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 paper 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. They offer a promising technical innovation with potential uses in healthcare, rehabilitation, and a variety of sectors, leading to greater productivity and, more importantly, overall quality of life.

This paper 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 be difficult to 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, brain-machine interfacing (BMI) 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 digestible to a GRU network that will be developed to be able to predict user intent, and eventually integrating it with the exoskeleton to provide both a method to seamlessly transition between AAN and RAN.

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

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:

- DSP Unit: Design and implementation of the EIM signal filter.
- Kinematics Angle Tracker: Configuration of an open-source Mo Cap 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 bio-impedance 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 **5**

1.1 State of the Art . . . 6

1.1.1 RT EEG-EMG HMI-Based Control System for a Lower-Limb Exoskeleton . . . 6

1.1.2 Assistive upper-limb MMI-controlled exoskeleton for severely impaired patients . . . 7

1.1.3 Volitional control of upper-limb exoskeleton empowered by EMG and ML . . . 8

1.1.4 A Wearable, Multi-Frequency Device to Measure Muscle Activity Combining Simultaneous Electromyography and Electrical Impedance Myography . . . 9

1.2 Problem . . . 10

2 Sensor Technology **11**

2.1 Electromyography . . . 11

2.1.1 Surface Electromyography . . . 12

2.1.2 Intramuscular Electromyography . . . 12

2.2 Electrical Impedance Myography . . . 12

2.3 Choice of sensors . . . 13

3 Software Overview **14**

3.1 Architecture . . . 14

3.2 Design . . . 15

3.3 Integration with POW-EXO . . . 16

3.4 Neural Network . . . 17

3.4.1 Feature Engineering . . . 17

3.4.2 Runtime limitations . . . 19

3.4.3 Optimizing the neural network . . . 20

3.4.4 Hyperparameter tuning 24

3.4.5 Calibrating for new subjects 25

4 Discussion **26**

4.1 Future Work 26

4.1.1 Exoskeleton Performance 26

4.1.2 Database 27

4.1.3 Internal Communication 27

5 Conclusion **29**

6 References **30**

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. ALS, Parkinson's disease, muscular dystrophy or 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) of individuals with 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. 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/software stack 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 stack is made up of the following components:

Hardware:

- Eliko Quadra Impedance Spectroscopy device with Single Shunt Front End attached. [2]
- Electrical Impedance Myography sensor electrodes are attached to the Eliko device.
- The POW-EXO exoskeleton.
- A Windows 11 PC to run all the software.

Software:

- Eliko sensor data interface with built-in DSP and statistical post-processing features for the data.
- Kinematic angle tracker, using computer-vision-based human pose estimation to track elbow angles.
- GRU Neural Network that predicts user intent (angle and payload mass) from the processed bioelectrical impedance data gathered from the Eliko interface.
- A visualizer for the data.
- A modified POW-EXO interface with the changes needed to accommodate our data structures.

1.1 State of the Art

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

1.1.1 RT EEG-EMG HMI-Based Control System for a Lower-Limb Exoskeleton [3]

This research presents a rehabilitation technique that employs a lower-limb exoskeleton and a human-machine interface (HMI). The HMI interprets data from a brain-machine interface (BMI) based on motor imagery. Motor Imagery Electroencephalograms (MI 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, which along with leg muscle EMG is used to control the exoskeleton.

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.

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

Existing solutions face challenges such as poor precision in identifying MI EEG and reliability difficulties during exoskeleton-assisted movements. However, this study demonstrates enhanced accuracy, performance, and dependability when employing multi-modal signals to operate the exoskeleton on healthy participants in online real-time.

Existing solutions face challenges such as poor precision in identifying MI EEG and reliability difficulties during exoskeleton-assisted movements. However, 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 study investigates the efficacy of a few EEG electrodes for dependable online control. 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.

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.

The main takeaways from this study are that we should consider more than just a muscle signal to control the exoskeleton, as well as how to properly implement exoskeletons in rehabilitation processes.

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

The Bridge exoskeleton, designed for individuals with muscular dystrophy, aims to enhance independence and quality of life. 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 hand movements in a three-dimensional space. 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. Usability, evaluated through the System Usability Scale, was rated as good.

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 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.

The key findings we can take away from this study are the optimization of actuator torque, as well as the general improvement of independence and quality of life for movement-impaired individuals.

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% accuracy 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 gave important insights into the challenges of deploying an ML model on embedded systems. In a robotic context, there are constraints in terms of system performance accuracy, user experience, energy consumption, and processing speed. It also highlighted 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.

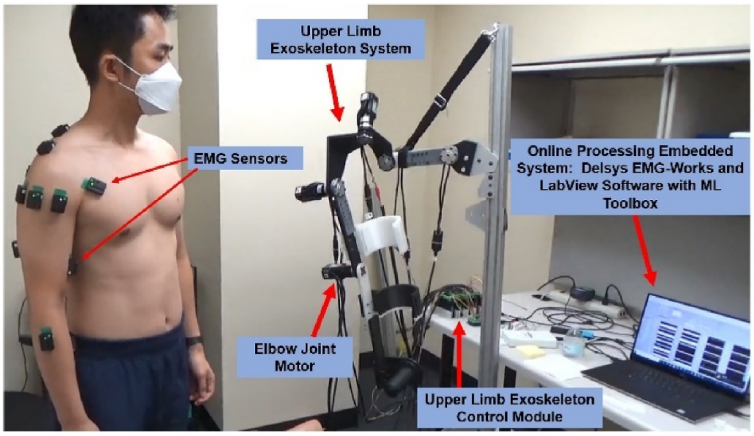


Figure 3: EMG and ML-controlled upper-limb exoskeleton.

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 W 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 an SNR of 67 dB. Every 1 ms, the system achieves data transmission speeds of $500\ kbps$ across RF. 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 or right leg drive. For successful robotics control, the study underlines the need to maintain an overall signal-to-noise ratio (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 tests during a variety of activities, demonstrating its use as a tool for investigating muscle states and force or torque. The modular architecture enables module combinations and simultaneous multi-frequency bioimpedance measurements. Overall, the instrument offers a viable platform for studying muscle states and comprehending muscular force and torque.

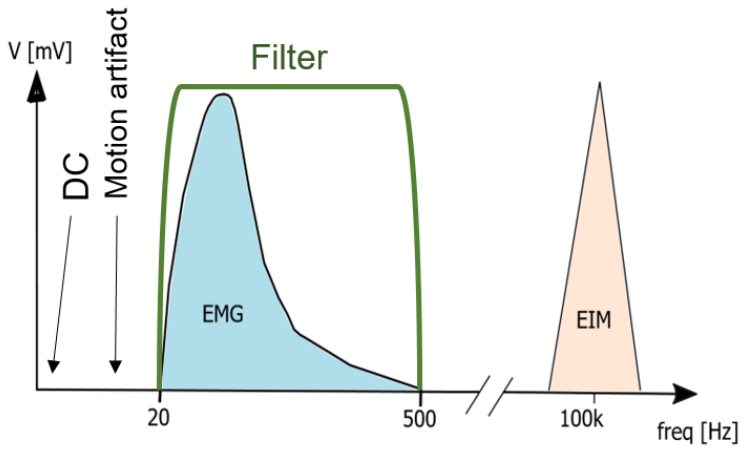


Figure 4: Spectrum of EMG and EIM signals captured at the same electrodes.

1.2 Problem

The main problem to solve with 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 PID 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 stack can be split into the following problems:

1. What are the relevant current-day use cases of sEMG, EIM, and EIT?
2. What are the pros and cons of using either of the three 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?
4. After post-processing the raw sensor data, how can it be transformed in order to use it in the exoskeleton's PID controllers?
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 different kinds of sensor technology in order to pick one to work with in the project.

2.1 Electromyography

Electromyography (EMG) measures muscle contraction and outputs a signal, which is proportional to the amount of muscle contraction; The larger the recorded tension amplitude, the stronger the muscle contraction, and the more muscles are activated. This is because when a muscle contracts, a burst of electrical activity is generated, which could be recorded from nearby tissues and bones. [9]

This technology is used to detect differences in current in the motor cortex, where brain activity signals the spinal cord and information about movement is transmitted to the relevant muscle via motor neurons. Upper motor neurons send signals to lower motor neurons, which initiate muscle activity by innervating the muscle at the neuromuscular joint. The neuromuscular joint, or neuromuscular junction, is a type of synapse where neuronal signals from the brain or spinal cord interact with skeletal muscle fibers, causing them to contract [9] [10] [11].

This process involves depolarization, a phenomenon in biology where a cell undergoes a shift in electric charge distribution, resulting in less negative charge inside the cell compared to the outside [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 tied to the concept of an electrochemical gradient, which is a gradient of electrochemical potential, usually for an ion that can move across a membrane. The gradient consists of two parts: the chemical gradient, or difference in solute concentration across a membrane, and the electrical gradient, or difference in charge across a membrane [14]. When there are unequal concentrations of an ion across a permeable membrane, the ion will move across the membrane from the area of higher concentration to the area of lower concentration through simple diffusion [14]. This change in the electrochemical gradient is essential for the function of many cells and the overall physiology of an organism [12].

EMG and force sensors are noisy and vary depending on the application. Estimating interaction torques from patient EMG and force data can be problematic due to irregular EMG-torque connections. EMG/FSR sensors can analyze data to categorize upper-limb motions, but adequate evaluation depends on user condition and muscle group selection. EMG-based control approaches are ineffective in cases of tremor or spasticity, making it crucial to consider these factors when designing robots. [15] [16] [17] [18]

The biological EMG signal, produced when muscles contract, is crucial in resolving human motion intentions. EMG-based control, which relies on the user's skin surface signals, is an effective method for assisting robotic systems, particularly power-assist exoskeleton robots, as it directly reflects the user's motion intention. [19]

2.1.1 Surface Electromyography

There are two main types of EMG sensors: intramuscular and surface EMG. The letter "s" in "sEMG" represents surface, where this type of EMG sensor collects data by applying EMG electrodes to the surface of the skin. This type of EMG sensor uses non-invasive technology, so it is painless. Commonly used in clinics and sports medicine. Although this is a quick and easy method of measuring EMG, it only applies to superficial muscles and depends on other factors such as the patient's weight, etc. Placement of the EMG sensor, performed in the area innervating the two tendons on top of the skin for better detection quality, is the first step of the procedure (Figure 5). The electrodes begin to detect electrical activity generated by muscle contraction or movement. [20]

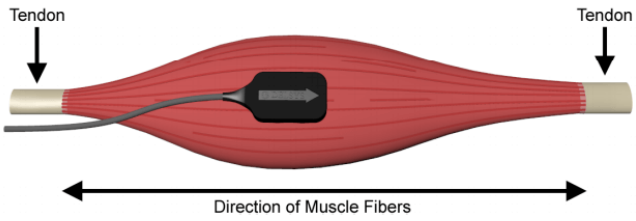


Figure 5: sEMG — Sensor placement illustration without the skin.

2.1.2 Intramuscular Electromyography

During the procedure, the sterile needle electrode captures electrical activity within the muscle, providing valuable insights into muscle function and nerve communication. The monopolar needle electrode ensures precise readings by penetrating the skin and reaching the target muscle tissue; see Figure 6. [20]

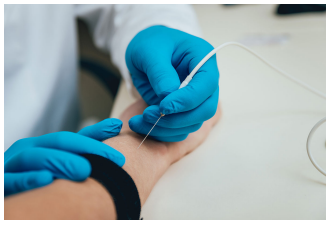


Figure 6: Intramuscular EMG — Monopolar Needle Electrode.

2.2 Electrical Impedance Myography [21] [22]

Electrical impedance myography (EIM) is a century-old technique that has evolved from estimating tissue volumes and evaluating skin lesions to assessing nerve and muscle conditions. It studies 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, providing insights into disease dependence. However, measurements beyond isometric contractions can be challenging due to changes in muscle or limb shape. Techniques, like impedance imaging, have been developed to overcome these challenges. [21]

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. 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. In terms of electrode usage, EIM typically employs four electrodes, with silver/silver chloride electrodes being the most common choice. Alternatives include dry electrodes, which prevent inadvertent contact between adjacent electrodes, and textile dry electrodes for long-term EIM monitoring during daily activities. Despite these advancements in technology, challenges remain in optimizing EIM's application. Further research is needed to refine electrode configurations, reduce the impact of subcutaneous fat, ensure effective current penetration into muscles, and capture muscle anisotropy. Muscle anisotropy refers to the different resulting echogenicity of soft tissues, such as tendons, when the angle of the transducer is changed. In ultrasound imaging, anisotropy is an artifact encountered, notably in muscles and tendons. It can affect the interpretation of the ultrasound image, potentially leading to an incorrect diagnosis. Therefore, capturing muscle anisotropy is crucial for accurate EIM measurements. [22]

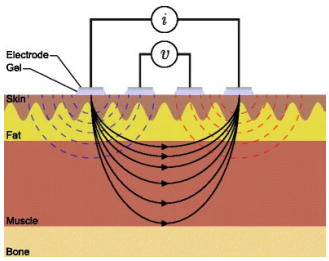


Figure 7: EIM — Four gel-adhesive electrodes to measure electrical impedance and illustrate the potential generated by current flow through tissue.

2.3 Choice of sensors

Based on the research done on the two types of sensors, it was chosen to work with EIM on the project. Additionally, EIM sensors provide a non-invasive and more comfortable alternative, eliminating the need for electrodes penetrating the skin (see Figure 7). This lack of intrusion contributes to increased user acceptance and long-term usability. Furthermore, EIM sensors have a higher spatial resolution, allowing for a more comprehensive assessment of muscle activity over a larger area, which is critical for capturing complex movements involving multiple muscle groups. Furthermore, EIM is less susceptible 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 enhanced signal quality enables finer control of the exoskeleton, allowing for a more natural and intuitive human-machine interface. In conclusion, the use of EIM sensors in this scenario is consistent with a user-centric approach that prioritizes comfort, accuracy, and ease of use for optimal human-machine interaction.

For an in-depth guide on how to set up and use the MUSE stack with EIM electrodes and the POW-EXO, refer to the Experimental Protocol in Appendix A.

3 Software Overview

This section explores the overview, architecture, design philosophy, and different concrete software components developed for use in this project.

3.1 Architecture

Along with the Eliko Quadra device, a C++ program was acquired, which interfaces with the device and connects data from the connected sensors. The program consisted of:

- PicometerReader.cpp — The main program, which interfaces using Eliko's API
- PicometerCtrl.dll — A dynamic link library providing picometer control functions
- PicometerCtrl.lib — Links the C++ project to PicometerCtrl.dll
- resource.h — Auto-generated Microsoft Visual C++ resource file settings
- pre_defined_params.h — Commands and parameters for a device communication protocol.

The software architecture was 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 through different methods, 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.

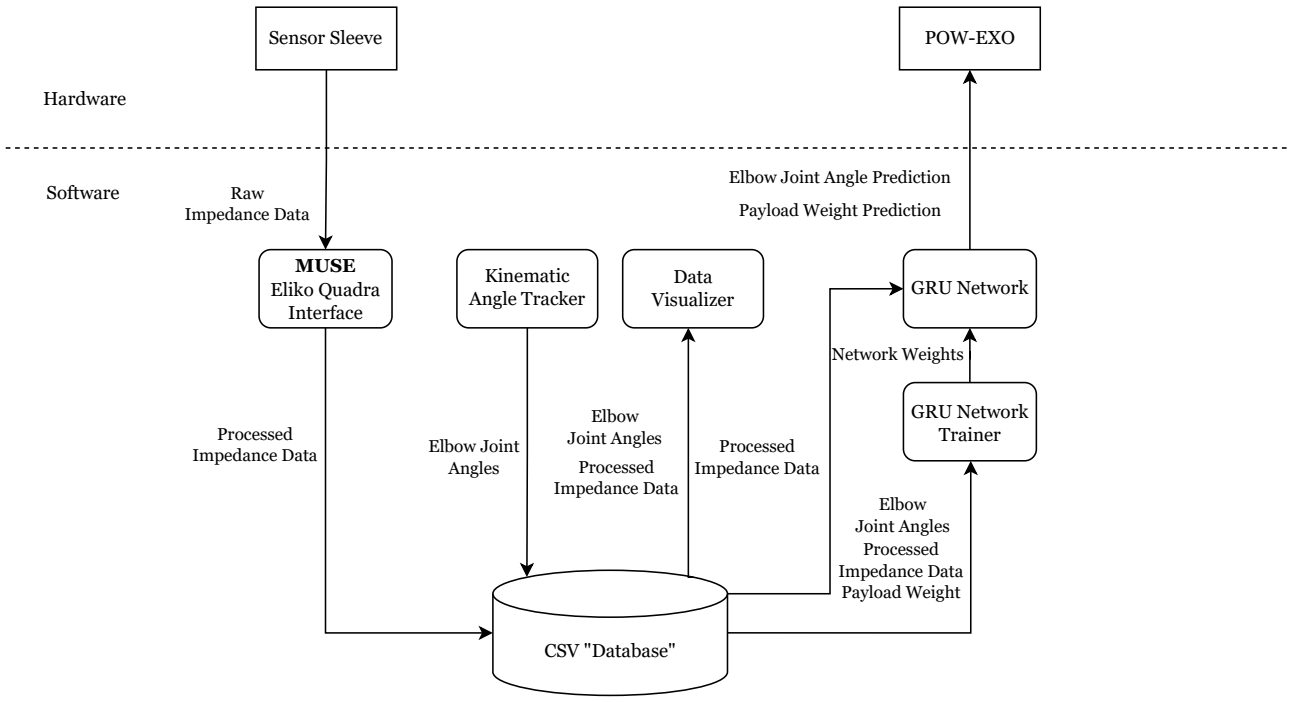


Figure 8: Diagram of Software Architecture.

On a lower, more practical level, we made sure to keep the main program file as free from bloat as possible, once again improving the -ity's. We accomplished this by having it only take care of the main status loop and state machine while refactoring file system handling, data processing, and Eliko Quadra interfacing into their own respective classes and header-only files.

3.2 Design

It was decided 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.

As previously stated, MUSE is designed around a state machine paradigm, which provides a simple and orderly method to interface with a sensor device and process the collected data. 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. User input and the changing context of data collection determine state transitions. Finally, the application manages memory dynamically by allocating and de-allocating a PicometerController object.

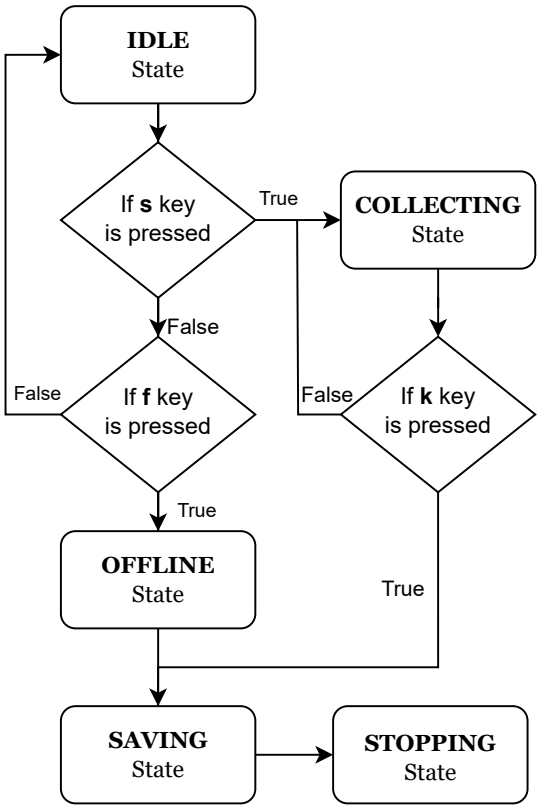


Figure 9: Muse State Machine Flow.

The design encourages 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.

3.3 Integration with POW-EXO

This section explains the logic behind the exoskeleton integration, which has the form of an interface between the GRU network and the exoskeleton control code.

Integrating with the POW-EXO starts out with modifying the already existing control code for the exoskeleton. The modifications should import the GRU network, convert the predictions into valid references for the POW-EXO control system, and then pass them through to it. The flow through the interface is seen in Figure 10 and is explained in detail below.

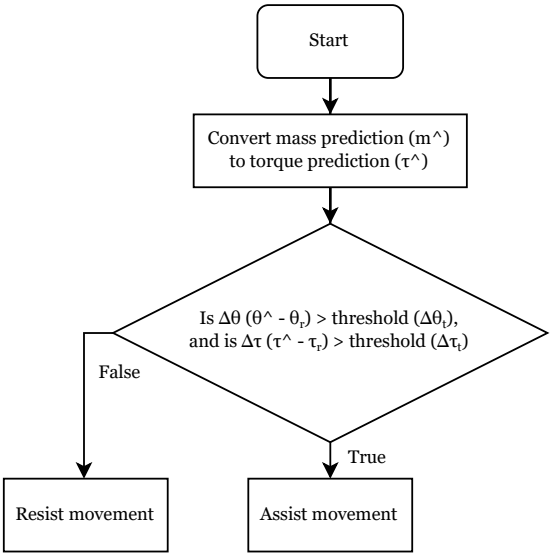


Figure 10: Exoskeleton Interface Flowchart.

From the GRU network, the desired references $\hat{\theta}$ & $\hat{m}$ are predicted. Then $\hat{m}$ needs conversion to torque τ_r, whereas $\hat{\theta}$ can be used directly as a reference θ_r.

Now we need to determine whether the two individual references should be updated, which is done by measuring them against minimum reference change thresholds $\Delta\theta_t$ & $\Delta\tau_t$, and only updating the references if the change is larger than the threshold.

Similarly, whether the current predictions require the **Resist As Needed** (RAN) or **Assist As Needed** (AAN) control loops is determined in the same conditional statements. This flow is seen in Figure 10.

3.4 Neural Network

3.4.1 Feature Engineering

This section discusses and justifies the extraction of different features from the experiments. For a complete overview over all the features used for developing the model, see table 3.4.1.5.

3.4.1.1 Sliding Window Averaging

Sliding Window Averaging, more commonly known as a moving average, is distinct from a regular running average. It is a data analysis method that finds various local averages based on a moving window, the size of which determines the number of averages and, in turn, the smoothness of the resulting data set. It acts as a sort of low-pass FIR filter without having to deal with the frequency domain. The purpose is to smooth out variations in the signal and highlight underlying trends or patterns. This requires a minimum number of iterations of input variables equal to the size of the buffer, before which the data is typically left raw or run through a regular running average.

For our purpose, a regular running average was decided as the most suitable way of filling out the data points before the sliding window averaging takes over. The implemented code is explained with equations 1 and 2.

$$k = \frac{n}{c} \tag{1}$$

$$y_i = \begin{cases} \frac{1}{i+1} \sum_{k=0}^{n} (x_i) & \text{if } i < k \\ \frac{1}{k} \sum_{i=n-k+1}^{n} (x_i) & \text{if } i \geq k \end{cases} \tag{2}$$

- 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 constant determined by the user.
- k: Is the computed buffer-size, determining the size of the moving window.

This feature was implemented in the early stages of the project as an attempt to further smooth out the signal after applying the FIR filter. See figure 11 for more details.

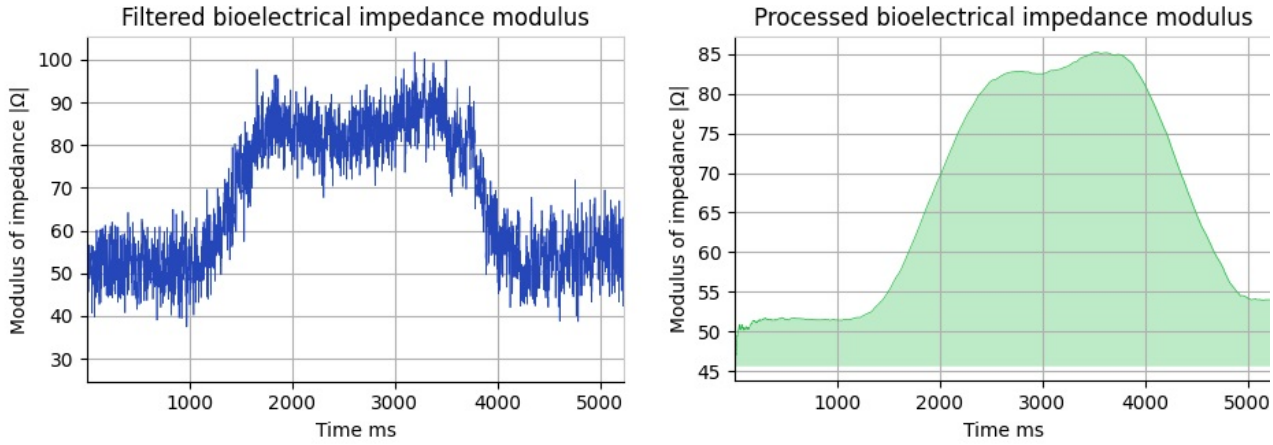


Figure 11: Image of a 6 kg FROM curl with jitters. Left: Filtered bioelectrical impedance's magnitude. Right: Corresponding sliding window averaging with $c = 5$.

As seen in Figure 11, the sliding window averaging helps in determining whether a given change in bioelectrical impedance is real or noise. Additionally, in the image, $c = 5$, which was later changed to $c = 75$, since we were worried about losing information from smoothing out the data too much.

3.4.1.2 Rate of change

lorem ipsum

3.4.1.3 Mean and median

lorem ipsum

3.4.1.4 Standard deviation and variance

lorem ipsum

3.4.1.5 Kurtosis

lorem ipsum

List of features
Median: Extracted from the EIM magnitude and phase. Giving us **2 features**
Mean: Extracted from the EIM magnitude and phase. Giving us **2 features**
Standard deviation: Extracted from the EIM magnitude and phase. Giving us **2 features**
Variance: Extracted from the EIM magnitude and phase. Giving us **2 features**
Kurtosis: Extracted from the EIM magnitude and phase. Giving us **2 features**
Sliding window average: Extracted from the EIM magnitude and phase. Giving us **2 features**
Rate of change: Extracted from the sliding window averages of EIM phase and EIM magnitude **2 features**
EIM Magnitude, EIM Phase: Elements from the unmodified EIM signal Giving us **2 features**

Which gives a total of 16 features to train on, where the two targets consist of the elbow joint angle and that the subject is lifting.

3.4.2 Runtime limitations

To use the GRU model, the data must be reshaped into sequences using a time step variable as an indicator of the number of data points that the model must evaluate while training. This enables the model to detect planned changes in the signal with greater accuracy.

However, creating the sequences is a computationally intensive process, $O(NMP)$, where N is the number of samples in the data set, M is the number of features represented by columns, and P is the number of data points in each sample. This is fine if the data set is small and contains few characteristics, but the difficulty soon grows as the number of data points and features increases, as does the time step value. We collected almost $500,000$ data points in our experiment, and the initial iteration of feature engineering had 24 features. See figure (insert figure reference).

A test was run to see how long it would take to create sequences of $1,000$ data points, each representing approximately one second of data, with varied time step levels and feature counts. This was accomplished utilizing a Lenovo IdeaPad C340-14IWL and the Jupyter notebook's built-in cell timer.

Sequencing 1,000 data points	**time step = 10**	**time step = 20**	**time step = 30**	**time step = 40**
features = 1	$0\ [s]$	$0\ [s]$	$0\ [s]$	$0\ [s]$
features = 2	$0\ [s]$	$0\ [s]$	$0\ [s]$	$0\ [s]$
features = 3	$0\ [s]$	$0\ [s]$	$0\ [s]$	$0\ [s]$
features = 4	$0\ [s]$	$0\ [s]$	$0.1\ [s]$	$0.1\ [s]$
features = 5	$0\ [s]$	$0.1\ [s]$	$0.1\ [s]$	$0.2\ [s]$
features = 6	$0.1\ [s]$	$0.1\ [s]$	$0.2\ [s]$	$0.6\ [s]$
features = 7	$0.1\ [s]$	$0.2\ [s]$	$0.3\ [s]$	$1.1\ [s]$
features = 8	$0.1\ [s]$	$0.2\ [s]$	$0.9\ [s]$	$1.8\ [s]$
features = 9	$0.2\ [s]$	$0.3\ [s]$	$1.4\ [s]$	$2.4\ [s]$
features = 10	$0.2\ [s]$	$0.4\ [s]$	$2.3\ [s]$	$3.7\ [s]$

As we expected, the problem worsens exponentially as the number of time steps and attributes rises. However, when more data is provided, the situation deteriorates. The table below demonstrates what happens to processing time when the number of data points increases by $1,000$. The number of features will be restricted to ten to keep things simple.

Sequencing 10 features	time step = 10	time step = 20	time step = 30	time step = 40
2,000 data points	1.3 $[s]$	9.4 $[s]$	14 $[s]$	19.1 $[s]$
3,000 data points	9.5 $[s]$	23.3 $[s]$	39.1 $[s]$	47.8 $[s]$
4,000 data points	22.3 $[s]$	59 $[s]$	1 $[m]$ 18.8 $[s]$	1 $[m]$ 37.1 $[s]$

The computation improved slightly after a few changes to the helper function that generated the sequences that helped eliminate the loop, but it was still too slow when all features were included. It was decided to relocate the code to Google Colab, which has more processing capability. It still took 13 minutes to create sequences with all characteristics and all data using 10 time steps.

The helper function was subsequently removed and replaced with a much simpler solution that just used one loop. This significantly reduced the runtime, which was reduced to 1.7 seconds for the entire dataset with temporal sequence length (SL) of 10. Nonetheless, the duration of constructing sequences should be considered throughout the implementation of the exoskeleton, when predictions must be generated in real time. Of course, the shorter the computing time, the smaller the sample size for each prediction.

3.4.3 Optimizing the neural network

Because the target hardware is a small embedded device, it was critical to keep the model's memory use to an absolute minimum while still solving the problem. As a result, the first training was performed with a relatively simple architecture, consisting of a single hidden layer with 32 neurons and a ReLU activation function, and an output layer with a linear activation function. All 16 characteristics were used in the training, with an SL of 10. The loss function is mean-squared error, and the optimizer 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 12: The first GRU model architecture.

The objective for utilizing an adaptive optimizer is to avoid having to tweak the learning rate for each training and to keep the optimizer from being trapped on a plateau or local minimum, which is a danger of using stochastic gradient descent (SGD). ADAM has shown to be one of the greatest adaptive optimizers, storing an exponentially decaying average of historical gradients. Similarly to momentum. The trade-off is a little lower anticipated generalization than when training with SGD. SGD takes more intensive training than ADAM to attain better outcomes. Another rationale for selecting an adaptive optimizer was that the project time restriction did not allow for significant training experimentation. [23]

The reason for adopting mean-squared error (MSE) as the loss function is because it is the preferable loss function when the distribution of the target variable is Gaussian, which is what the elbow angle from biceps curl samples almost looks like.

It is a popular choice for regression problems, and in our situation, we want to predict the continuous values of the joint angle and mass, which is where MSE comes in handy. The squared difference between the expected and actual values is calculated. Because the mistake is squared, the outcome is always positive. This also implies that greater errors result in more errors than smaller errors, penalizing the model severely for making mistakes. [24]

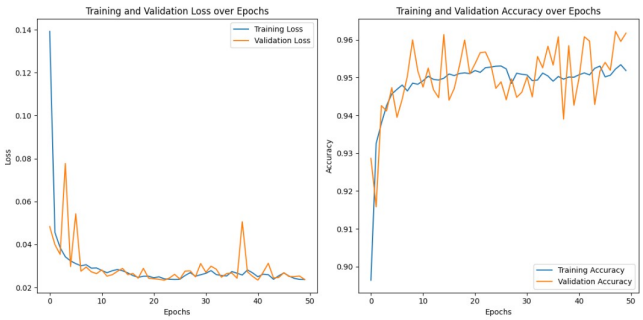


Figure 13: Convergence of Loss and Accuracy for the first GRU model.

The first training was done on 200 epochs, but the trend was already apparent after 50, therefore the number of epochs was reduced to 50 in the interest of reducing time spent.

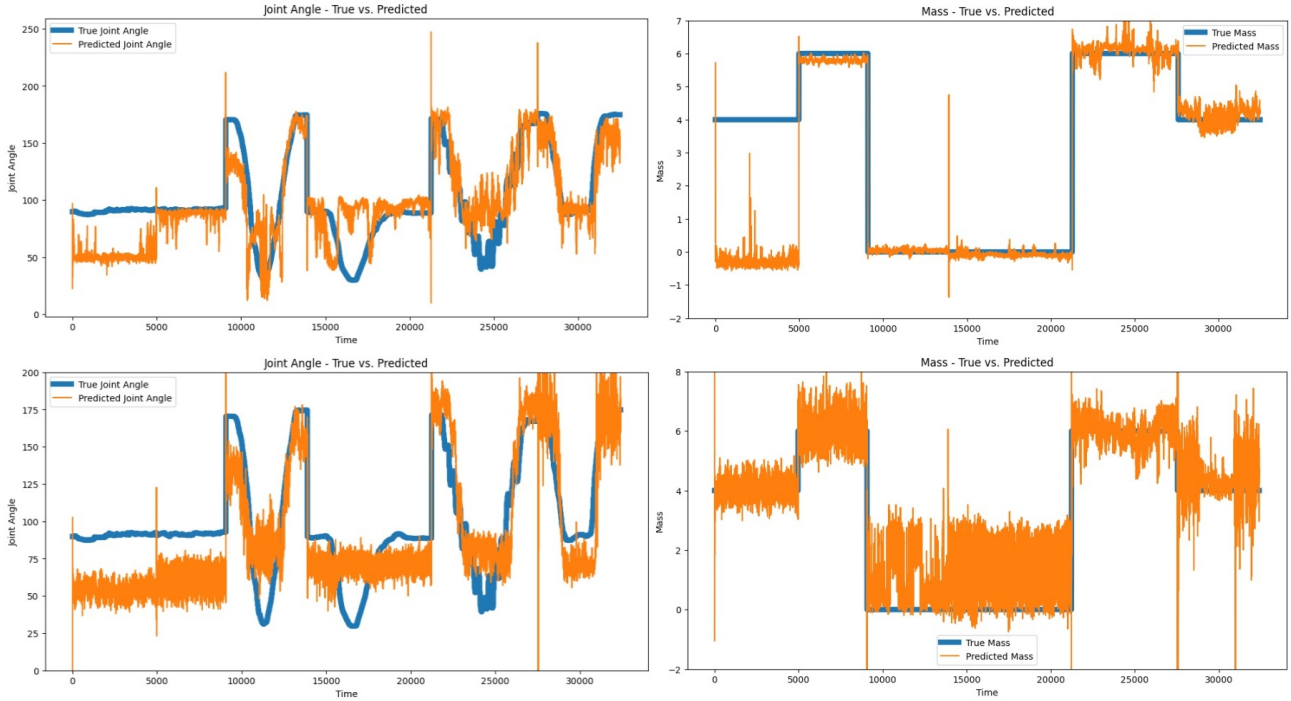
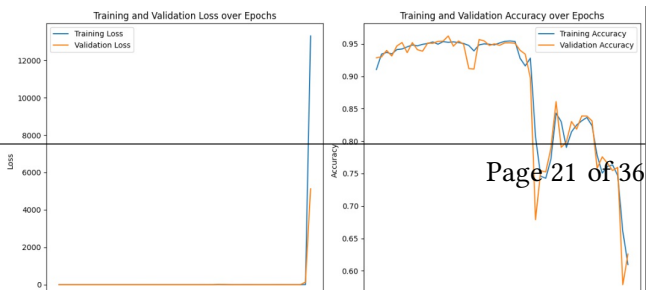


Figure 14: The first two GRU models predicting test data. Top row: Model with SL of 10. Bottom row: Model with SL of 20.

The model stabilized at 0.03 loss and 96% during training, which appeared positive on the surface, but when the model was given to a test set that it had never seen before, the accuracy plummeted drastically to 75.27% and loss was 0.36, suggesting overfitting. By raising the SL to 20, the problem did not improve much, achieving 95% accuracy and 0.035 loss at best, but then the accuracy plummeted to 84.37% at the conclusion of training. On the test set, however, we saw an increase in accuracy to 80.64% and a loss of 0.33. This might imply that raising the SL would increase generalization.

However, the visualization of the test data, as shown in Figure 14, reveals something fascinating.



While training with an SL of 20 appears to generalize better in terms of mass, it also appears to generate significantly more noise than training with an SL of 10.

To test this hypothesis, one last training with an SL of 30 was performed before modifying the model architecture to address overfitting.

While GRU solves the vanishing gradient problem (VGP) 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 quick increase in loss and the unstable training behavior during training are indicators of this. See figure 16. This might be owing to insufficient weight initialization or an excessively fast learning rate. Too big initial weights might result in significant gradients during training. Using an extremely fast learning rate may cause the model to update weights with excessively large gradients, resulting in instability and divergence.

VGP can still occur when the SL grows extremely lengthy. If the sequence is too lengthy, the gradients may reduce as they propagate over time, making long-term dependencies harder for the model to understand.

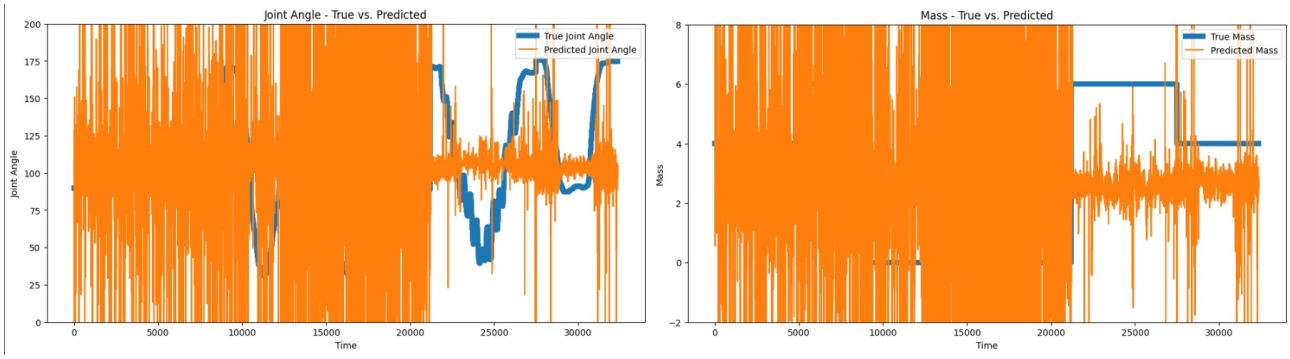


Figure 16: GRU model trained with SL of 30, showing the effect of EGP.

As seen in Figure 16, we have found the culprit of the noise. In fact, the model looses its ability to generalize completely as the training progresses.

Gradient clipping was used to remedy this by limiting the amount of the gradient during back-propagation. The clipping threshold was set to a value of ± 0.5. Furthermore, the starting 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 is intended for ReLU activation, where it initializes weights from a normal distribution. The choice was taken whether to adopt a normal or uniform distribution, with the normal distribution being chosen since it was less prone to statistical outliers and noise.

```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 Input-Layer (GRU)           (None, 30, 32)            4800

 dropout (Dropout)           (None, 30, 32)            0

 gru_1 (GRU)                 (None, 32)                6336

 dropout_1 (Dropout)         (None, 32)                0

 dense_6 (Dense)             (None, 2)                 66

=================================================================
Total params: 11202 (43.76 KB)
Trainable params: 11202 (43.76 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
```

Another GRU layer with 32 neurons was added in the hopes of improving accuracy, and dropout was used as a regularization approach on both the recurrent connections and each layer to address the issue of overfitting. Refer to 17.

Another training was performed with an SL of 30 and a dropout rate of 20% on all parameters to examine the impact of the adjustments.

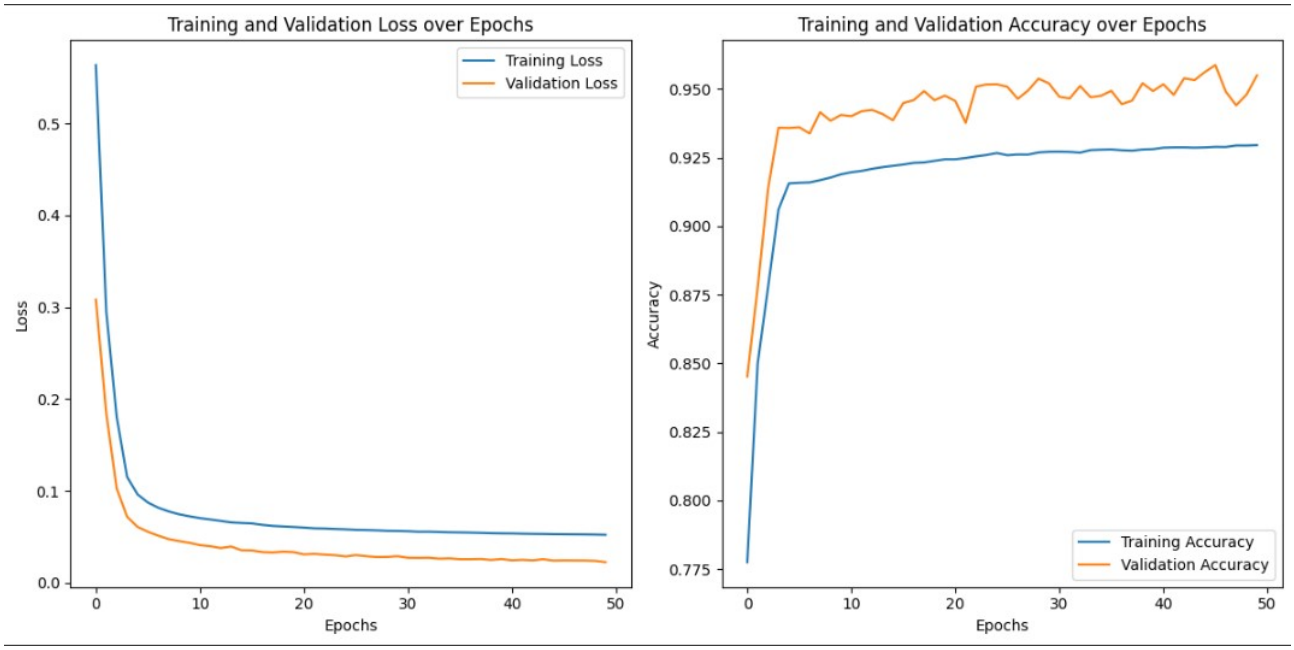


Figure 18: New model with changes address EGP, with an SL of 30 and Dropout of 20% across all layers.

As seen from the smooth convergence without any irregularities in Figure 18, the EGP appears to be stable for the time being, with an accuracy of 93% and a loss of 0.052. On the test data, this reduced to an accuracy of 86% and a loss of 0.167, which is still the best generalization seen thus far, but also an indicator of overfitting.

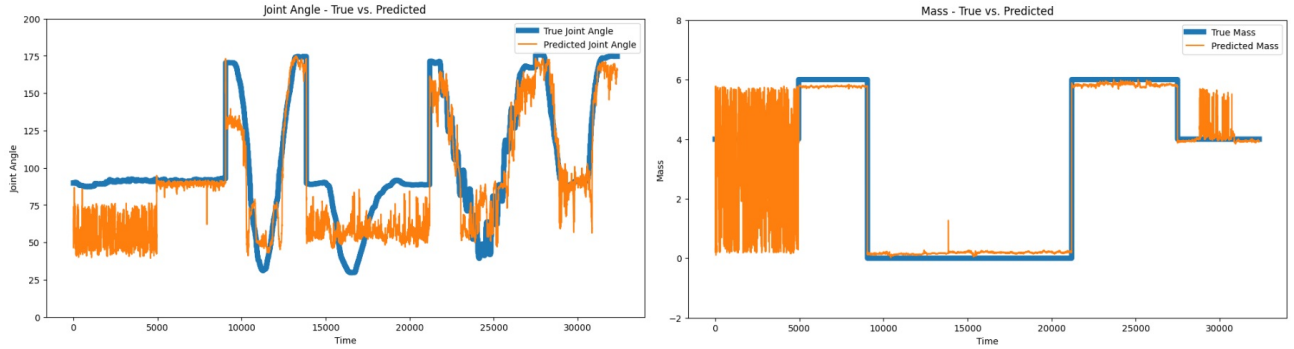


Figure 19: Second model with an SL of 30. EGP has been addressed, but weird behavior is still occurring between time steps 0 and 5,000, and at 30,000.

While figure 19 is promising in general, especially in terms of mass prediction, the joint angle appears to require significant improvement. We also detect some produced noise, which is expected to disappear as

generalization improves.

To reduce overfitting and enhance model accuracy, batch normalization was introduced on each layer, coupled with L2 regularization, and the number of neurons in the second layer was increased to 64, with a dropout rate of 40% instead of 20%.

The goal of adopting batch normalization is to normalize the activations from each layer so that the gradient descent may converge more quickly during training, hence further stabilizing the training. [25]

L2 regularization is preferred over L1 regularization because L1 (also known as Lasso in regression problems) effectively nullifies features having a non-significant contribution to the correlation between features and target. Because we still don't know which traits are most important for estimating angle and mass, which may change for each target, L1's total elimination of data is undesirable for the time being. In contrast, L2 (or Ridge) still allows for a minor contribution of less important traits. The disadvantage of adopting L2 over L1 is that L2 performs poorer on outliers, which should be kept in mind for future training. [26]

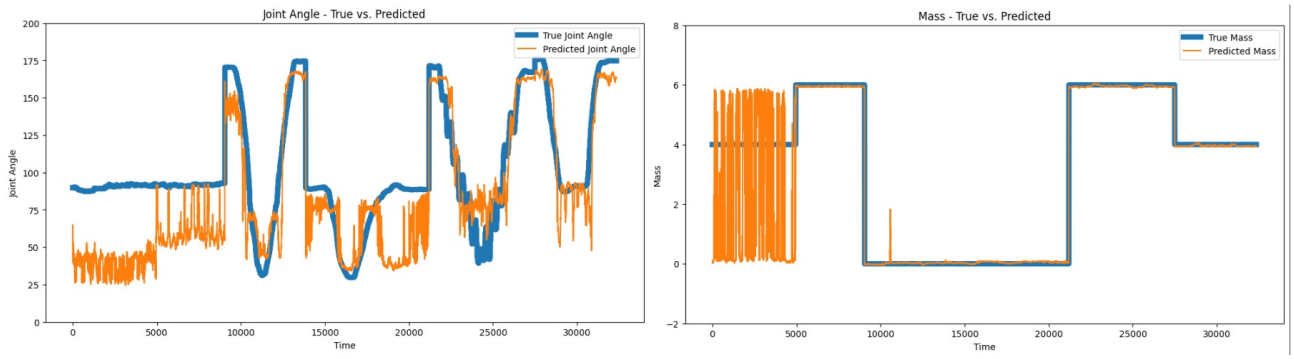


Figure 20: Second model with an SL of 30, L2 regularization (0.01), and 40% dropout on second layer.

The improvements resulted in a reduction in training accuracy to 90% and a rise in loss to 0.096, however on the test set, accuracy increased to 87% and loss climbed to 0.297. Figure 20, on the other hand, displays a superior match on the mass test data. The angle prediction also appears to suit the curvatures of the ground truth more correctly. The gap between training and test accuracy has narrowed, as predicted, but overfitting appears to be an issue. It was decided at this stage that additional progress would need hyperparameter adjustment.

3.4.4 Hyperparameter tuning

When done manually, hyperparameter tweaking may be a time-consuming and difficult task. Fortunately, there are methods that can assist in automating this approach. 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 what distribution it should use to pick the values for each run. While grid search is finite, meaning it ends when all combinations are exhausted, Random and Bayesian can continue indefinitely. [27]

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 takes 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 don't 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 more precision, we don't want it to come at the expense of loss, because there is still some overfitting going on. That suggests we've 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.

In order to decide which hyperparameters to choose for optimization

3.4.5 Calibrating for new subjects

We retrieved the mean and standard deviation of each feature from the data set for normalization and used them to fit new data to the model prediction. Because impedance data is predicted to vary greatly between individuals, the mean and standard deviation of each characteristic are also expected to fluctuate for new subjects. Calibrations must be conducted for each individual in order to achieve proper scale. This is accomplished by collecting static and dynamic data for the topic in issue with varying masses. The mean and standard deviation of each feature are then extracted from these samples, and the difference between these and the values from the training data is calculated. The difference will then be deducted from the mean and standard deviation of the training data, which should result in proper scaling.

4 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.

4.1 Future Work

This section explores what would be the next natural steps for MUSE if we had more time to work on it.

4.1.1 Exoskeleton 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 [28], 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 an assumption that the only force affecting the payload is gravity, we can use the formulae for force and torque to calculate an estimated maximum payload that can be lifted by the exoskeleton.

$$F = m \cdot a \tag{3}$$

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

$$\tau = F \cdot l \tag{4}$$

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

In our case, the formula becomes

$$\tau_{max} = F_{max} \cdot l_a = m_{max} \cdot g \cdot l_a \tag{5}$$

$$4.1\ [Nm] = m_{max} \cdot 9.82\ [N] \cdot 0.21\ [m] \tag{6}$$

Reducing and isolating m_max on the left-hand side

$$\frac{4.1\ [Nm]}{2.06\ [Nm]} = \frac{m_{max} \cdot 2.06\ [Nm]}{2.06\ [Nm]} \tag{7}$$

$$m_{max} = 1.99\ [kg] \tag{8}$$

From equations 5 through 8, we can see that the maximum payload that the exoskeleton can lift is roughly a third as much as the maximum weight that the GRU network was trained with, which was $6\ [kg]$. This is supported by the fact that the maximum torque of the exoskeleton is roughly a third of the torque needed to lift the aforementioned $6\ [kg]$ as seen in equations 9 through 11 below.

$$F_{6kg} = 6\ [kg] \cdot 9.82\ [N] = 58.92\ [N] \tag{9}$$

$$\tau_{6kg} = 58.92\ [N] \cdot 0.21\ [m] = 12.37\ [Nm] \tag{10}$$

$$\frac{4.1\ [Nm]}{12.37\ [Nm]} \approx 0.33 \tag{11}$$

So in conclusion to the above findings, the right way forward would be to install a motor with at least triple the torque rating to the one currently installed in the exoskeleton.

4.1.2 Database

Currently, the MUSE software stack uses a "database" made up of several .CSV files to ease data exchange between the stack's components, including the Eliko data interface, kinematic angle tracker, data visualizer, and exoskeleton control interface. It's clear that depending on this strategy isn't the most effective way to manage data interface across different applications.

The most apparent modification 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.

4.1.3 Internal Communication

There is currently no connection between the various applications in the software stack, and synchronous data collection between the Eliko data interface and kinematic angle tracker is accomplished via a keyboard macro. This, like the database selection, is a suboptimal solution. There are several solutions to this problem, with the two discussed in this section are TCP sockets and Robot Operating System (ROS).

4.1.3.1 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 bioelectrical impedance 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.

On the negative, 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 delay 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.

4.1.3.2 Robot Operating System

Another method is to write the various applications in the stack 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.

Moreover, ROS's flexibility for programming nodes in C++ or Python interchangeably inside the same stack 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.

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.

5 Conclusion

6 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] Ram Murat Singh and Shantanu Chatterji. "Trends and Challenges in EMG Based Control Scheme of Exoskeleton Robots- A Review". In: 2012. URL: https://api.semanticscholar.org/CorpusID:15171946.

[20] 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).

[21] 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.

[22] 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.

[23] 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.

[24] 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/.

[25] 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.

[26] 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.

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

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

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

Appendix

Appendix A — 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.

Physical Setup

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 ≈ 1.5 cm apart along the bicep, with the lowermost electrode also ≈ 1.5 cm from the elbow joint. The elbow joint should be aligned with the lines, as shown in Figure 21. It is important that the two middle sensors are as close to the middle of the bicep as possible.

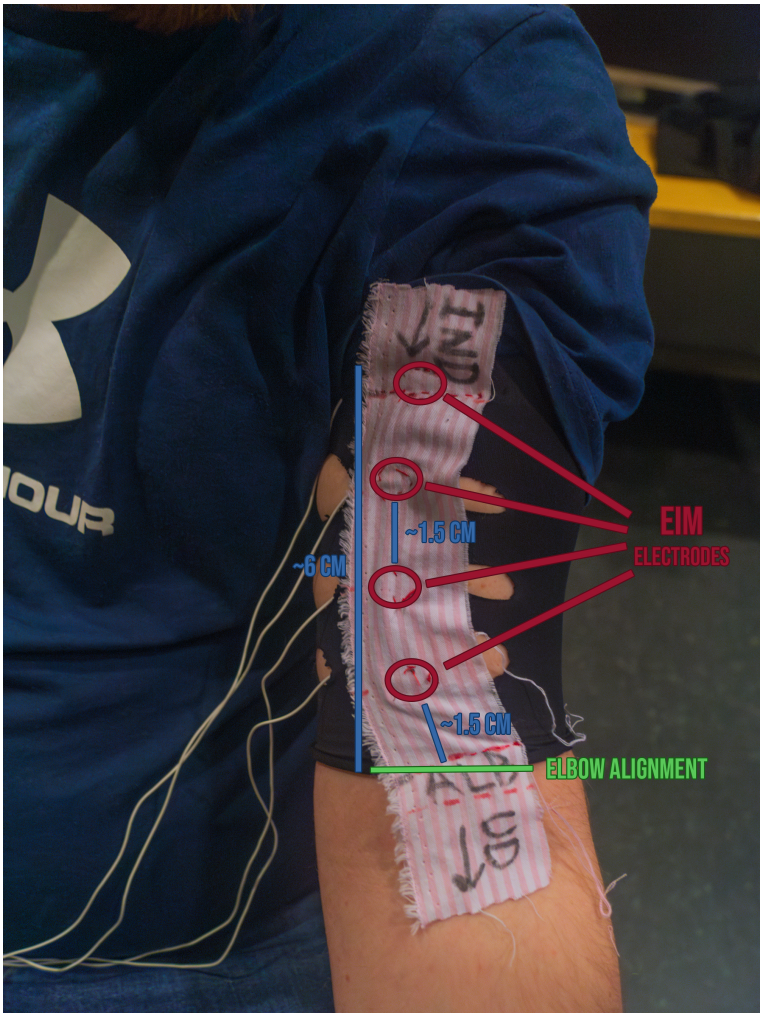


Figure 21: Visual guide for sensor sleeve construction

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:

$$\tau_{ff} = \tau_g \cdot a \tag{12}$$

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.

$$\tau_g = m \cdot g \cdot l \tag{13}$$

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.

$$\tau_{fb} = k\left(\theta_r - \theta\right) + B\left(\dot{\theta}_r - \dot{\theta}\right) \tag{14}$$

Where τ_{fb} is the feedback control signal, k and B are tuning variables that the control loop self-tunes, 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.

$$\dot{\theta}_r = \frac{\Delta\theta}{\Delta t} = \frac{\theta_r - \theta}{t_{i+1} - t_i} \tag{15}$$

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

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.

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 stack, please refer to the README in the MUSE GitHub repository [29], where it is elucidated in thorough detail.

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.

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 make sure to 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 make sure to 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, then fully bend your arm slowly and make sure to 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 make sure to 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 make sure to 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 dataset for training the GRU network, covering various weights, grips, and movements to enhance predictive accuracy from EIM data. We recommend creating a folder for each sample to contain the data.

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 21 for a visual guide.
2. Connect to the computer containing the software stack, run the **MUSE** Eliko data interface, and create a throwaway sample (Press **S**, **K**, then **Y** and **Enter**) so the startup sequence of the Eliko device doesn't cause a mismatch in time.
3. Start the **visualizer.py** program so you can visualize data and sanity check it.
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 of 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. See figure 22 for an example of how a full range of motion jitter would look in the visualizer.

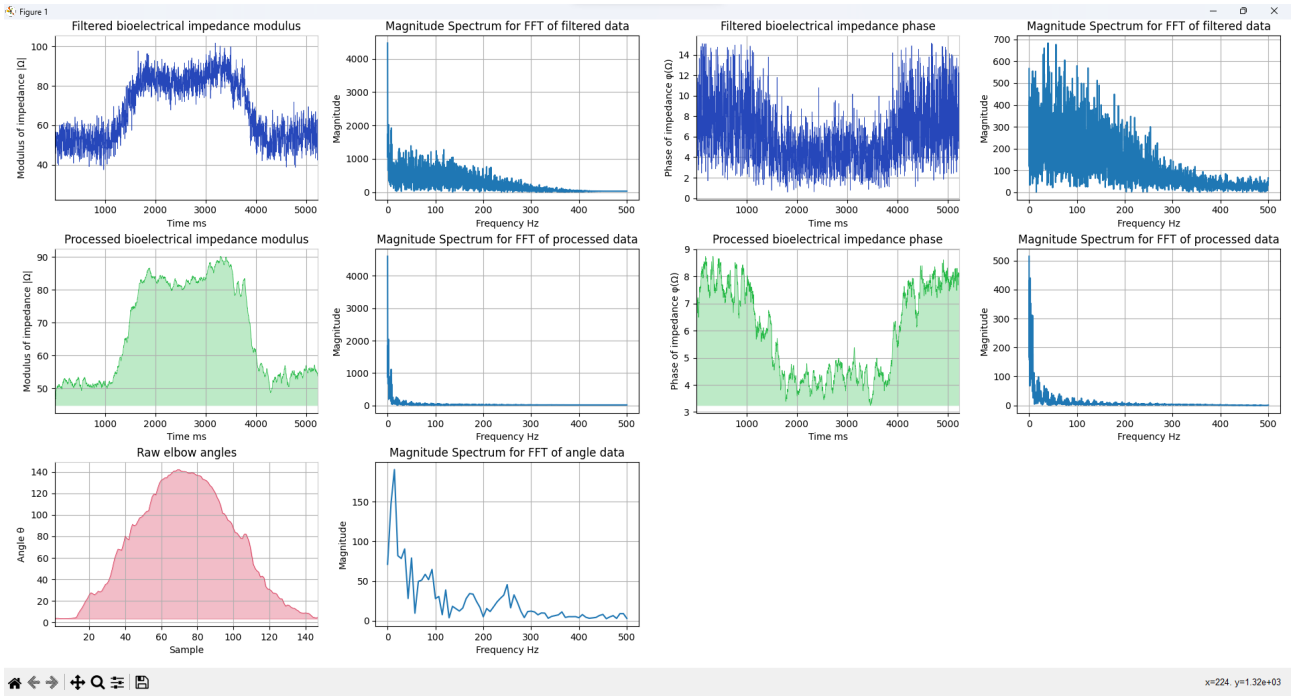


Figure 22: Full Range of Motion with Jitters Example.