented considerable obstacles. Despite these hurdles, we have successfully implemented a solution that enhances the exoskeleton's functionality and offers promising implications for end users with neurological, motor, and muscular deficiencies.

Looking forward, there are several areas for potential improvement and further research. These include the exploration of other sensor technologies, the optimization of data post-processing techniques, and the investigation of additional use cases for sEMG and EIM. The integration of these advancements into the POW-EXO exoskeleton will undoubtedly contribute to the ongoing evolution of this technology, with the ultimate goal of improving productivity and quality of life for its users.

7 References

[1] Xiaofeng Xiong, Cao Danh Do, and Poramate Manoonpong. "Learning-based Multifunctional Elbow Exoskeleton Control". English. In: *IEEE Transactions on Industrial Electronics* 69.9 (Aug. 2022), pp. 9216–9224. ISSN: 0278-0046. DOI: 10.1109/TIE.2021.3116572.

[2] Eliko Tehnoloogia Arenduskeskus. *Quadra Impedance Spectroscopy*. URL: https://eliko.tech/quadra-impedance-spectroscopy/.

[3] Susanna Yu. Gordleeva et al. "Real-Time EEG–EMG Human–Machine Interface-Based Control System for a Lower-Limb Exoskeleton". In: *IEEE Access* 8 (2020), pp. 84070–84081. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.2991812.

[4] Xiangmin Lun et al. "A Simplified CNN Classification Method for MI-EEG via the Electrode Pairs Signals". In: *Frontiers in Human Neuroscience* 14 (2020). ISSN: 1662-5161. DOI: 10.3389/fnhum.2020.00338.

[5] Martin Stevens. "Multimodal Signals and Communication". In: (Feb. 2013). DOI: 10.1093/acprof:oso/9780199601776.003.0006.

[6] Marta Gandolla et al. "An assistive upper-limb exoskeleton controlled by multi-modal interfaces for severely impaired patients: development and experimental assessment". In: *Robotics and Autonomous Systems* 143 (2021), p. 103822. DOI: 10.1016/j.robot.2021.103822.

[7] Biao Chen et al. "Volitional control of upper-limb exoskeleton empowered by EMG sensors and machine learning computing". In: *Array* 17 (2023), p. 100277. DOI: 10.1016/j.array.2023.100277.

[8] Chuong Ngo et al. "A Wearable, Multi-Frequency Device to Measure Muscle Activity Combining Simultaneous Electromyography and Electrical Impedance Myography". In: *Sensors (Basel, Switzerland)* 22 (Mar. 2022). DOI: 10.3390/s22051941.

[9] Bryn Farnsworth Imotions. *What Is EMG (Electromyography) and How Does It Work?* July 2018. URL: https://imotions.com/blog/learning/research-fundamentals/electromyography-101/ (visited on 06/09/2023).

[10] Irwin B. Levitan and Leonard K. Kaczmarek. *The Neuron: Cell and Molecular Biology*. Oxford University Press, 2015. ISBN: 978-0199773893.

[11] Marvin Zuckerman. *Psychobiology of Personality*. Cambridge University Press, 1991. ISBN: 9780521359429.

[12] Carolyn Perry. *Neuromuscular junction: Structure and function*. 2023. URL: https://www.kenhub.com/en/library/anatomy/the-neuromuscular-junction-structure-and-function.

[13] Jane B. Reece et al. *Neurons, synapses, and signaling*. Pearson Education, 2011.

[14] Sunil Nath and John Villadsen. "Oxidative phosphorylation revisited". In: *Biotechnology and Bioengineering* 112.3 (2015), pp. 429–437. ISSN: 1097-0290. DOI: 10.1002/bit.25492.

[15] Abdul Manan Khan et al. "Adaptive impedance control for upper limb assist exoskeleton". In: *2015 IEEE International Conference on Robotics and Automation (ICRA)* (May 2015), pp. 4359–4366. ISSN: 1050-4729. DOI: 10.1109/ICRA.2015.7139801.

[16] Muhammad Gull, Shaoping Bai, and Thomas Bak. "A Review on Design of Upper Limb Exoskeletons". In: *Robotics* 9 (Mar. 2020), p. 16. DOI: 10.3390/robotics9010016.

[17] Minal Bhadane et al. "Re-evaluation of EMG-torque relation in chronic stroke using linear electrode array EMG recordings". In: *Scientific Reports* 6 (June 2016), p. 28957. DOI: 10.1038/srep28957.

[18] O Tate and Diane Damiano. "Torque-EMG relationships in normal and spastic muscles". In: *Electromyography and clinical neurophysiology* 42 (Oct. 2002), pp. 347–57.

[19] Shawn Seeedstudio. *What is EMG sensor, Myoware and How to use with Arduino?* 2019. URL: https://www.seeedstudio.com/blog/2019/12/29/what-is-emg-sensor-myoware-and-how-to-use-with-arduino/ (visited on 06/09/2023).

[20] Seward B Rutkove and Benjamin Sanchez. "Electrical impedance methods in neuromuscular assessment: An overview". en. In: *Cold Spring Harb. Perspect. Med.* 9.10 (Oct. 2019), a034405.

[21] Benjamin Sanchez and Seward B Rutkove. "Electrical impedance myography and its applications in neuromuscular disorders". en. In: *Neurotherapeutics* 14.1 (Jan. 2017), pp. 107–118.

[22] Dec. 2023. URL: https://en.wikipedia.org/wiki/Fast_Fourier_transform.

[23] Amberle McKee. *A Data Scientist's Guide to Signal Processing*. Aug. 2023. URL: https://www.datacamp.com/tutorial/a-data-scientists-guide-to-signal-processing.

[24] Vittal Rao Janardhana. *Difference between FIR filter and IIR filter (with comparison chart)*. Feb. 2021. URL: https://circuitglobe.com/difference-between-fir-filter-and-iir-filter.html.

[25] Manika Nagpal. *Your 101 guide to model selection in machine learning*. Oct. 2023. URL: https://www.projectpro.io/article/model-selection-in-machine-learning/824.

[26] Online Manipal Editorial Team. *10 popular regression algorithms in machine learning*. Oct. 2023. URL: https://www.onlinemanipal.com/blogs/popular-regression-algorithms-in-machine-learning.

[27] Yagna Dakshina. *Types of RNN in deep learning*. July 2023. URL: https://deeplearningofpython.blogspot.com/2023/05/TypesofRNn-deeplearning-keras-architectures.html.

[28] Philipp Wirth. *Which Optimizer should I use for my ML Project?* Dec. 2020. URL: https://www.lightly.ai/post/which-optimizer-should-i-use-for-my-machine-learning-project.

[29] Jason Brownlee. *How to Choose Loss Functions When Training Deep Learning Neural Networks*. Aug. 2020. URL: https://machinelearningmastery.com/how-to-choose-loss-functions-when-training-deep-learning-neural-networks/.

[30] Ketan Doshi. *Batch Norm Explained Visually — How it works, and why neural networks need it.* May 2021. URL: https://towardsdatascience.com/batch-norm-explained-visually-how-it-works-and-why-neural-networks-need-it-b18919692739.

[31] Kurtis Pykes. *Fighting Overfitting With L1 or L2 Regularization: Which One Is Better?* Aug. 2023. URL: https://neptune.ai/blog/fighting-overfitting-with-l1-or-l2-regularization.

[32] Juan Navas. *What is hyperparameter tuning?* Feb. 2022. URL: https://www.anyscale.com/blog/what-is-hyperparameter-tuning.

[33] Jason Brownlee. *Understand the impact of learning rate on neural network performance.* Sept. 2020. URL: https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/.

[34] Jason Brownlee. *Difference between a batch and an epoch in a neural network.* Aug. 2022. URL: https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/.

[35] Vitaly Bushaev. *Understanding rmsprop - faster neural network learning.* Sept. 2018. URL: https://towardsdatascience.com/understanding-rmsprop-faster-neural-network-learning-62e116fcf29a.

[36] Jeff Heaton. *The number of hidden layers.* Oct. 2023. URL: https://www.heatonresearch.com/2017/06/01/hidden-layers.html.

[37] Thomas Therkelsen and Simon Vinkel. *Therkelsen/MUSE: Muscular Signal Processing for Exoskeleton Control (MUSE).* Sept. 2023. URL: https://github.com/Therkelsen/MUSE.

[38] Alyssia Simpson. *Do curls work triceps? say goodbye to flabby arms!* Aug. 2023. URL: https://stridestrong.com/do-curls-work-triceps/.

[39] *When to use GRU over LSTM? - Data Science Stack Exchange.* 2016. URL: https://datascience.stackexchange.com/questions/14581/when-to-use-gru-over-lstm.

[40] *How can a GRU perform as well as an LSTM? - Data Science Stack Exchange.* 2018. URL: https://datascience.stackexchange.com/questions/29105/how-can-a-gru-perform-as-well-as-an-lstm.

[41] Robotis. *XM430-W350-T/R.* URL: https://emanual.robotis.com/docs/en/dxl/x/xm430-w350/.

[42] E. G. Britton, J. Tavs, and R. Bournas. "TCP/IP: The next generation". In: *IBM Systems Journal* 34.3 (1995), pp. 452–471. DOI: 10.1147/sj.343.0452.

[43] Morgan Quigley et al. "ROS: an open-source Robot Operating System". In: vol. 3. Jan. 2009.

[44] ROS. *ROS Wiki.* URL: https://wiki.ros.org/.

Appendix

Appendix A - Movement Sample Examples

Static Movements

Two examples of static movements are shown below through the help of the visualizer. Note that while the data seems highly irregular for a static sample, that is due to how the y-axis is scaled when there is no data in the full range ($0 - 180°$).

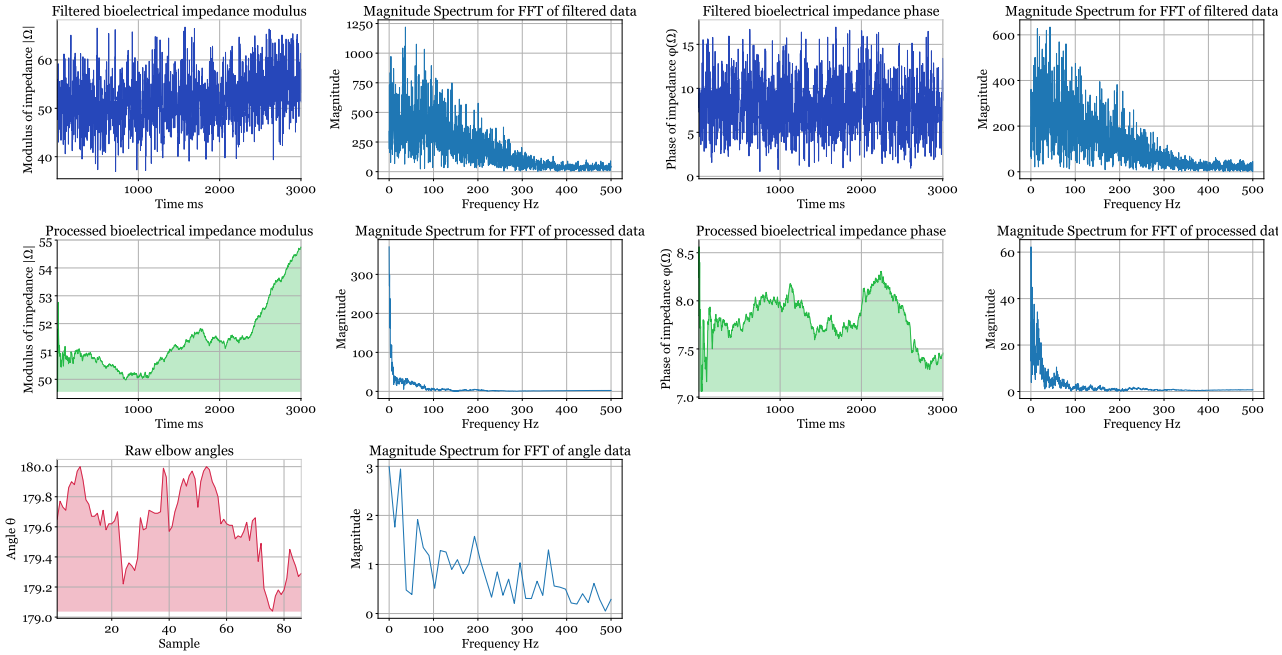


Figure 37: $180°$ - 6 $[kg]$ - Regular Grip

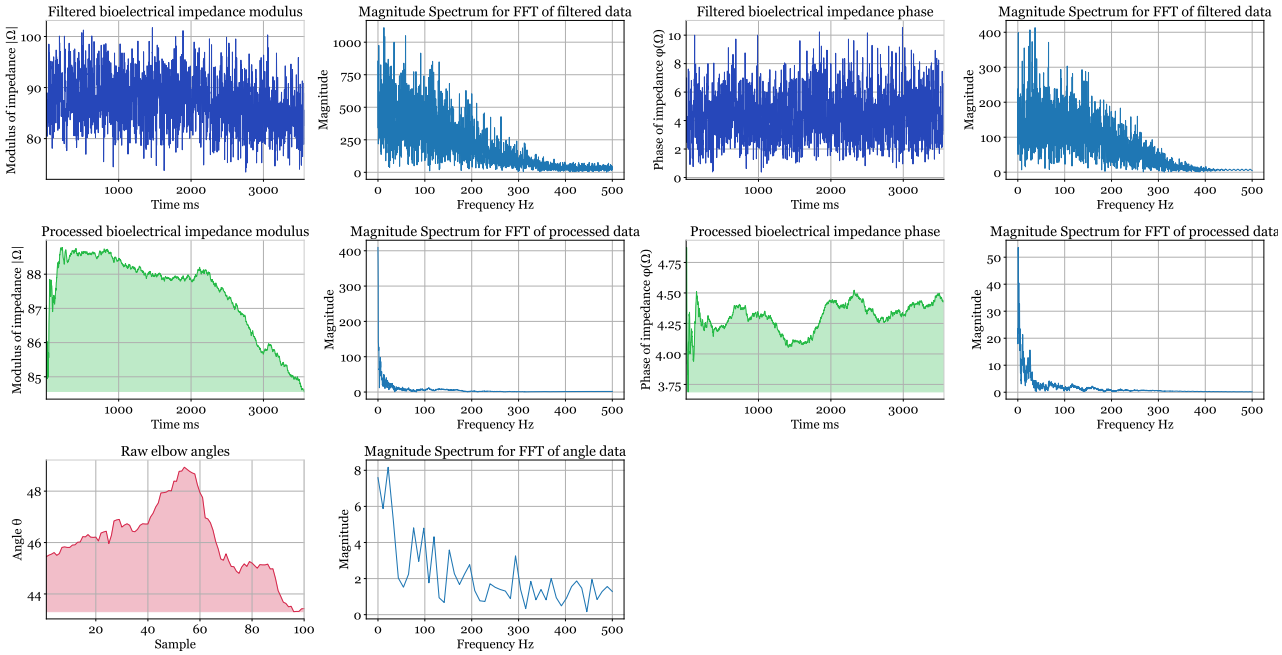


Figure 38: $45°$ - 6 $[kg]$ - Regular Grip

Dynamic Movements

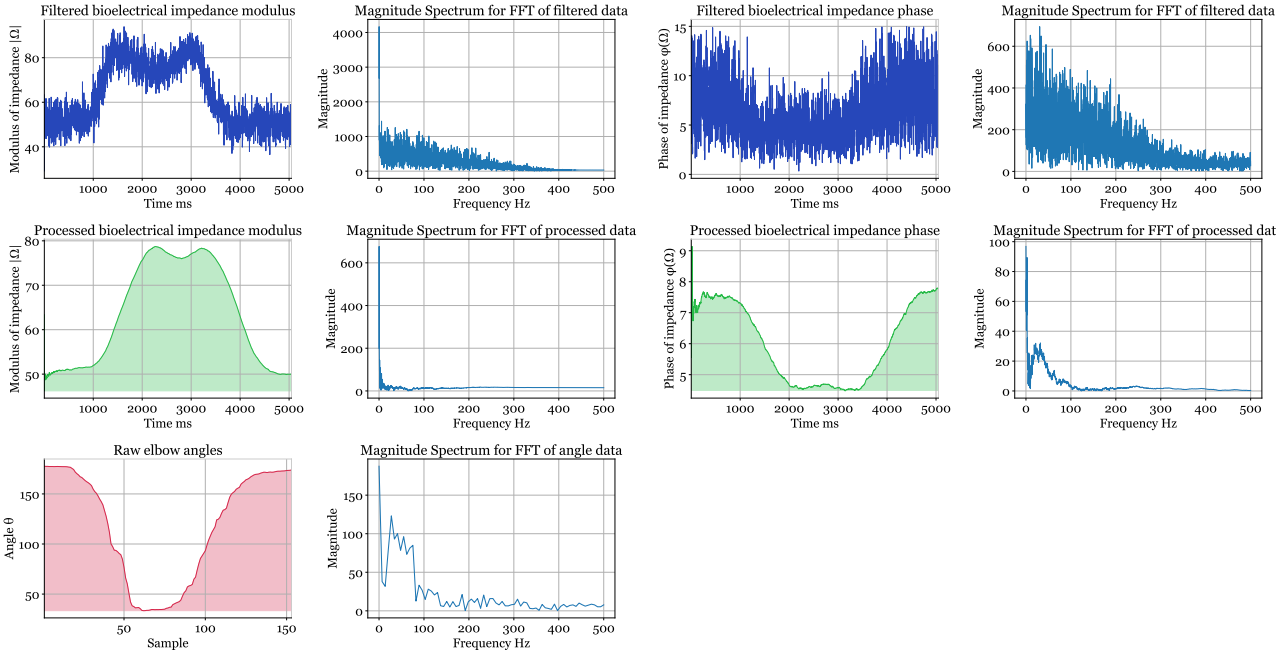


Figure 39: Full Range of Motion - 6 $[kg]$ - Regular Grip

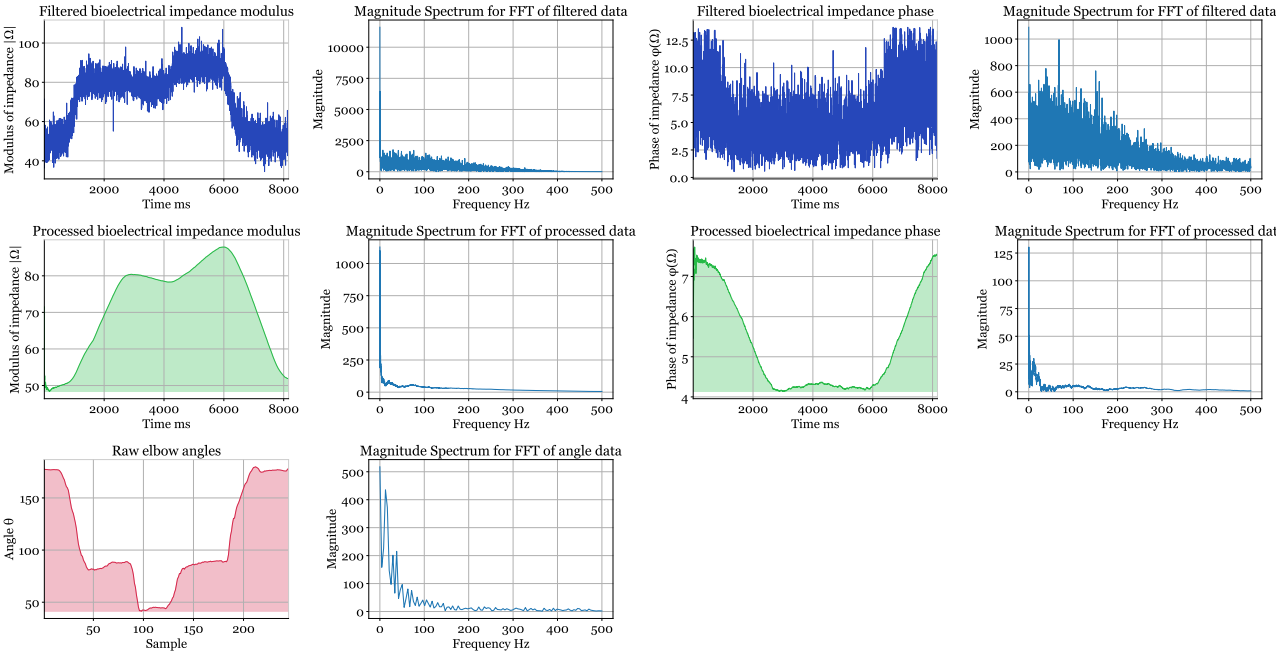


Figure 40: Full Range of Motion (Stop) - 6 $[kg]$ - Regular Grip

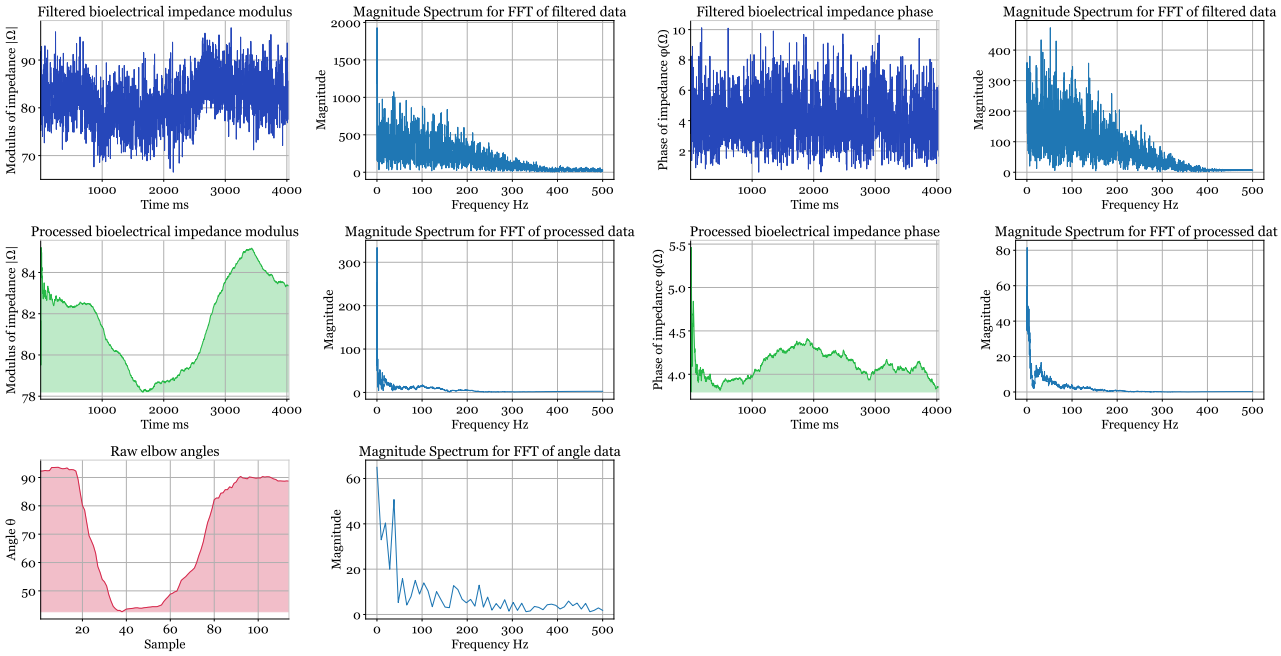


Figure 41: Half Range of Motion (Upper) - 6 $[kg]$ - Regular Grip

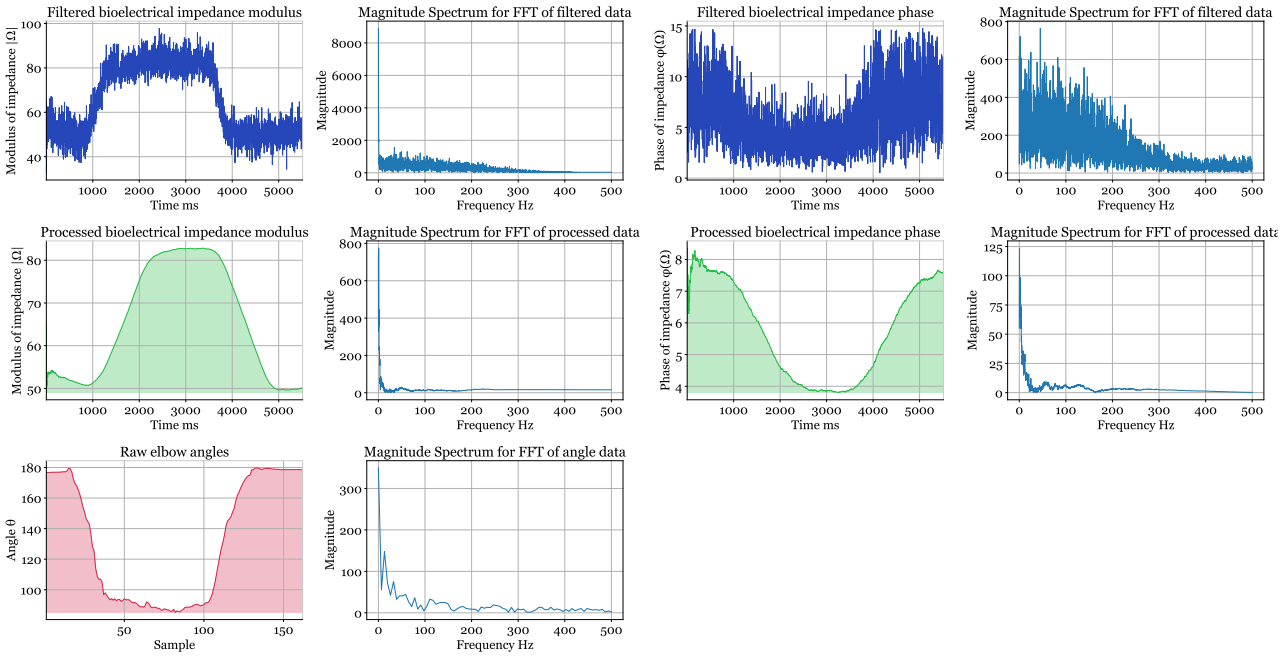


Figure 42: Half Range of Motion (Lower) - 6 $[kg]$ - Regular Grip

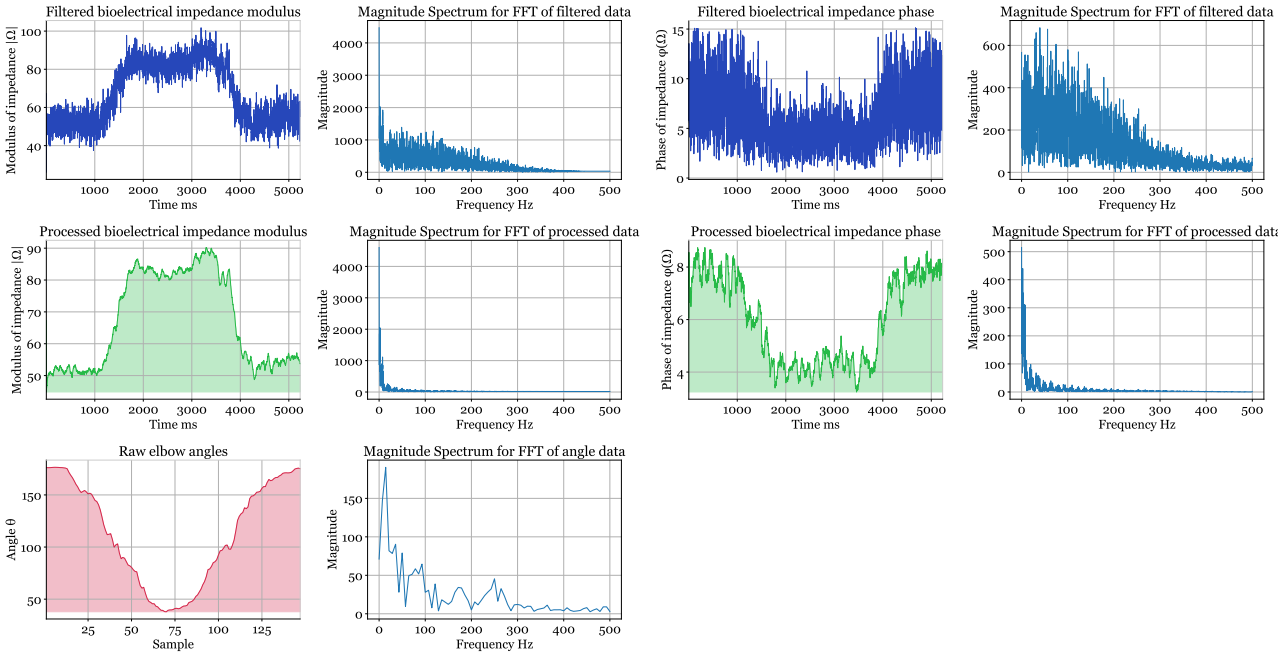


Figure 43: Full Range of Motion (Jitters) - 6 $[kg]$ - Regular Grip

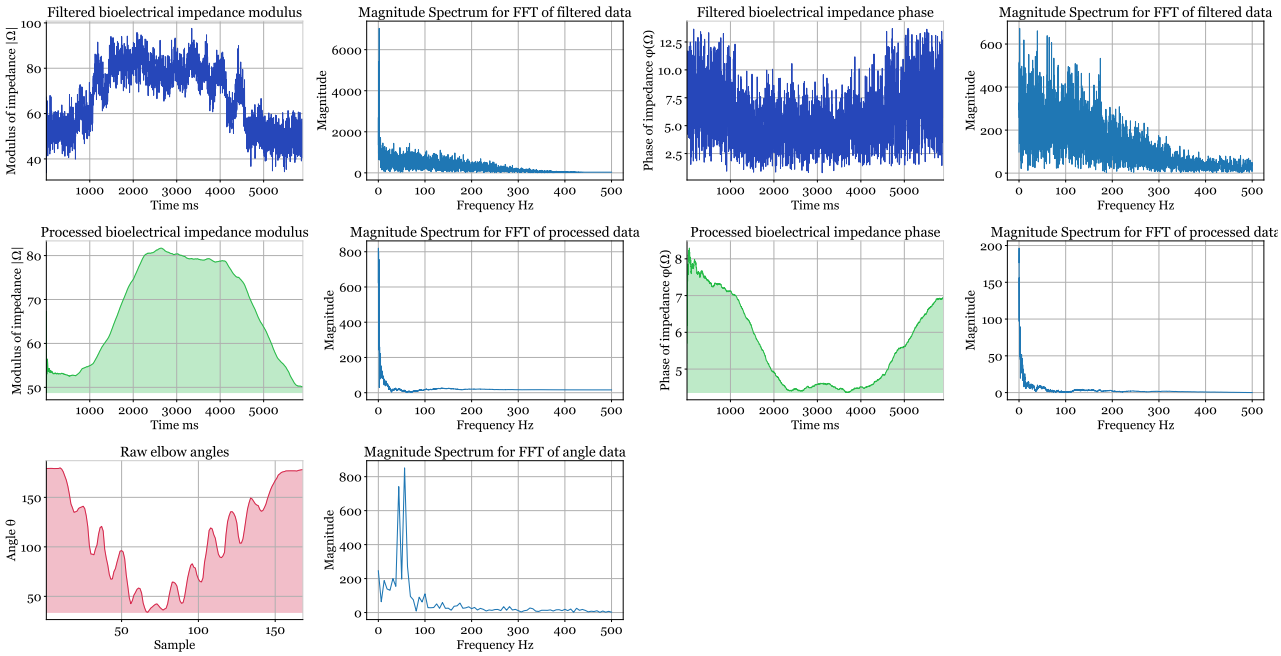


Figure 44: Full Range of Motion (Pulsing) - 6 $[kg]$ - Regular Grip

Appendix B - Interpolation Process Diagram

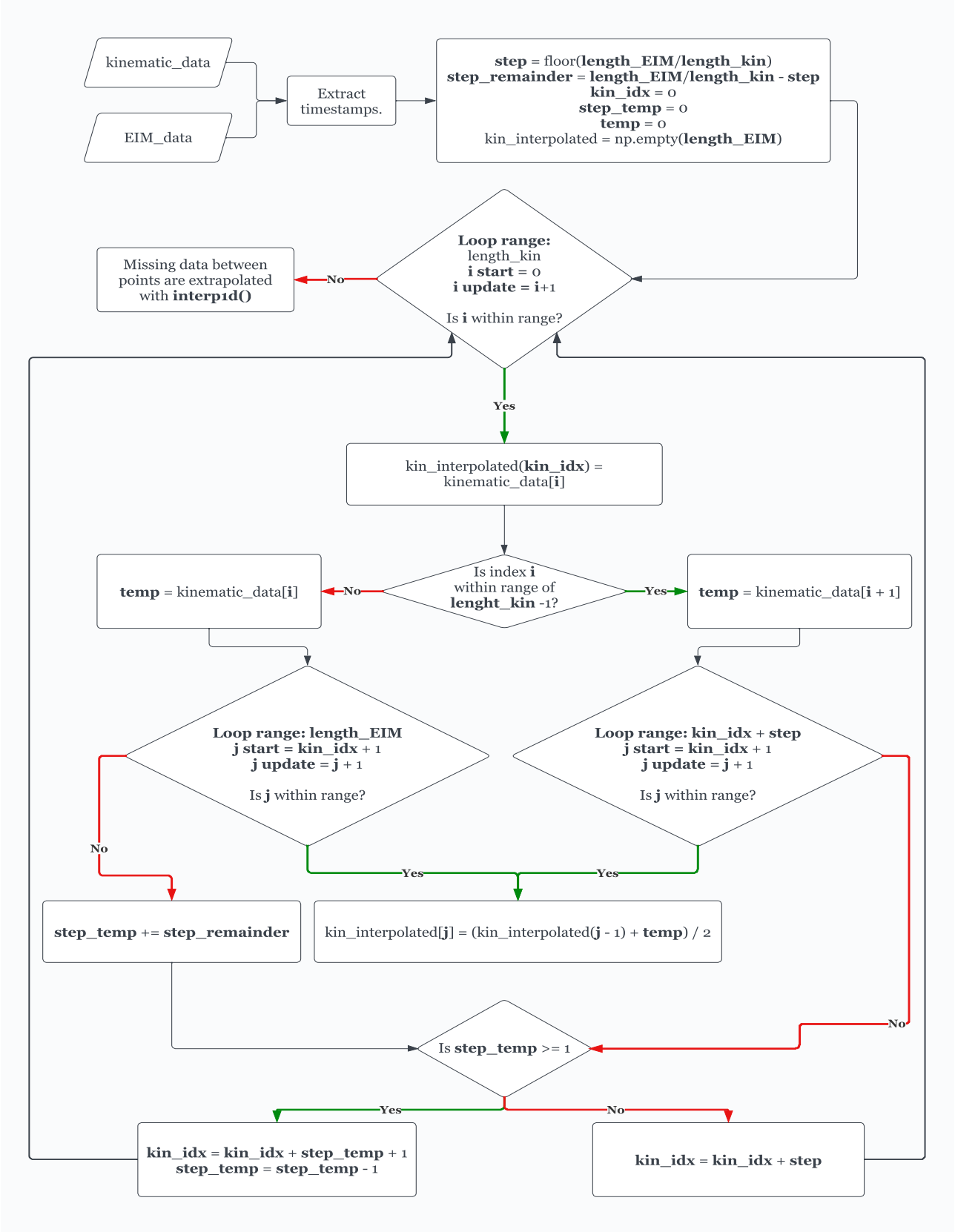


Figure 45: Linear interpolation process of the kinematic data