Neuroprosthetic Device for Video Game Control

Worcester Polytechnic Institute, Worcester, MA 01609

A Major Qualifying Project submitted to the faculty of WORCESTER POLYTECHNIC INSTITUTE, in partial fulfillment of the requirements for the degree of Bachelor of Engineering

By Francis Coghlan and Drew Silvernail

Date: 2024

Report Submitted to: Professor Adam Lammert, Professor Taimoor Afzal

Worcester Polytechnic Institute

Table of Contents:

Table of Tables

Table of Figures:

Acknowledgments:

Our team would like to express our appreciation for the guidance and assistance from Professor Adam Lammert and Professor Taimoor Afzal of WPI. We also would like to express our gratitude to our families for supporting and listening to our thoughts and ideas and supporting us throughout writing this paper. This project was only as successful as it was with your support.

Abstract:

This research presents the design and development of a novel assistive device that seamlessly integrates surface electromyography (sEMG) and voice control modalities. The device targets users with limb amputations or limited mobility, empowering them to interact with video games intuitively. sEMG electrodes capture muscle activation signals, which are translated into game actions. Additionally, a microphone facilitates voice commands, providing supplementary control and adaptability. The design process emphasizes considerations for signal processing, adjustable algorithms for sEMG classification, and developing a user-friendly voice control interface. This research addresses a gap in the field of assistive technology by offering a multimodal control mechanism using real-time inputs that enhances accessibility and user experience within the gaming domain and beyond.

I. Project Proposal

1. Introduction:

Prosthetic research for amputation has primarily focused on the replacement of missing limbs with artificial substitutes. From basic functionality to sensory feedback, such prosthetics focus on controlling a physical device to interact with objects in the real world or to enable those with physical disabilities to operate devices. Typically, these prosthetics rely on electromyography (EMG) sensors for user input, allowing the ability to grab, release, hold, press, and so on. However, the question must be asked of whether such a method is the most efficient means of interaction or if it is possible to remove the physical prosthetic altogether for an operator to interact with an electronic device.

Accessibility research has made major improvements in the past twenty years. In the field of sEMG, currently, the few limitations left are recording deep muscle activity and the unintentional recordings of muscles other than the targeted muscle group. sEMG is typically used in clinical applications or for utilizing myoelectric prosthetics, but it also has the capability of controlling electronics through muscular impulses. Voice control has long been a sought-after tool for accessibility and has made improvements in recent years through the use of improved audio capture and analysis tools, as well as AI to analyze and process sound.

This project aims to design a multimodal system meant to allow user control within a video game, with the goal of being intuitive. The system should be compatible with multiple games, and be capable of connecting to different computer systems, and function with minimal user intervention. This paper begins with an overview of the latest publicly available research on prosthetic devices and interfaces for the disabled, as well as the various options for interaction with a prosthetic device, primarily focused on EMG research and voice control software over other control methods. It then details this project's approach to the proposed problem, and then

goes into the design methodology for creating a device meant to allow those with missing limbs to interact directly with computers. Due to time constraints, this project focuses primarily on the development of a device meant to activate the most commonly used keys of a keyboard for video gaming, to remove the need to create a device to control the entirety of a keyboard, though such an interface may be possible in the future.

1.1. Identified Problem:

Those with a physical disability, such as a hand amputation, who struggle to interact with electronics appropriately, typically will have to make uncomfortable adjustments or creative workarounds to control devices, to the point where they may not be capable of playing video games at the same functionality as other people.

1.2. Project Objective:

This project objective is to design and build an active control interface to allow someone to control a video game within a virtual sandbox environment.

1.3. Project Specifications:

The project requires the discovery of appropriate sEMG sensor locations on the human body and the development of a functional voice control system. Once locations are noted, the project needs an interface that actively samples sEMG and audio signals, interprets the signals, and sends appropriate signals to a computer. The signals will be processed into button presses, specifically ‘w’, ‘a’, ‘s’, ‘d’, and ‘space,’ which a user can use to move their avatar within the virtual environment. For more complex movements, such as aiming with the mouse, a Bluetooth mouse would be provided. This is done with the assumption that the user only has a single amputated hand. The goal is for a user to be capable of performing a complex task (e.g.,

completing a level of a game) with the device and mouse at similar or slightly slower paces than another using a mouse-keyboard interface.

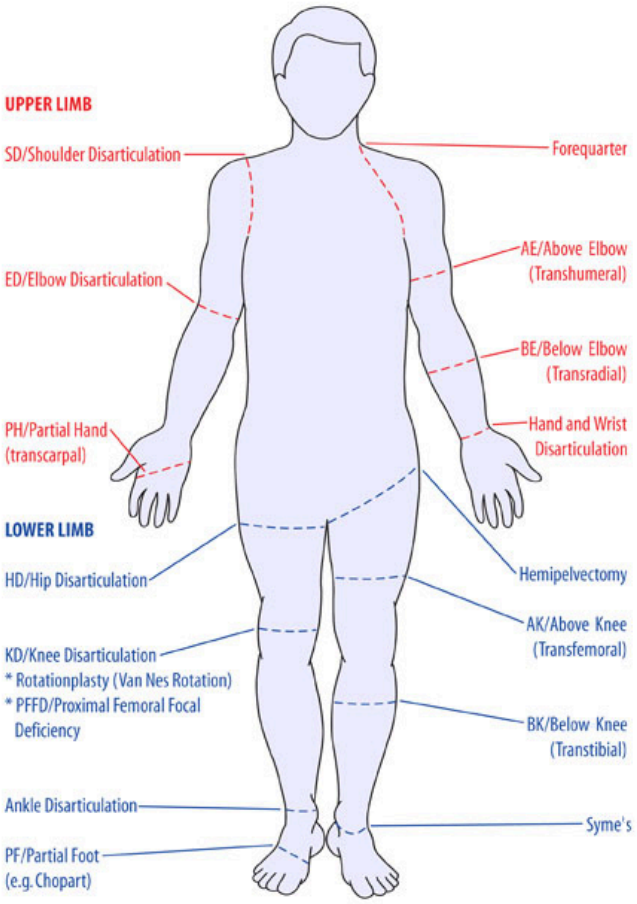
2. Background

2.1. The Problem.

Amputation is the variable removal of an extremity via medical intervention. Reasons for needing an amputation range from bodily trauma, infectious diseases, or even cancer. It was estimated in 2017 that 57.7 million people were living with limb amputation due to traumatic cases alone worldwide (McDonald et al., 2021). Alternatively, individuals may be born with congenital anomalies that lead to a missing or deformed hand or foot. Regardless of source, the site of the surgical amputation can change, though almost always, it would lead to at least the loss of a foot or a hand, depending on whether the upper or lower limb is amputated. Different patients, however, can require differing levels of amputation, and the more a limb is removed, the more the functionality of the limb diminishes, as shown in Figure 2.1.

In the case of upper limb amputation, limb functionality significantly decreases when the hand is amputated. This decreases an individual's autonomy in performing everyday tasks. As such, worldwide efforts have been made to develop prosthetic solutions. Solutions can range from practical or affordable prosthetic devices that solely focus on the mechanical replacement of the limb to prosthetics capable of interpreting and transmitting signals to and from the nervous system of the user through EMG, granting the capacity of both control and feedback sensations.

Figure 2.1: Amputation sites (Physiopedia, (n.d.)



For upper limb injuries, there is an express need for prosthetic devices capable of multiple operations. However, prosthetics primarily focus on having multifunctional uses for everyday life, which can lead to them falling short in providing fine motor skills or control to do specific tasks in a time-sensitive manner. For instance, interaction with electronic devices, such as smartphones or computers, may take those using a prosthetic longer to interact with. Specialized human interface devices (HIDs) have been made to assist those with amputation, such as one-handed keyboards, foot controllers, and facial interface devices, but each has their own advantages and disadvantages.

2.2. The Definition and Source of EMG

Electromyography (EMG) is a procedure meant to measure muscle response or electrical activity from myoelectrical signals. EMG sensors allow one to look into myoelectrical signals, measure muscular performance, and even activate devices based on the signals received from the sensors (Konrad, 2006). EMGs are standardly used in myoelectric prosthetics or interfaces, but are also typically used for clinical diagnosis of neurological or neuromuscular problems (Reaz et al., 2006).

A motor unit (MU) is the smallest functional unit for neural control of muscular contraction. It is the cell body and dendrites of a motor neuron, the multiple branches of the axon, and the muscle fibers that innervate it. All muscle fibers of a given unit act as one and are innervated by a single neuron. Excitability of muscle fibers through neural control is a major part of muscle physiology (Konrad, 2006; Etana et al., 2023).

Much like nerves, the excitability of muscles is explained by the difference in membrane electrical potential. The membrane is semi-permeable, with a differing concentration of ions between the inside and outside of the muscle cell producing a difference in potential. At resting potential, where the muscle is inactive, the voltage is approximately between -80 mV to -90 mV. When the muscle cell is activated by the nervous system, membrane channels open up to allow Na+ ions to diffuse across the membrane. Known as depolarization, this causes the electrical potential of the cell to change from the resting potential of -80 mV to +30 mV. This voltage change is almost immediately reversed from a further exchange of ions with an active ion pump mechanism. This restoration of voltage potential across the cell is known as repolarization. However, this reversed voltage change typically overshoots below -80 mV for a short period of time, before being restored back to resting potential. This overshoot is known as hyperpolarization. These rapid changes of voltages across the cell membrane are known as

action potentials (Konrad, 2006). The EMG signal is derived from these action potentials traveling along the surface of muscle fibers (Konrad, 2006). Action potentials propagate along muscle fibers in both directions from their origin, terminating at the tendon zone present at the end of each fiber (Etana et al., 2023).

The generating of potential differences originating from each MU are defined as the motor units action potentials (MUAPs) of active motor units (MUs), which are made up of thousands of action potentials (AP) generated by individual fibers of MUs in the muscle (Merletti & Muceli, 2019). The motor unit action potentials of all active motor units are superposed and detected as a single bipolar EMG signal with symmetric distribution of positive and negative amplitudes (Konrad, 2006). The EMG signal is effectively the sum of MUAPs originating from different MUs (Etana et al., 2023). The two most important mechanisms influencing the magnitude and density of the observed signal are the recruitment of MUAPs, where multiple MUAPs activate in response to a single reflex or induced by the central nervous system, and their firing frequency, the amount of times MUAPs fire for a muscular contraction. These mechanisms are what modulate the force output and response of the involved muscle for different actions, or differing requirements of force (Konrad, 2006).

2.3. RAW EMG signal

A standard EMG signal is an unfiltered and unprocessed signal detecting superposed MUAP electrical potentials. When a muscle group is relaxed, a mostly noise-free EMG baseline can be seen, though the baseline noise can depend on different factors, such as the quality of the EMG amplifier, environmental noise, and quality of preparation. A relaxed muscle has no significant EMG activity due to lack of depolarization and action potentials. But raw EMG activity, such as when a muscle is tensing, is typically in a random shape, where one raw EMG recording of muscle activation cannot be precisely reproduced in the same shape as a second

activation of the same muscle group. The actual set of motor units being recruited for motions are constantly changing within the group of available motor units with each flex, meaning that sometimes, a differing number of motor units in differing positions will activate (Konrad, 2006).

Surface EMG (sEMG) is a two-dimensional distribution of electrical potential over the skin. (Merletti & Muceli, 2019). Raw sEMG signals can range between positive and negative 5000 microvolts, and typically the frequency of these signals ranges between 6 and 500 Hz, though the signal shows most frequency power between ~20 and 150 Hz (Konrad, 2006).

EMG signals can be influenced from their raw state by several factors, such as tissue characteristics, cross-talk produced by neighboring muscles around the targeted muscle group, physical changes between signal origin and detection site (such as sensor movement), external noise from electrical environments, and the selection of electrodes and amplifiers themselves. Most of these factors can be minimized or controlled by proper preparations and checking laboratory conditions (see B.7) (Konrad, 2006).

In regards to tissue characteristics, the human connective tissue and skin layers have a low pass effect on the raw EMG signal, where human tissue blocks electrical signals of higher frequencies and only allows signals of lower frequencies to pass through (Merletti & Muceli, 2019). As such, any surface EMG sensors will not be able to measure the original firing and amplitude characteristics. But the sEMG signal does reflect the recruitment and firing processes of the motor units within a measured muscle (Konrad, 2006).

2.4. Refinement of EMG signal

EMG electrodes are typically configured in a bipolar configuration, with two different nodes for an electrode pair, and one for the common ground reference electrode. Typically for the gathering of EMG signals, skin surface electrodes are used due to their non-invasiveness,

as they can simply be placed and secured on the skin to read raw EMG data, though more invasive probes such as fine wire or needle electrodes can be used to read EMG signals in deeper muscle layers. This paper will focus on sEMG signals to minimize wearer invasiveness. Disposable surface electrodes are most commonly silver/silver chloride pre-gelled electrodes, and are recommended for general use in sEMG experiments (Konrad, 2006).

The quality of the EMG can improve based on skin preparation prior to the placement of electrodes. Removing as many obstacles between the sensor and the source of the EMG signal, the muscles, can help make the signal clearer to read and process. The main goals for skin preparation are stable electrode contract, and low skin impedance.

Impedance is the effective opposition of an electrical circuit to alternating current, which rises from the effects of resistance and reactance in the circuit. Resistance is the measure of opposition to the flow of electric current, while reactance is opposition presented to alternating current by inductance and capacitance. Reactance stores energy and shifts it by a quarter of a cycle relative to phase of the voltage across the reactance element, before returning the energy to the circuit (Urone, 2012).

Skin impedance is the resistive response of a specific skin region to externally applied electrical current. Impedance depends on various factors, like structural thickness and moisture content, presence of sweat glands, age of individuals, geographical location, local temperature and humidity. Hair follicles and sweat glands exhibit resistive properties in the skin, while the lipid bilayer exhibits capacitive properties. Thus, the heterogenous layered nature of the human skin makes it difficult to determine the electrical behavior of skin with physiological tissue conditions and develop accurate models without flaws (Bora & Dasgupta, 2020) (Murphy, 2021).

When EMG is recorded using surface electrodes, the amplitude of the signal is attenuated by the connective tissues, skin layers, and subcutaneous adipose layer that is

present between the source of MUAPs and the skin surface (Etana et al., 2023). Thus, most sEMG recordings require amplifiers to record the weak electrical signals.

Most EMG amplifiers are designed for skin impedance levels between 5 and 50 kOhm, primarily when used with a bipolar electrode setup. There are no general rules for skin preparation, but different methods exist, such as removing hair between electrodes and skin, and cleaning the skin of dead skin cells, which produce high impedance. In addition, the targeted test exercise and conditions are important: if the measurement of EMG signals is during slow or small movements, simple alcohol cleaning may be sufficient, and if measurements are made in dynamic conditions, such as running or accelerated movements, thorough preparation is needed (Konrad, 2006).

2.5. EMG surface electrodes and electrode placement

There exist three general types of surface electrodes: wet (conductive paste or gel), dry (metal-skin contact with no gel or paste), and insulating (capacitive electrodes) with no electric contact with skin. The most common sEMG sensors are Ag-AgCl pediatric ECG electrodes, though some industries have developed reusable, more flexible electrodes for sEMG measurements (Merletti & Muceli, 2019).

The number of electrodes used for each muscle group may vary from two to hundreds, depending on the task being required. And ultimately, the placement of those sensors can vary between people, even if they are meant to target the same muscle group (Merletti & Muceli, 2019). Different studies have come up with different means of determining the best locations for sEMG sensors, primarily based on both the type of sensor they utilized and the task the researchers are attempting to do. One paper, for instance, has a ring of sensors located around the upper forearm, running signals into a computer algorithm to recognize when the user is

moving their hand into specific gestures, such as a fist, hand open, wrist flexion, wrist extension, and so on (Oña ED, 2022).

Regardless, every individual will have noticeable differences in sEMG signal quality due to biological and environmental differences. If the sEMG sensors are placed in exactly the same location on their body, they will also have noticeable differences in sEMG signal quality, such as amplitude. As such, the user has to determine what location would be best suited for their sensors prior to measuring EMG signals, typically through trial and error (Bora, 2020). That does not mean, however, that there are not standard general locations for specific muscle groups where sEMG quality is, on average, consistent. These muscle groups can be seen in Figure 2.2 and Figure 2.3 below.

Figure 2.2: Frontal view of typical electrode sites. Left side indicates deep muscles and positions for fine wire electrodes, while the right side is for surface muscles and electrodes. (Konrad, 2006)

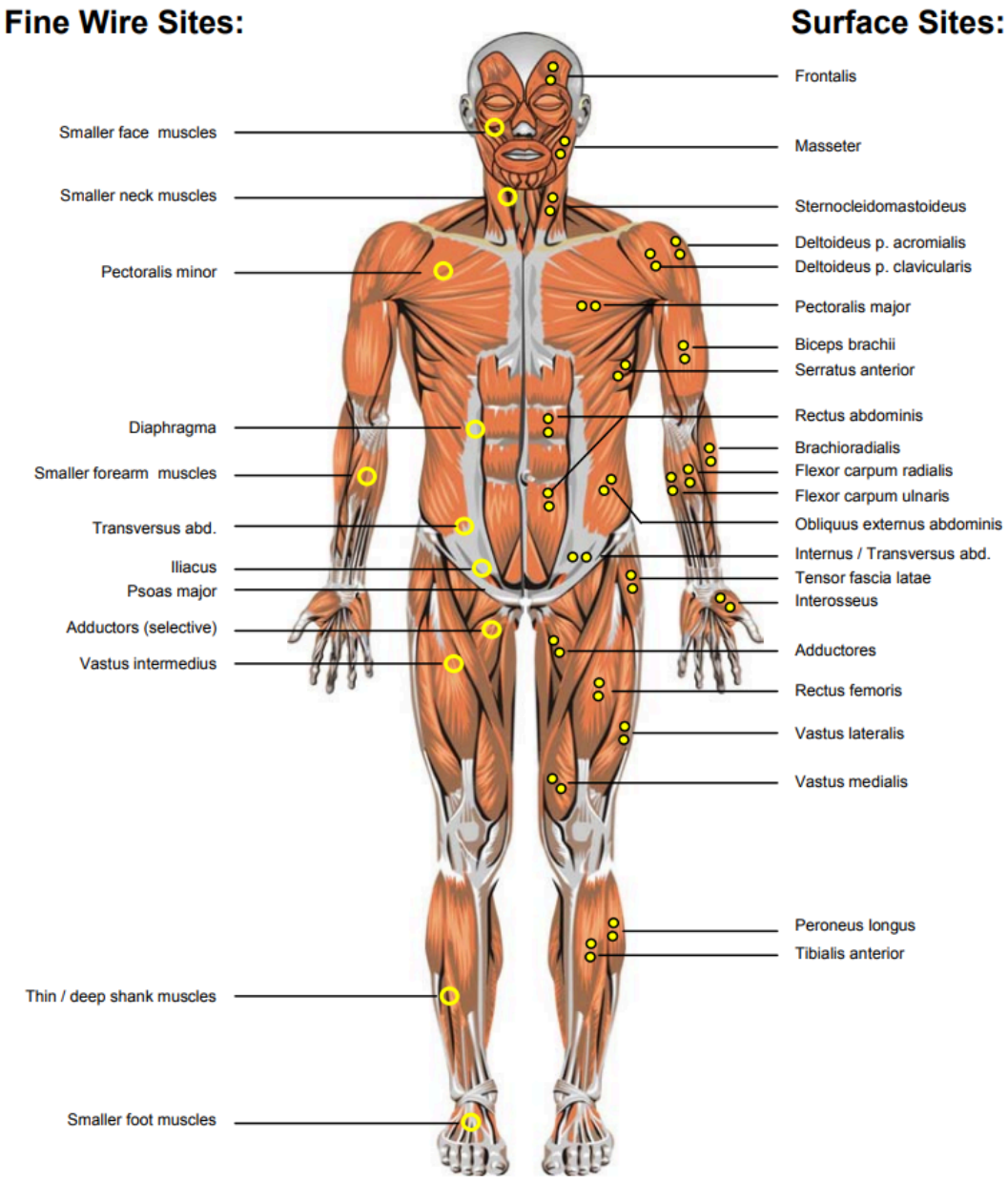
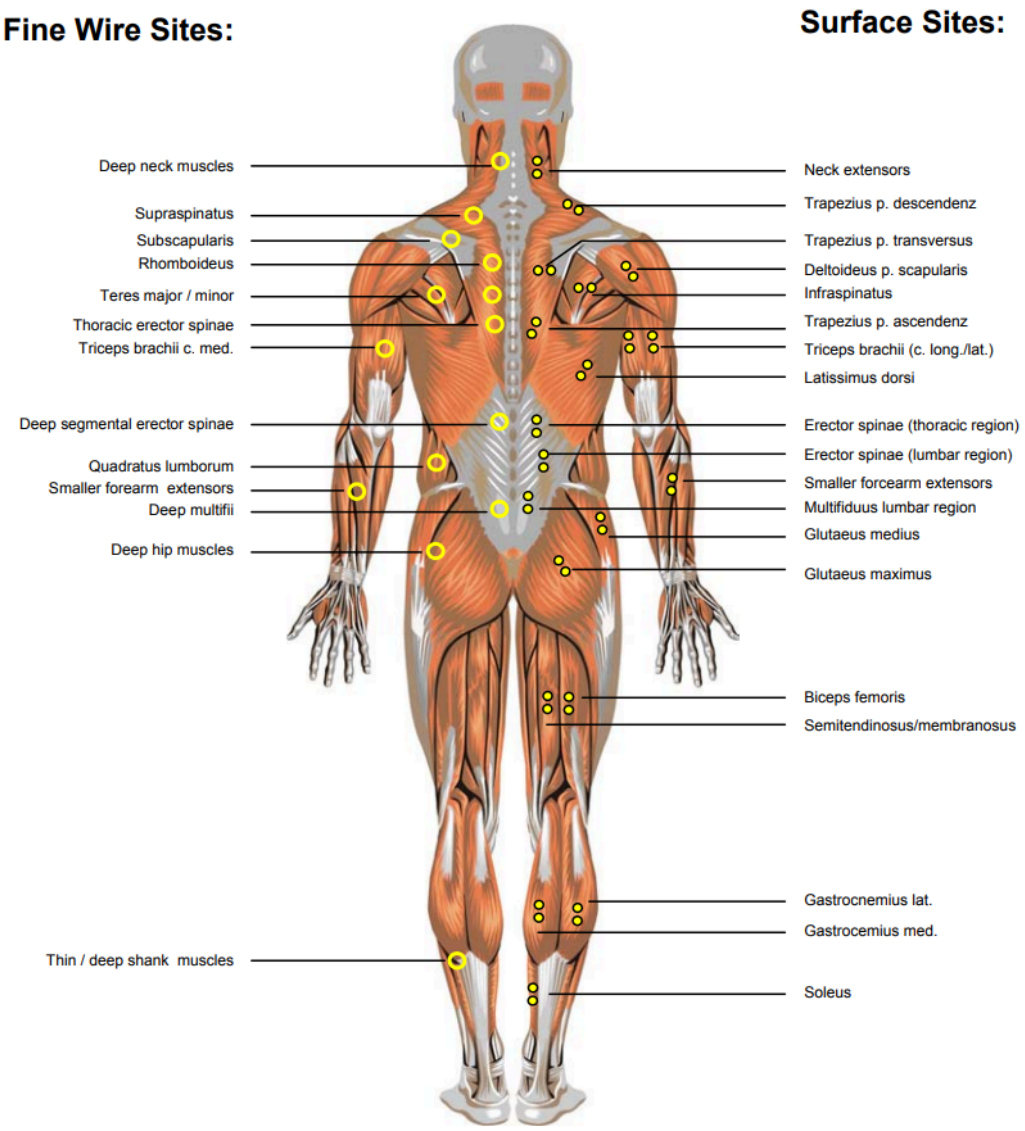


Figure 2.3: Dorsal View of typical electrode sites. Left side indicates deep muscles and positions for fine wire electrodes, while the right side is for surface muscles and electrodes. (Konrad, 2006)



2.6. EMG Amplifiers, Bandpass Filter and Sampling Frequency

EMG-amplifiers are differential amplifiers, meant to reject or eliminate signal artifacts. Differential amplification detects potential differences between electrodes, and cancels interference out. For instance, background noise from a nearby electrical device may reach sEMG electrodes. But noise signals typically reach both electrodes with no phase shift, and are equal in phase and amplitude. These types of signals are common mode signals, and the electrodes will have a gain based on these signals (Konrad, 2006).

The Common Mode Rejection Ratio (CMRR) is the term defining the relationship between differential and common mode gain. It effectively allows differential signals, such as the signals originating from raw EMG data from muscles, while filtering out noise signals, such as a signal coming from a nearby electrical device. CMRR should be as high as possible to eliminate the most interfering signals to improve quality (Konrad, 2006).

A raw EMG signal that has been read from sEMG electrodes has typical charges between a few microvolts to 2-3 millivolts. With amplifiers, the signal is multiplied by a factor of 500 to 1000. The input impedance of the amplifier should have a value of at least 10x of the given impedance of the electrode (Konrad, 2006).

Filtering an EMG signal can be done through multiple means, at once, and commonly, analog filters and digital filters are used to refine an EMG signal. In addition, algorithms can be utilized to also refine the signal along with the analog front-end. An analog filter uses analog electronic circuits made up from components such as resistors, capacitors, and operational amplifiers to produce a desired filtering effect. Typically, a high-pass filter, used to remove low frequency signals from a signal and allows high frequency signals to pass through, and a low-pass filter, used to remove high frequency signals and allows low frequency signals to pass

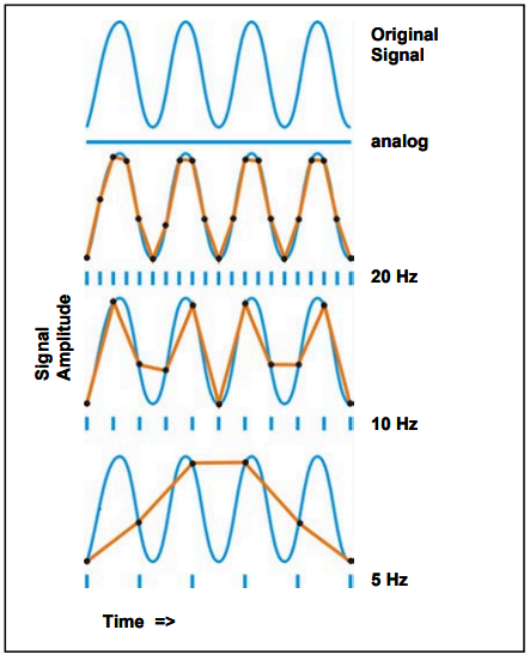
through, are used to create a band pass filter, to filter out interfering signals outside of the range of EMG signals to allow only the EMG signal through.

The frequency range of an EMG amplifier, with a bandpass filter, should start from 10 Hz highpass and go up to 500 Hz lowpass. Notch filtering, typically used to cancel power hum, destroys too much EMG signal information, and needs to be avoided (Konrad, 2006).

Digital filters, on the other hand, use a digital processor to perform calculations and mathematical operations on sample values of the signal to refine it. For digital filters, EMG signals have to be converted from an analog voltage to a digital signal, through A/D conversion. The resolution of A/D measurement boards has to be capable of converting an amplitude range of +/- 5 volts. A 12 bit A/D board can separate the voltage range of the input signal into 4095 intervals, which should be sufficient for most setups. However, very small signals may need higher amplification to achieve a better amplitude resolution (Konrad, 2006).

The determination of sampling frequency is necessary to properly transmit an EMG signal. The sampling rate of the A/D board must be twice as high as the maximum expected frequency, as determined by the Sampling Theorem of Nyquist. Sampling a signal at a frequency too low results in aliasing effects, which can result in significant loss of signal information (Konrad, 2006). This effect can be seen in Figure 2.4.

Figure 2.4: Effects of A/D sampling frequency on a digital signal. Too low frequencies result in loss of signal information (Konrad, 2006).



The frequency of EMG signals is located between 10 and 250 Hz (Konrad, 2006). An amplifier band setting of 10 to 500 Hz is recommended for EMG signals. This results in a sampling frequency of at least 1000 Hz or even 1500 Hz to prevent any loss of signal.

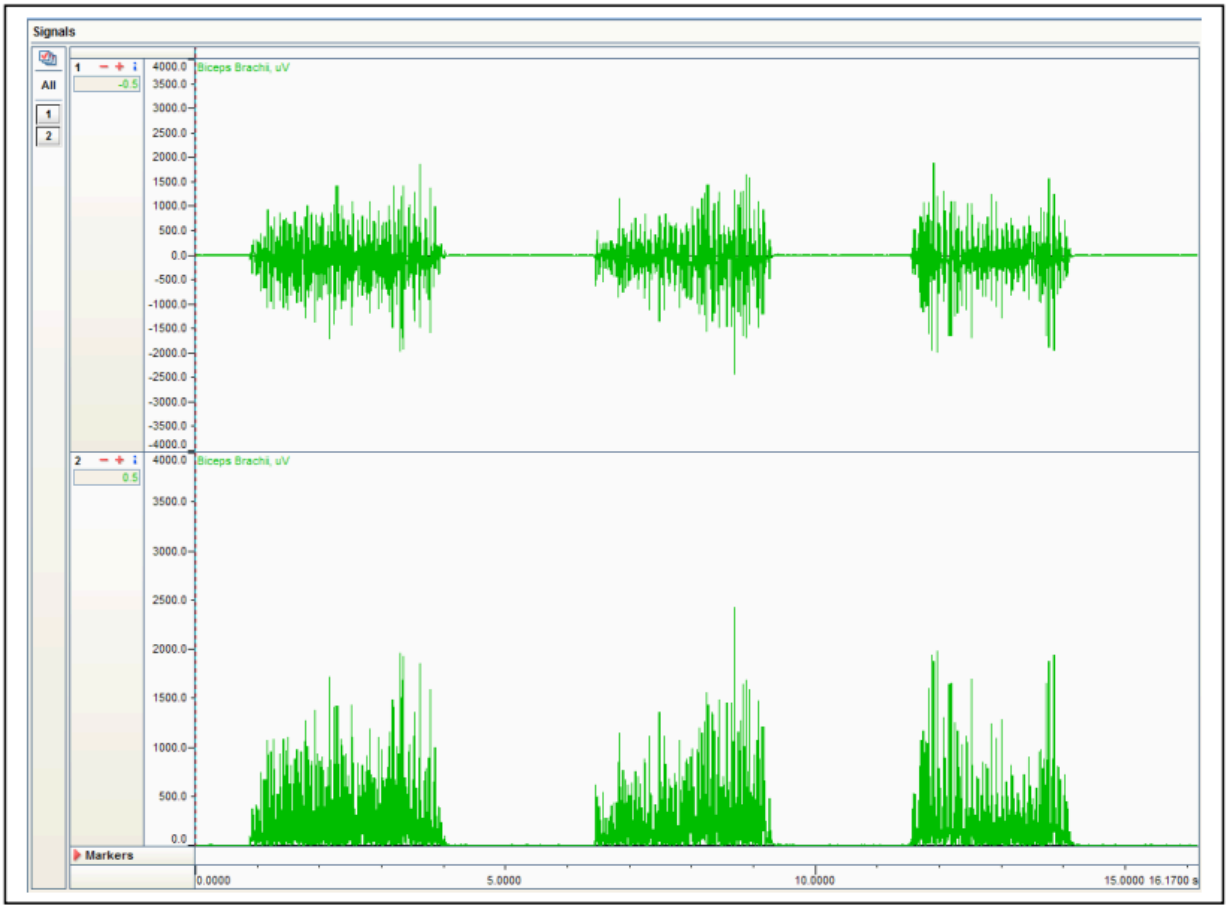
2.7. EMG Signal Processing - Rectification:

The original EMG recording already contains important information and can serve as appropriate documentation of muscle innervation. But amplitude analysis techniques can be used to increase reliability of readings.

A first step for EMG processing is a full wave rectification, whose effects can be seen in Figure 2.5. Negative spikes are moved up to positive, or are reflected by the baseline. This allows for standard amplitude parameters like mean, peak/max value, area to apply to the

curve. Otherwise, the raw EMG signal has a mean value of zero, and standard processing techniques cannot be applied properly.

Figure 2.5: Comparison of raw EMG recording to Full Wave rectified EMG recording (Konrad, 2006).



The interference pattern of EMG is inherently random. Different motor units may be recruited to perform the same motor action, or MUAPs may superimpose on each other in differing ways. Effectively, a raw EMG signal indicating muscle activity cannot be reproduced. However, it is possible to apply digital smoothing algorithms to minimize the non-reproducible part of the signal, allowing for more reproducible signals that can be used for the activation of electrical devices such as prosthetics. Two smoothing algorithms are available for this process.

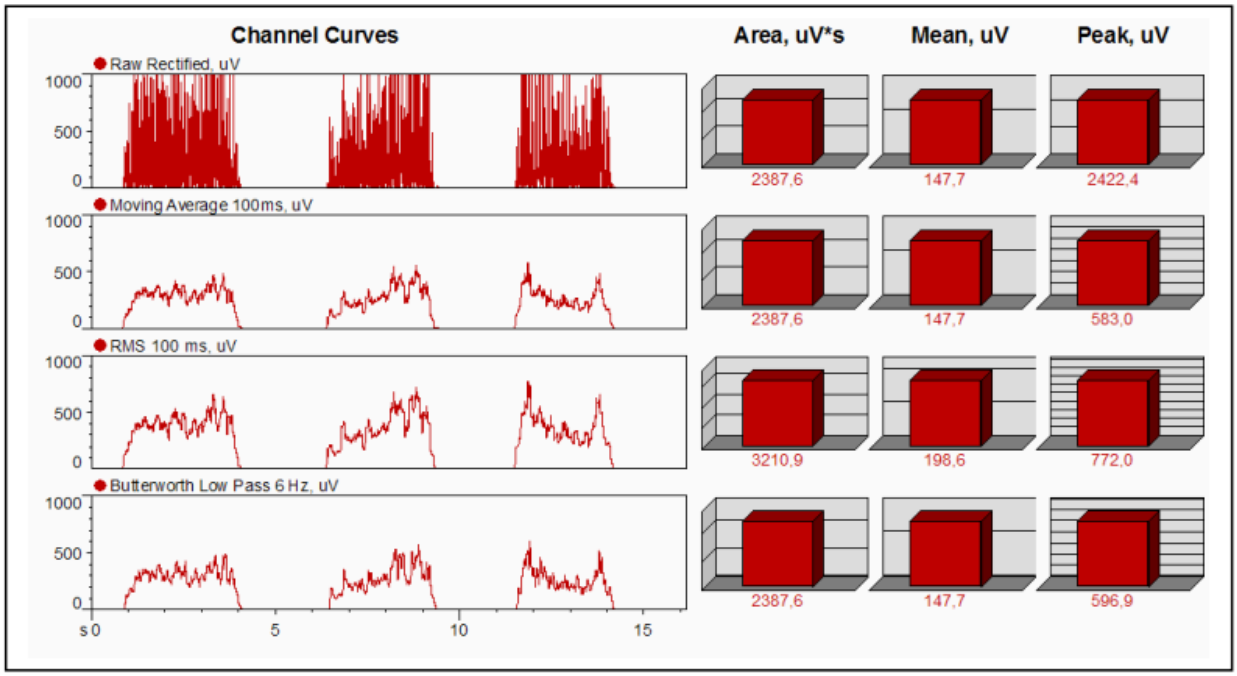
Moving average (MovAg) is an averaging method where, based on a predefined time window, a certain amount of data is averaged using a sliding window technique. Sections of the EMG signal are taken based on the time window, where it is then averaged.

Root Mean Square (RMS) utilizes the square root calculation to smooth the signal. RMS can result in higher EMG amplitude data than MovAg, and is the preferred algorithm for smoothing.

Both algorithms have a defined time sampling window. Typically, time durations of 20 ms are appropriate for fast movements, while time durations of 500 ms are suited for slow or static activities instead. The higher the time window, the higher the risk of a phase shift and signal loss (Urone, 2012). Typically, a sampling window between 50 to 100 ms is sufficient in most conditions.

An alternate method to the above algorithms are filters such as low pass filters. A low pass filter at 6 Hz can be used to create a linear envelope EMG instead of digital algorithms, where a linear envelope is the combination of the low pass filter and a raw EMG signal full wave rectification. The signal is converted from a spiky wave with fast oscillations into a smoother, more linear line (Esposito, 2023). Examples of linear envelope EMGs resulting from the algorithms or filters can be seen in Figure 2.6.

Figure 2.6: Comparison of RAW EMG to linear envelope EMGs MovAg, RMS, and Butterworth Low Pass (Konrad 2006)



2.8. EMG control of devices

sEMG sensors are a viable source of input for designing a human-machine interface (HMI). When traditional means of HMIs are unsuitable for controlling devices, such as keyboards, or button presses, sEMGs can be used instead. Typically, this will be done by tensing muscles being read from sEMG sensors, and the increased detected amplitude of sEMG signals would trigger desirable outputs on devices from any detected activity.

As sEMG sensors are typically utilized when traditional means of HMIs are unviable, research has shown that there typically rises three different types of devices sEMG sensors would control: 1) A medical rehabilitation device to restore functionality to its user, 2) a replacement prosthetic limb to replace a missing limb, 3) an external device meant to act as an alternate means of interaction.

A recent work titled ‘Processing Surface EMG Signals for Exoskeleton Motion Control’ shows an example of a medical rehabilitation device would be an EMG-driven lower limb exoskeleton, meant to assist in physical rehabilitation for wearers (Yin et al., 2020). It utilizes sEMG sensors to calculate and derive gait cycle durations of various walking speeds. Both sEMG signals and calculated gait cycle durations were utilized to control the motion speed of the exoskeleton. This kind of technology shows promising signs of enhancing motor recovery in patients.

There are a large variety of EMG-controlled prosthetics being both developed and sold. Replacement prosthetic limbs are diverse in their costs, complexity, and capabilities (Smidt, 2023). For a recent example in the development of sEMG prosthetics, a work titled “A Low-Cost EMG-Controlled Anthropomorphic Robotic Hand for Power and Precision Grasp” focuses on the design of a low-cost affordable prosthetic hand, utilizing commercial EMG armband for its sensors, and utilizing a modified open-source six-degree-of-freedom hand prosthetic (Sánchez-Velasco et al., 2020). The processing of EMG data was replaced by a portable hardware system that was built into the design of the prosthetic limb and EMG armband.

More rare are devices designed to utilize sEMGs for the main method of communication in HMIs. An example of such a device would be an electric wheelchair with an attached robotic arm for grasping and manipulation purposes, meant to primarily help those suffering from severely reduced muscle function (Vogel et al., 2020). Another example is a system that utilizes EMG and EEG (Electroencephalography) signals to control a smart home for individuals with paralysis (Chai et al., 2020).

Some EMG-based HIDs (Human Interface Devices) are more focused on the enhancement of the user’s capacity to interact with devices. For instance, the paper “Typing Everywhere with an EMG Keyboard: A Novel Myo Armband-Based HCI Tool” documents the

utilization of a Myo armband to utilize hand gestures, read through sEMG signals, to type with a nine-key keyboard layout similar to keypads in mobile phones (Fu et al., 2020). However, the Myo armband technology relies on the capacity to make hand gestures, and for those with a hand amputation, renders it more difficult with which to interact.

2.9. Limitations of sEMG devices:

sEMG has several limitations that prevent more active use. For instance, in an active workplace environment, such as in industry, or athletics, electrodes have a tendency to be displaced from the body of the user, requiring the need for either repositioning or replacement of the electrode (Olmo, 2020). In addition, sEMG measurements have a variety of limitations, ranging from the inability to measure deep muscle activity, to active interference from various sources. Many of these limitations and methods to limit them have been detailed in earlier sections.

Most of the issues can be resolved with further investments, but the technology can become more complex, less modifiable, as investment price increases. For instance, it is possible to design an sEMG armband with multiple sensors wrapped around the arm, paired with neural network training to read the movement of the arm and recognize intentional gesture, based on monitored sEMG signals, but both price and time required to train the machine for signal processing increases as a result (Zhang, 2023). In addition, these devices can become too specialized, and too specific for user to user. With the prior example, the armband can steadily read muscular impulses from a group of subjects, around 30, once the neural network is trained, but the paper does not describe how well the device would generalize to other subjects that did not provide data for neural network training.

2.10. History of Voice Control

Various innovations such as voice recognition software, eye-tracking systems, and adaptive controllers have enabled many individuals to engage in digital experiences previously inaccessible to them. Despite these advancements, a critical gap remains in providing real-time control solutions for individuals with severe physical disabilities. While existing technologies offer alternative input methods, they often lack the responsiveness required for immersive gaming experiences. A voice control software, in combination with the previously described sEMG system, should provide ample dimensions for control.

2.11. Voice Control in Accessibility

Voice control revolutionizes accessibility by providing alternative ways to interact with technology, especially for individuals with motor impairments, vision loss, or cognitive difficulties. People who struggle with traditional input methods, like keyboards or touch screens, can use their voice to dictate text messages, compose emails, search the web, and control various devices. For example, a person with a spinal cord injury or conditions like ALS might struggle to use a traditional keyboard or mouse. Voice control lets them navigate websites, open applications, and even compose emails – all with spoken commands. Or someone with dyslexia can use voice-to-text software that allows those who struggle with typing to dictate reports, emails, or even creative works effortlessly.

This enhanced independence fosters greater inclusion in the digital world. However, there's a trade-off between speed and accuracy. Simpler signals, like indistinct noises or vocalizations, are processed faster as the software analyzes a less complex signal. In contrast, high-specific commands, such as dictating a long paragraph of text or requesting detailed information, require more time for the software to understand the nuances of speech, including individual words, intonation, and context.

Limitations of Voice Control

Currently, there are two different methods for using one's voice to control secondary devices. The first and most well-known is Speech Recognition. This type of recognition uses sophisticated algorithms to determine what word was spoken, and then internal logic determines the following course of action. This design excels at specificity and control but lacks speed, and incorrect recognition can lead to other annoying issues like opening the wrong file or typing the wrong word, requiring user intervention to remedy. The other common form of voice control and the method we chose to explore for this project was audio analysis. As opposed to speech recognition, audio analysis extracts mathematical data from the audio signal in real-time, sending that data to the program to use as input signals. This technique does not have as many dimensions of control as speech recognition but excels at speed and accuracy.

2.12 Multimodal Systems

Multimodal systems hold immense promise within the realm of accessibility. By allowing users to interact with technology through a blend of input methods, they cater to varied abilities and preferences. Consider a user with limited hand mobility; they might navigate a menu using eye-tracking technology, paired with voice commands to execute specific actions. Or, someone with a hearing impairment could rely on haptic feedback coupled with visual cues on their smart device for notifications and alerts. These combinations create flexibility and adaptability in ways that single-mode systems often cannot.

Our research aims to push this multimodal vision further by exploring the potential of sEMG alongside traditional audio analysis. By combining modalities in this way, we envision a system where a user could perform precise, localized muscle movements in combination with voice control, enabling a level of control currently unattainable.

2.13. Video Games

Video games are often implemented into electromyography applications for muscular training, typically by focusing on repetitive, singular movements to control a simple game. For instance, in “Surface Electromyography-Driven Therapeutic Gaming for Rehabilitation of Upper Extremity Weakness: A Pilot Study,” focuses on using a video game similar to ‘Flappy Bird’, where a single muscular impulse is enough to control the game (Liu et al., 2022). This was done to create an enjoyable form of physical therapy, where patients would be motivated and engaged with repeated methods to retrain muscles with severe extremity weakness. However, most of the time, these electromyography applications are designed with a single video game in mind, or the video game itself is designed solely for the therapeutic application. sEMGs have been utilized in other applications as well, such as interaction with a computer, but again suffer from limitations, such as being only designed for mouse interaction. However, it should be possible to design an sEMG interface capable of interactions with multiple, different games, each with differing levels of complexity.

For instance, it should be possible to design a sEMG interface to control one of the most popular games worldwide, Minecraft. With over 300 million sales as of October 2023, the game should serve as an appropriate test for a game controller sEMG device (Zachary, 2023).

3. Project Approach

The need for a device like this is based around exploratory research, where it is apparent that in recent years there has not been a casual use of multimodal systems. To elaborate, most research focuses on the creation of interface devices meant for fundamental purposes, such as prosthetics, or equipment interfaces, such as interfacing with a wheelchair for movement, or interfacing with a smart home to ease interactivity. However, most research focuses on granting the ability to meet fundamental needs, or multiple needs at once. There hasn't been research shown, at least recently, on equipment meant to assist in interactivity with electrical equipment on a more casual use, such as video games or interacting with a keyboard.

The project's main objective is to design a control system to connect for a PC computer to emulate the "wasd" keys and spacebar at the hardware level, and combine multiple EMG inputs with voice signals to allow for more outputs. Note that the "wasd" keys are standard for use in video gaming, typically for moving a player's in-game avatar within a virtual environment. The target audience for this device are those with a single amputated hand.

The project assumes that the intended user for this system will be an amputee, but can still use a mouse with the other hand. The mouse will have additional buttons on it to add more interactivity in comparison to the limited amount of outputs the device will be able to create. Theoretically, anyone else can use this system in place of a keyboard. In addition, this project focuses solely on gaming as a testing medium, instead of typing on a full keyboard. As such, the project will assume the need to set up the game, possibly through keyboard interaction if necessary for tasks such as logging in, typing in a password, etc.

3.1 Project Specifications:

Table 3.1: Design Specifications of sEMG Interface

Design Specifications	
sEMG Signal Detection	Sampling Frequency of 1000 Hz to 1500 Hz Read signals through three conductive electrodes per sensor placed on the skin, and are functionally available at home.
Isolation	Minimal risk of electric shock when the system is connected to a device connected to the power grid.
sEMG Signal Filtering and Processing	Capable of applying rectifying and enveloping signal processing techniques
Transmission of Data	Capable of reading sEMG signals and outputting appropriate keystrokes within 500 ms
Logic	Able to read 4 sEMG sources and process to generate 6 possible outcomes.
Keyboard Controller	Functionally act as an external HID device capable of interacting with Windows OS systems through the emulation of keyboard strokes.
Size/Weight/Portability	Lightweight to fit on a person, with minimal interference to the person's ability for interaction.
Additional Control	External wireless interface devices instead of wired devices, such as a mouse.
Microphone	Microphone to capture audio and send to computer for analysis
Digital Audio Interface (optional)	Some microphones may require additional power or special inputs which can be satisfied through the use of a digital audio interface
Audio Analysis	Python Scripts are utilized in this project for the analysis of audio signals captured by the microphone

Table 3.1 shows the required specifications of the interface for the project to be evaluated as successful. On top of having a functional interface, there were additional goals for

this project to try to achieve, primarily to improve the system or improve use of the system, such as:

1. Comfort, ease of use.
    a. To have 75% of users be able to set up the electrodes and microphone without errors given written instructions within 5 minutes.
2. Latency of communication between controller and computer.
    a. With a mean average of 500 ms upon muscle contraction/vocalization to computer output.
3. Sensitivity of the sEMG sensors to outputs:
    a. Within 10 false positives and 10 false negatives per 100 intended keystrokes for all four individual outputs.
4. Additional outputs from four input sources, such as shown in Table 3.2.
    a. Example 1: Activation of left and right gastrocnemius muscle groups will lead to a new output on the keyboard, such as spacebar, corresponding to jumping.
    b. Example 2: Activation of both biceps leads to a different output on the keyboard, such as shift.
5. The system would be capable of connecting and outputting key strokes to any computer. This means compartmentalizing most of the code onto the Arduino Boards, instead of having an additional program run on the computer during use.

Table 3.2: Example Control Layout for EMG Signal Input and Keyboard Output

EMG INPUT/KEYBOARD OUTPUT	W	A	S	D	Spacebar	Shift
Left Gastrocnemius Group		X			X	
Right Gastrocnemius Group				X	X	
Left Bicep	X					X
Right Bicep/Left Tricep			X			X

4. Design

The device has different requirements that are necessary to function: It must be able to interface with an user with minimal noise input, while being able to process the gathered sEMG and audio data to determine if activity is a valid user input, or if it is generated from noise or accidental muscle activity. Once a valid user input is detected, it sends out keyboard signals to the computer to process and run.

Several additional features would have been implemented if there was enough time to do so. The device itself would be more comfortable if the electrodes were reusable, and if one could simply slide the device into position like on an armband, rather than sticking it to the skin via electrodes. In addition, the device being designed to be capable of plugging into different computer systems for user simplicity, much like most Bluetooth HID devices, would increase the ease of use for most users.

Initial prototyping and preliminary data were gathered using Delsys Trigno Centro. Delsys was utilized primarily to record signals from different muscle groups, primarily focusing on the upper arm, the lower arm, and the leg. The data gathered was used to determine optimal locations for placing wearable sEMG sensors on the body, with the deciding factors being

determined by both signal quality and signal interference. Signal quality was how corresponding sEMG data was with the muscular activity of the targeted muscle group, where moving or tensing the muscle group would elevate detected electrical activity read by the sensor. Signal interference was whether an sEMG sensor would pick up electrical activity from non-target muscular groups, where accidental movement or tension on muscles around the target muscle would result in increased sEMG activity from the sensor.

Matlab was used in the preliminary post-processing of sEMG data recorded from Delsys equipment, multi-signal processing of different measurements of sEMG data from different sensors, as well as establishing basic logic for keyboard control.

Finally, to test the feasibility of keyboard control on Windows computers, Python was utilized as a preliminary method to control a keyboard via programs, rather than the keyboard itself. Multiple python libraries were used to test this functionality, such as pyautogui, keyboard, and PyDirectInput. It must be noted that video games typically utilize key presses in the format of DirectInput objects, which work directly with device drivers, rather than character objects. This kind of interactivity can be seen whenever a first-person-shooter video game is loaded, and the mouse cursor vanishes to be replaced by a crosshair.

4.1 sEMG System Design:

The hardware modules are shown in Figure 4.1, and a physical image of the hardware used is shown in Figure 4.2 (See Appendix 1 for hardware specifications). The basic software logic is displayed in Figure 4.3 but is further elaborated in the text below.

Figure 4.1: Hardware Modules

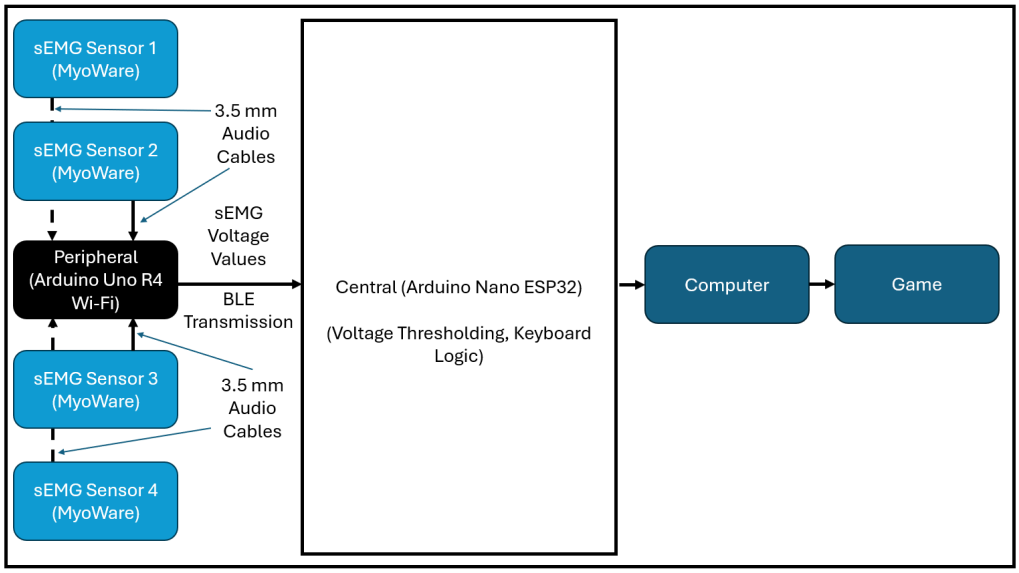


Figure 4.2: Peripheral Arduino Uno R4 WiFi, Battery Pack, MyoWare 2.0 Arduino Shield & Link Shield & Muscle Sensors. Bottom Left: Central Arduino Nano ESP32 board & USB-C cable

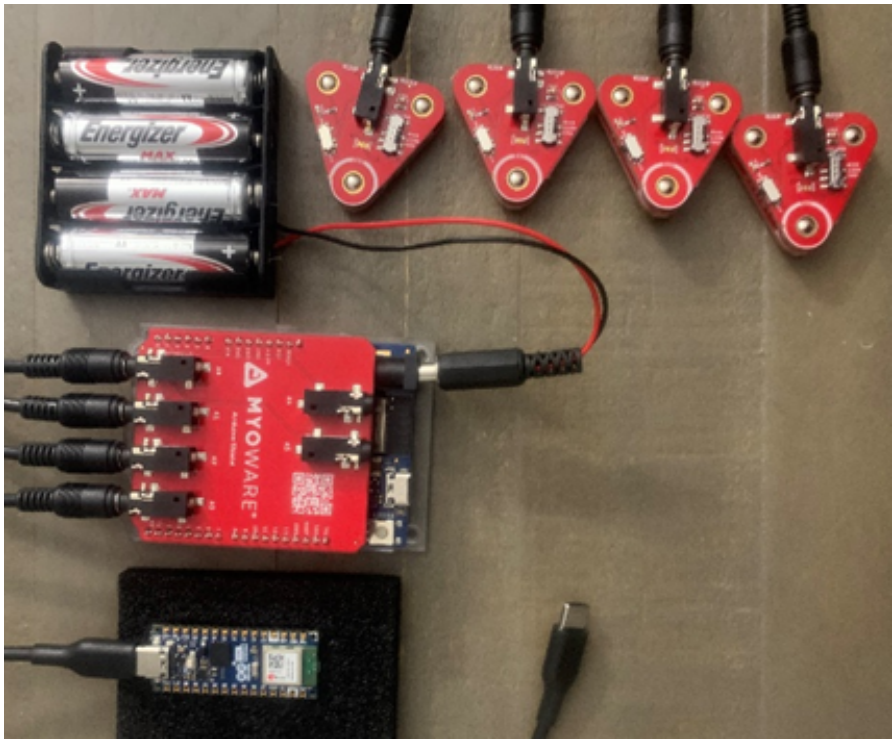
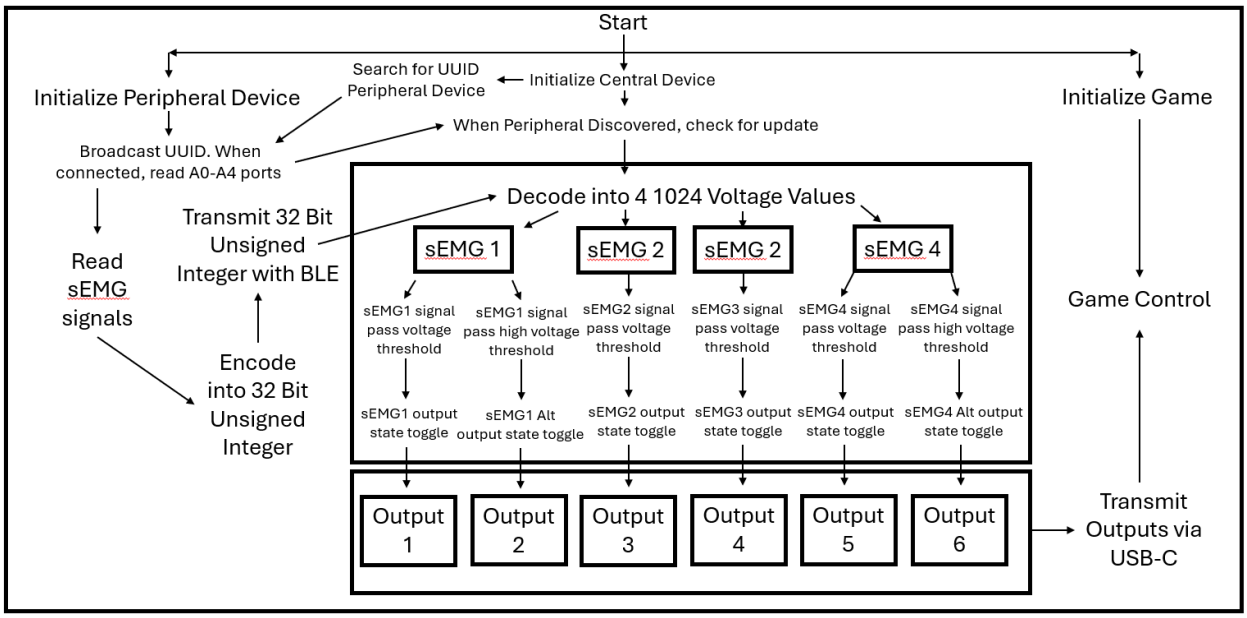


Figure 4.3: Software Logic, Final Variant



The MyoWare 2.0 Muscle Sensors read signals from the ECG Kendall 30x24 mm snap hydrogel electrodes when worn. The Muscle Sensors record, rectify, and envelope signals through analog bandpass filters, a full-wave rectification circuit, and an envelope detector circuit. It transmits the enveloped signal to the Link Shield extension, which then transmits voltage readings through the 3.5 mm TRS to TRS cable into the Arduino Shield. The Arduino Shield in turn sends the recorded signals into A0 through A3 ports on the Arduino Uno R4 Wifi board.

The sEMG sensors have to be physically separated from the device generating key outputs to a computer connected through USB-C, in order to isolate the system from the power grid, and to prevent the sensitive sEMG sensors from reading active inputs on any directly connected peripheral device. As such, the Arduino Uno R4 Wifi Board then transmits sEMG values through its ESP32-S3-MINI-1-N8 module, where the device broadcasts a BLE UUID (universally unique identifier) for the central device to recognize and connect to, which, when connected, transmits readings in the form of a 32 bit unsigned integer, attached with an

BLEUnsignedLongCharacteristic for the Arduino Nano ESP32 board to read. Readings from four different channels are compressed from 1024 bits to 256 bits and are assigned in specific positions within the 32-bit long unsigned integer, but are extracted and then expanded into 1024-bit-long unsigned integers in the Nano ESP32. Some resolution is lost, but there is still enough resolution to recognize sEMG inputs. Each sEMG reading is updated at a rate of 10 ms per reading.

The central device, the Arduino Nano ESP32, handles both voltage threshold logic and keyboard logic. Taking the single 32-bit unsigned integer and extracting sEMG recordings, the Nano ESP32 reads each channel at 50 ms per reading. The central device records these data values per iteration and follows an internal logic per iteration. Key Values are stored in the Arduino Nano as states for each sensor. They store whether a key output should be active or inactive, based on sEMG inputs, and will act as state toggles.

Each sensor sEMG recognition is tied to an initial delay. This delay ticks down when a signal is above an activation voltage threshold each iteration, until it reaches the delay interval. Then, the signal is recognized as valid, in which it then inverts the state of the associated Key Value, whether it be from inactive to active, or active to inactive. If the sEMG value goes below their associated deactivation voltage threshold, the delay resets back to 0. This delay system is to prevent voltage spikes from possible interference. In the same iteration, statements check to see if their associated Key Values are active or inactive. If they are active, then the Key output they are connected to generates their associated output. In the latest instance, differing voltage thresholds allowed for more than one output per sEMG sensor. This multiple threshold-based output was based on the left bicep and right bicep.

4.2 Voice Control Software Design

To optimize the voice control software for real-time responsiveness, accuracy and speed were prioritized. Traditional speech recognition, which involves complex algorithms to convert spoken words into text, was deemed too computationally intensive for the desired level of responsiveness. This means it would take too long for the software to analyze and understand spoken commands, creating a noticeable delay between the user speaking and the software acting upon those commands.

To achieve real-time responsiveness, the project employed simple Python scripts for audio analysis. These scripts focused on establishing volume thresholds for triggering the software, as well as initial experiments with pitch analysis. A specific volume threshold of 250 out of 255 was implemented using integer data from a PyAudio-generated audio stream (Pham 2006). This relatively narrow range was selected to prevent accidental activations caused by background noise or unintentional sounds made by the user. For example, a cough or a slammed door might reach a high enough volume to trigger the software if a wider threshold were used. By setting a specific volume threshold, the software can distinguish between these extraneous sounds and intentional user input intended to activate the voice controls

4.2 Decisions:

The final hardware configuration and software design went through a number of revisions, during which key decisions were made at each step of the process, which are described in the following section.

Delsys Triago, after some consideration, was deemed not viable for designing an sEMG interface. While each sensor was wireless, fairly easy to attach to the body, and transmitted high signal quality to the central computer, the limited program Delsys Triago was based on made it

difficult to design an active controller for a computer. Triago was designed primarily to be a product for human movement studies and leaves little room for modification.

Myoware was chosen as the basis for this project primarily due to the available support and modular equipment. Myoware sensors are meant to be easy to use in a variety of designs and are compatible with Arduino hardware. There is plenty of documented evidence that Myoware is perfectly functional for most needs. Their systems are compartmented compared to having to design a breadboard-based system, and they have both technical support and documented guides for using their equipment properly. Designing an exposed circuit with purchased electronic components, such as op-amps, capacitors, and resistors mounted on a breadboard, poses multiple risks, such as the individual components falling out of the breadboard, being misplaced, or being burned out. Buying a fully integrated design such as Myoware means less risk of damage to internal components.

The Myoware hardware requires Arduino components to function. The project had a choice between designing the final product with a single Arduino board or with a double board. After exploring what was possible with a single board, the Arduino Uno R4 Wifi, the project invested in purchasing another Arduino product, the Arduino Nano ESP32. Initially, the goal was for the Arduino Uno R4 Wifi board to act as both the central hub for the sEMG sensors as well as to communicate key presses to a computer over Bluetooth, but after some investigation, it was deemed inviable, as the Arduino boards cannot be configured to act as a wireless Bluetooth keyboard. However, they can be used as a wired USB-keyboard. With that in mind, the ESP32 was purchased, as information can be sent over BLE (Bluetooth Low Energy) from the Arduino Uno R4 to the ESP32, which the ESP32 board would interpret and then send keyboard strokes to a connected computer through a USB port. This, in turn, would convert the device into a plug-and-play system, allowing it to be connected to different devices and control

them instead of requiring a specialized program to initialize the device every time a user wishes to use it.

There were two different options for the project in regard to the electrodes used by the system. It is possible to use reusable electrodes instead of disposable electrodes, where the user can comfortably slip on a sleeve sewn with metal-based electrodes and adjust them. On the other hand, disposable electrodes would be more accurate, as each electrode was secured on the skin via adhesive. However, these electrodes are only useful in the short term and are harder to readjust.

More preferable sensors were found from a different source, known as uMyo sensors. The uMyo product is a wearable sEMG sensor attached to an armband meant to securely hold the sensors against the skin while they come equipped with dry, reusable electrodes. Multiple uMyo sensors can be connected to a single receiver, and the hardware present on the uMyo sensors is compatible with Arduino libraries and Arduino hardware. However, for this year, the Ukraine-based company producing the open-source sensors has stopped selling their equipment until September 2024, primarily because of the Ukraine war.

Determining voltage thresholds can depend heavily on the location each sensor is placed on the user and can vary heavily from person to person. After some research into different models of signal analysis, this project determined that voltage thresholds would be the best approach for the device. Alternate options were found, such as signal recognition, where software is trained to recognize the shape of different sEMG signals to produce different outputs. However, these alternate options were not pursued for the following reasons. While time constraints were a significant factor in deciding on voltage thresholding over signal recognition, considering the fact that this device is meant primarily for gaming, having to wait seconds for software to store, analyze, and recognize specific sEMG signals before creating an

output would cause a not-insignificant delay, considering most games require a rapid response time. While it is possible for this kind of signal recognition to be within the required high-response time period, because such software would be hosted on the ESP32 and its limited SRAM memory of 512 kB, it was determined that it would be safer to use the simpler option of voltage thresholds instead.

The project decided to implement a toggling system for controlling key inputs, where keys were toggled into the on or off states upon the detection of a corresponding valid sEMG signal. It is possible to design the device so that when an sEMG signal goes above a designated voltage threshold, the device holds down the key until the sEMG signal is no longer above the voltage threshold. However, that requires the user to constantly tense the muscles for the desired movement until they no longer need it. In a game like Minecraft, where the player may be moving forward for a long period of time, using toggling would help increase the long-term comfort of the user.

Initially, a pitch analysis section was also devised for the software. However, this idea was scrapped due to the unpredictability of the pitch analysis software utilized by PyAudio. These issues could be resolved, however, with the use of faster Fourier transforms (FFT), which are used to derive pitch information from waveforms, and better noise reduction (filtering). In its current state, the pitch analysis would not output the expected dominant frequency, and it is speculated that this is due to environmental noise and the way the data is processed by the program.

4.3 Optimization:

Software optimization would be heavily reliant on user feedback, user preferences, and voltage thresholding recalibration every session. sEMG readings rely on different characteristics, such as humidity, skin impedance, sensor locations, and so on. These factors

can change from day to day or even from session to session with the proposed sEMG interface. As such, calibrating voltage thresholds was needed, even if it was a minor adjustment of values, after reading corresponding outputs from each sensor. In addition, initial suggestions on combining two channels of sEMG inputs to produce a new key output were not guaranteed to be implemented; while possible, easier methods of adding additional controls may be discovered during development. Primary optimizations for voice control mainly come down to reducing noise in the signal, as our software does not use speech recognition but extracts data quickly from the signal. Better hardware would also create a clearer environment for the analysis.

The suggested control schematics, having the user be capable of moving in all four cardinal directions, may be reconsidered during gameplay testing: for instance, the user may not be utilizing an sEMG sensor as much as they could be if it was connected to a different key output. An example of changing the control schematic would be moving backward, as, alternatively, the user can turn around and move forward to go in that direction rather than traveling backward blindly. Another example would be changing outputs from toggles into single outputs or bursts of outputs, e.g., moving left or right by a few seconds rather than moving left or right continuously in response to their corresponding sEMG input.

II. Methods and Results

5. Methods:

Figure 5.1: Location of MyoWare muscle sensors. Top Left: Left Bicep. Bottom Left: Right Bicep. Right: Left and Right Gastrocnemius Groups.

The MyoWare 2.0 Muscle Sensors were placed on the left bicep, right bicep, and left and right gastrocnemius groups, as shown in Figure 5.1. These sensors read sEMG signals through the disposable Hydrogel snap electrodes. The initial approach for this project, once initial data and experimentation had been concluded, was to start by gathering data from four sensors, to ensure that equipment was functional and to ensure that data could be gathered and recorded simultaneously. As such, this required an initial assembly.

After testing the purchased Arduino Uno R4 Wifi through a test program to see if the board functioned, the project then moved to using MyoWare 2.0 ecosystem equipment,

including the muscle sensor, link shield, and Arduino shield. As documented by MyoWare, any computer used for Serial debugging of hardware directly connected to MyoWare muscle sensors was disconnected from the power grid to safeguard against electrical shock. In addition, while debugging, a wireless keyboard and mouse were used to interact with the computer. At the same time, it was connected to the MyoWare sensors, as per MyoWare documentation, to prevent the sensors from picking up a signal from interacting with a wired keyboard or trackpad. Otherwise, the muscle sensors would have had false readings and constantly output the maxed voltage value, as the pin would be read as High. The sEMG side of the project initially used Myoware's Sparkfun tutorial programs to determine whether the sensors were functioning upon purchase. The peripheral and central device programs were then expanded to create the programs used in the project, such as multisensor encoding, sEMG voltage thresholding, and keyboard logic. To enable Bluetooth functionality, the ArduinoBLE library was used (Arduino, 2019). Two other libraries were used to allow USB keyboard functionality from the central device to the computer: the USBHIDKeyboard.h library and the USB.h library (Saavedra, 2015) (Sloth, 2015).

For testing, the initial program would print values read from each sensor to the serial for the Arduino IDE software to see if it could accurately read muscle tension and relaxation. The readings are in analog-to-digital (ADC) values, where the analog voltage, ranging from 0V to 5V after the gain is applied, is then converted into a range from 0 to 1023. While it is possible to convert these values back into voltage ranges, it was decided that to reduce complexity, the above ranges of 0 to 1023 would be used for measuring and calibrating the system with voltage thresholds. Once debugging and testing were concluded, and all components were deemed viable, the project moved on to BLE transmission.

The Arduino Nano ESP32 Board was used as the central device, while the Arduino Uno R4 Wifi was used as the peripheral device for BLE transmission. The central device was

programmed to search for the peripheral device when enabled. The peripheral device is powered by an external AA battery pack. When connected, the voltage values of each sensor were condensed into an 8-bit unsigned integer, then collectively stored into a single 32 unsigned bit integer. This 32 unsigned-bit integer is then transmitted to the central device. When the central device detects an update in the data characteristic of the peripheral device, it updates its stored voltage values by extracting and then expanding each value from the 32 unsigned-bit integer. From there, it sends the newly obtained voltage values to Serial print for debugging.

When debugging is concluded and voltage values are being transmitted correctly to the central device, voltage, and keyboard logic are then implemented into the central device. Voltage logic was determined to be based on voltage thresholding. When an sEMG signal reaches above a voltage threshold for a period of time, typically by tensing a measured muscle group, the device would then store that instance as a valid signal, which is then used by the keyboard logic section.

Determining voltage thresholds can depend heavily on the location of each sensor placed on the user and can vary heavily from person to person. As such, each muscle group needs an initial sEMG measurement to determine appropriate sensor locations. sEMG sensor locations are verified and noted down for every new subject.

Keyboard logic uses the detection of valid sEMG signals from voltage thresholds and audio signals through volume thresholds to determine the user's intent. The device's goal was to be capable of moving the character forward, backward, left, and right, as well as output diagonal movements by pressing two adjacent keys together, e.g., forwards and left to move forwards diagonally to the left. In addition, the device's goal was to be capable of creating two more outputs, jumping, through the spacebar, and sneaking, through pressing the shift key. The target source and their associated key binds are shown in Table 5.1 below.

Table 5.1: Initial Control Layout for EMG Signal Input and Keyboard Output

INPUT	W	A	S	D	Spacebar	Shift
Left Gastrocnemius Group		X			X	
Right Gastrocnemius Group				X	X	
Left Bicep	X					X
Right Bicep			X			X
Loud Audio Signal					X	

For utilizing the spacebar to jump and the shift key to sneak in the game, it was initially decided that the sensors associated with both the left and right gastrocnemius groups, located in the calves of the legs, would be used. Seeing that the left calf was being utilized for moving left, and the right calf was associated with moving right, it was highly unlikely that both would be on at the same time, as that would produce direct opposite movements. The same holds true for the muscles associated with moving forward and backward. As such, those keys would be suitable for creating additional outputs.

However, the system changed from using two simultaneous inputs to using two different voltage thresholds, for the left and right biceps. A low voltage threshold would be used to generate one output and a high voltage threshold would generate another, as seen in Table 5.2. This allowed the user to create multiple outputs with a single muscle without the need for a simultaneous output of both muscles, where mistiming tensions may lead to the system only reading a single input instead of a combined multi-input.

Table 5.2: Final Control Layout for EMG Signal Input and Keyboard Output

EMG INPUT/KEYBOARD OUTPUT	W (Walk)	W + Alt (Sprint)	A	Spacebar (Single Input)	Spacebar (continuous)	D
Left Gastrocnemius Group			X			
Right Gastrocnemius Group						X
Left Bicep	Low Threshold	High Threshold				
Right Bicep				Low Threshold	High Threshold	

To implement key toggling, the ESP32 would send a keystroke over USB when a valid sEMG signal is detected, and continuously maintain that key output until the same valid sEMG signal is detected again. Upon the second detection of a valid sEMG signal, the ESP32 ceases to maintain that key output to the computer. Toggling is implemented for all four key outputs to reduce muscle strain on the user. The ESP32 also stores four values internally to reference, with all four pointing to an individual sensor or key press to be used for outputting the spacebar key or the shift key.

When two of the four-movement values, either ‘forward’ and ‘backward’ or ‘left’ and ‘right’, are on, they toggle the ‘space’ or the ‘shift’ keys on or off. As such, even when toggled, the device should still be capable of outputting the standard forward, backward, left, and right movement, even as the character is jumping or moving slowly. .

In order to perform additional interactions such as looking around, or interacting with the inventory, the user will be using a traditional mouse with built-in keys. These keys are rebound within the game to different controls, to allow the player to access the game’s inventory, and

perform standard mouse interactions such as attacking, aiming, and character orientation. Further possibilities for development will be detailed within the recommendations section.

For testing the device at different developmental stages, different trials were implemented to both record values to use for development and to test the device's functionality. First are the threshold calibration trials, which gather typical sEMG signals using the MyoWare muscle sensor environment. These signals would be used to program the central device's signal recognition software. Then, the sensitivity trials would test system outputs by measuring how responsive the sEMG system was, with a user tensing and relaxing various muscles to different sensors and the system generating key outputs in response to recognized sEMG signals.

Finally, Game Trials were used to test how viable the system was in comparison to a regular keyboard-mouse interface. The sEMG device and the voice control software were tested independently of each other to avoid crosstalk in trials. The in-game system of advancements, where completing certain tasks would be noted as game progression, was used for the sEMG control and score of the Google Dino Run game for the voice control software. Each trial performed with the sEMG-Mouse interface would measure the amount of advancements completed within a set timespan and compare it to a control trial completed with a keyboard-mouse interface to evaluate whether the sEMG-Mouse interface was comparable with normal users using a mouse and keyboard.

5.1 Threshold Calibration Trials:

To measure data to create initial values for voltage thresholds, the central device would be programmed to output voltage thresholds to a serial printout, where they would be ported into an Excel document for subsequent analysis. Each muscle group was measured in three trials. The device would record at least ten muscle activations over a short period of time per trial, and the Excel document would display the gathered sEMG data. Valid or intended sEMG

signals would be identified via manual annotation, where active signals and baseline signal voltage levels would be identified and then averaged. Mean signal duration would also be identified. All three trial values would be averaged to get an average baseline, average active, and average duration of active signals, from which final thresholds and threshold durations would be derived. It must be noted that the final voltage threshold and threshold durations must be below the calculated averages, or else risk a chance of false negatives.

Explicit calibration was not found to be necessary for the voice control software in its initial design. Currently, the software relies on a fixed volume threshold of 250 to determine when a user intends to issue a command. This standardized approach offers the advantage of simplicity. However, calibration could be a valuable addition to accommodate users who might struggle to consistently reach that volume level due to physical limitations.

If customization is implemented, a calibration process to establish a personalized, lower threshold would increase the software's accessibility. It's important to note that lowering the threshold comes with a trade-off: the system becomes more sensitive to background noise and unintentional vocalizations, increasing the likelihood of accidental triggers. To mitigate this, additional noise filtering techniques might need to be implemented alongside a customized volume threshold, adding a layer of complexity to the software design.

5.2 Sensitivity Trials:

To test the responsiveness of the sensors with the gathered voltage threshold and durations, a second test would be used to evaluate system performance. Each sensor and its respective output would be tested individually, where a series of thirty muscle contractions would be performed to see if the system would recognize these contractions as valid signals and create their output. These results would be recorded by how many signals were recognized out of the thirty muscle contractions and used to evaluate the sensitivity of the system.

5.3 Game Trials:

A single user was selected to control the devices, due to time constraints, considering the various factors that can change each use of the device per individual, such as muscle strength, electrode placement, and so on. The single user had past experience with the games used to test the sEMG system, Minecraft and Google Dino Run. Scoring was determined via the in-game advancement system, where certain actions would be noted as a significant benchmark for in-game progress, and score respectively.

To test the completed devices, it was decided that Minecraft and Dino run would be perfect. Different tasks were derived for a player to interact with, primarily, goals and objectives within the game that would be common for any new player.

These tasks were based on the already present in-game advancement system and score. Advancements in Minecraft are challenges for players to complete, effectively meant to act as a guide for newer players to naturally progress towards. But they can also be used as benchmarks to evaluate player progress over a period of time.

For the purposes of evaluating the user's ability to use the current system, the sEMG+Mouse interface would be used for 20 minute test sessions, in which the number of achievements would be measured and recorded. If the system suffers a malfunction, the test session will be paused to fix the problem, before being resumed. The amount of time it takes to fix the issue will not be counted as time used for the test session. This decision was made because this test is focusing on how responsive the system is to the user, and testing software configuration; hardware issues are normally solved by replacing or readjusting electrodes or hardware connections.

For the voice control test, the user was simply asked to play Google Dino Run using sharp, precise vocalizations to trigger the space bar via the microphone placed in front of them, causing the character to jump over incoming obstacles. These obstacles increase their speed as time progresses, increasing difficulty. The user's score will increase until they fail to jump over an object, at which point the score would be recorded. 30 trials of this were completed with a control and experimental group. The control group utilized a standard keyboard, while the experimental group used the voice control software and microphone. Audio capture and analysis was all handled within Python. Libraries included PyAudio, Pyautogui, and librosa was originally considered for pitch but were ultimately not needed.

After a trial, the player would take the time to note down results and observations, as well as any technical issues that may have occurred during playing. This would include deaths, time duration, and any user notes or adjustments made to the system. These results would be compared to playthroughs with traditional keyboard and mouse controls for both games.

6. Results:

6.1 Threshold Calibration Trials:

The overall goal of the low-level calibration trials was to establish thresholds for the sensor, determining what was the most comfortable max output and duration for a user, and what was the average relaxed baseline for the user. An individual sensor was recording sEMG values from their designated muscle group, and the user was requested to tense, clench, or activate the muscle for a duration of time that was comfortable for them. The data was recorded in sets, which could be plotted as such in Figure 6.1. The average baseline, average active, and average signal duration were recorded to determine the final thresholds and durations of signals.

Figure 6.1: Example of Voltage Outputs in Threshold Calibration Trials, Left Bicep

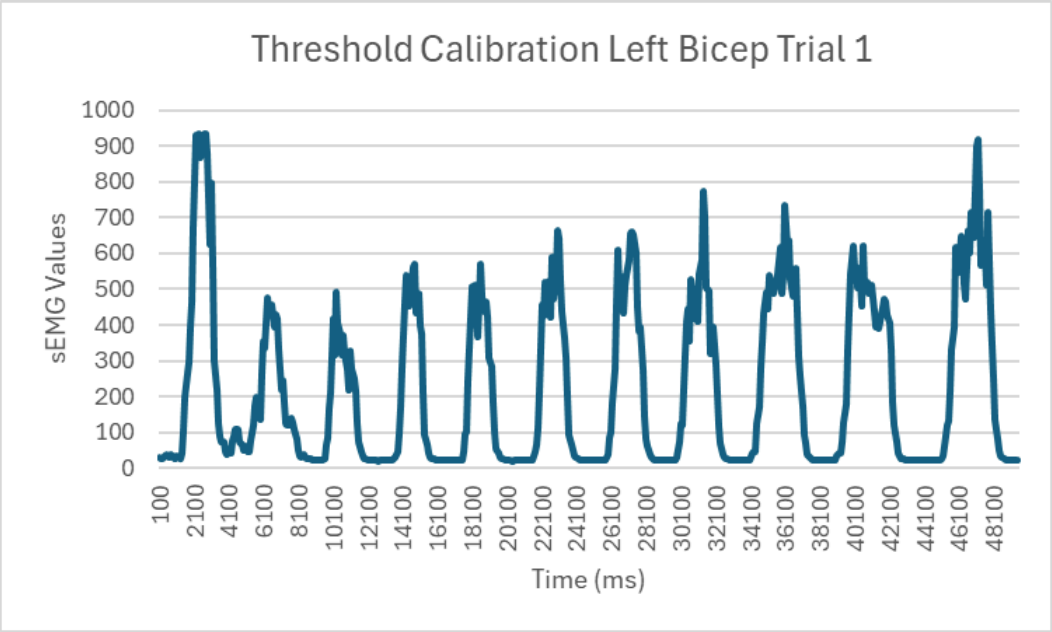


Table 6.1: Average Durations and Signal Outputs. *[1]: (Normalized Signal ADC Values)*

Muscle Group	Average Baseline[1]	Average Active[1]	Average Duration (ms)	Final Thresholds[1]	Threshold Duration
Left Bicep	41.63	357.83	878.00	300	800
Right Bicep	43.23	195.56	1736.97	150	800
Left Gastrocnemius	33.65	126.91	1267.88	100	1000
Right Gastrocnemius	38.40	123.28	1315.99	100	1000

The resulting average baseline, average active, and average durations of active sEMG signals are shown in Table 6.1. From these values, threshold durations and final threshold values were derived in ADC values, which were used to create an initial sEMG controller interface. Signal duration for the right bicep was significantly lowered from 1700 ms to 800 ms

due to wanting both left and right bicep inputs to respond quickly and have equal response times.

It must be noted, however, that these averages can change, not just from person to person but from trial to trial, based on the placement of the electrodes, skin impedance, and other environmental factors. These thresholds, however, were useful for encoding an initial value to calibrate as needed in future trials. In addition, with duration, it was decided that for a faster response, shorter durations would be implemented to be within the range of 1 second, and despite the drastic change in comparison to the average durations of the Right Bicep group, later tests have shown that this held minimal impact to results.

6.2. Sensitivity Trials:

After initial voltage calibrations, the device was then tested via four different outputs to evaluate the responsiveness of each sensor compared to each individual output. These outputs were tied specifically to one sensor, rather than one sensor being capable of creating multiple outputs, as this test was meant to evaluate the effectiveness of these sensors in response to a user's desired output.

A true positive output would correspond to muscle tension, followed by a keyboard output. A false positive would be no muscular tension, then keyboard output. True negative would be muscle relaxation or no muscular tension, then no keyboard output. A false negative would be muscular tension, then no keyboard output.

In Table 6.2, the results show the true positive outputs out of all outputs, with the note that no keystrokes were recorded that indicate a false negative output. This is because of how it only recognizes valid signals when they rise above certain voltage thresholds. When the

sensors and electrodes are applied to the body correctly, as they were in these series of tests, if a sensor is unable to read an sEMG signal, they default to a single voltage value. As this voltage value is always below the voltage threshold, the device never reads a false negative. However, this result can theoretically change if the sensors are applied incorrectly to be outputting continuously changing erroneous magnitude voltage values of an sEMG signal, but in that case, the user should be able to notice and adjust the sensors.

Table 6.2: Device Sensitivity Test

Muscle Group	# Keystrokes/Number of Intended Activations	%Sensitivity
Left Bicep	29/30	96.67
Right Bicep	25/30	83.33
Left Gastrocnemius	24/30	80.00
Right Gastrocnemius	23/30	76.67

6.3 Game Trials:

The key outputs of the sEMG device had been changed from the initial configuration, as per user request. When initially playing Minecraft, the user relied heavily on moving forwards, left, and right, but rarely, if ever, moved backward. As such, inputs and outputs had a change of configuration, as seen in Table 5.2. The game trials used this configuration to test the feasibility of the sEMG-Mouse interface.

Scoring for the sEMG tests was determined via the in-game advancement system for Minecraft, where certain actions were noted as a significant benchmark for in-game progress. After initial trials, the user noted that one of the sensors was rarely, if at all being utilized for controlling the in-game avatar, resulting in a change of outputs (see chapter 7.3, Game Trials under Analysis and Discussion).

On average, the user with the sEMG-mouse interface scored at least 1 advancement or score lower than the control interface (mouse and keyboard), and showed no relative improvement over time, as shown in Figure 6.2. Trial 5 skews the number of advancements downwards, though trial 5 had multiple hardware faults, resulting in a lower number of advancements. Overall, though, the resulting average scores of the sEMG-mouse interface were close to that of the traditional mouse and keyboard interface, as seen in Figure 6.3, and the difference in scores could be accounted for due to the stronger familiarity of a user with the traditional keyboard and mouse, and the lower number of faults experienced with the keyboard and mouse.

Figure 6.2: Advancement Scores per Trial, with Control and sEMG+Mouse Interface

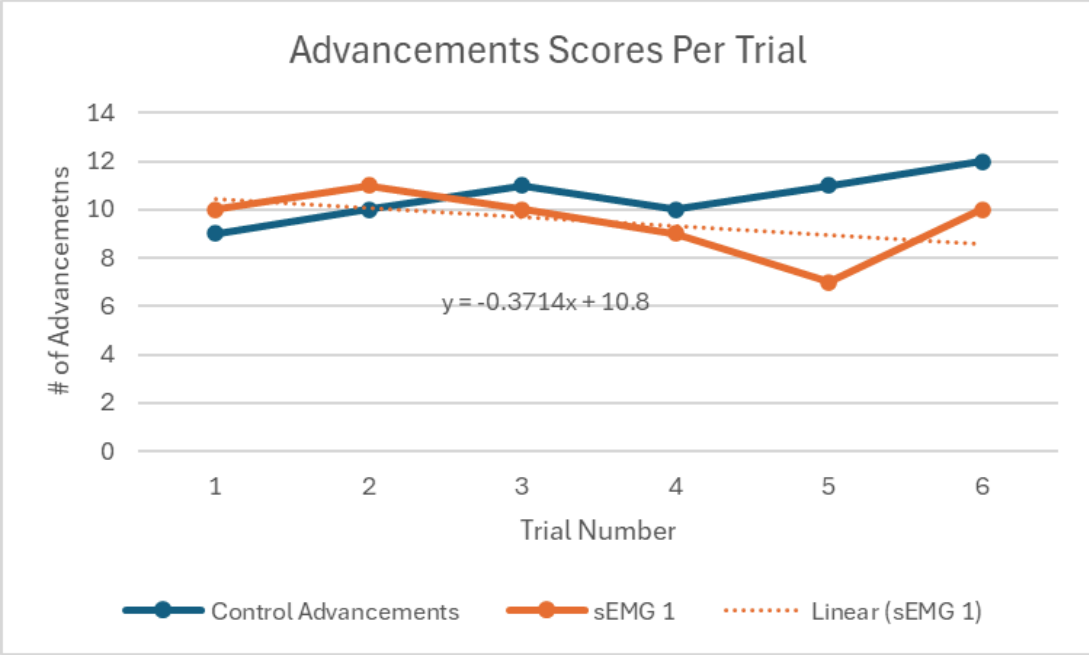
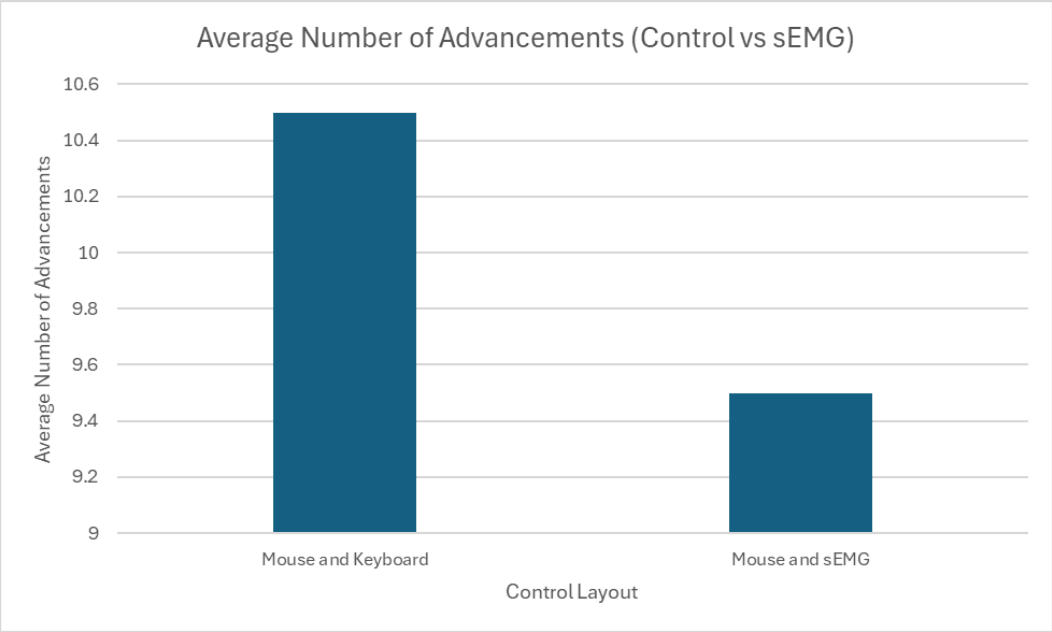


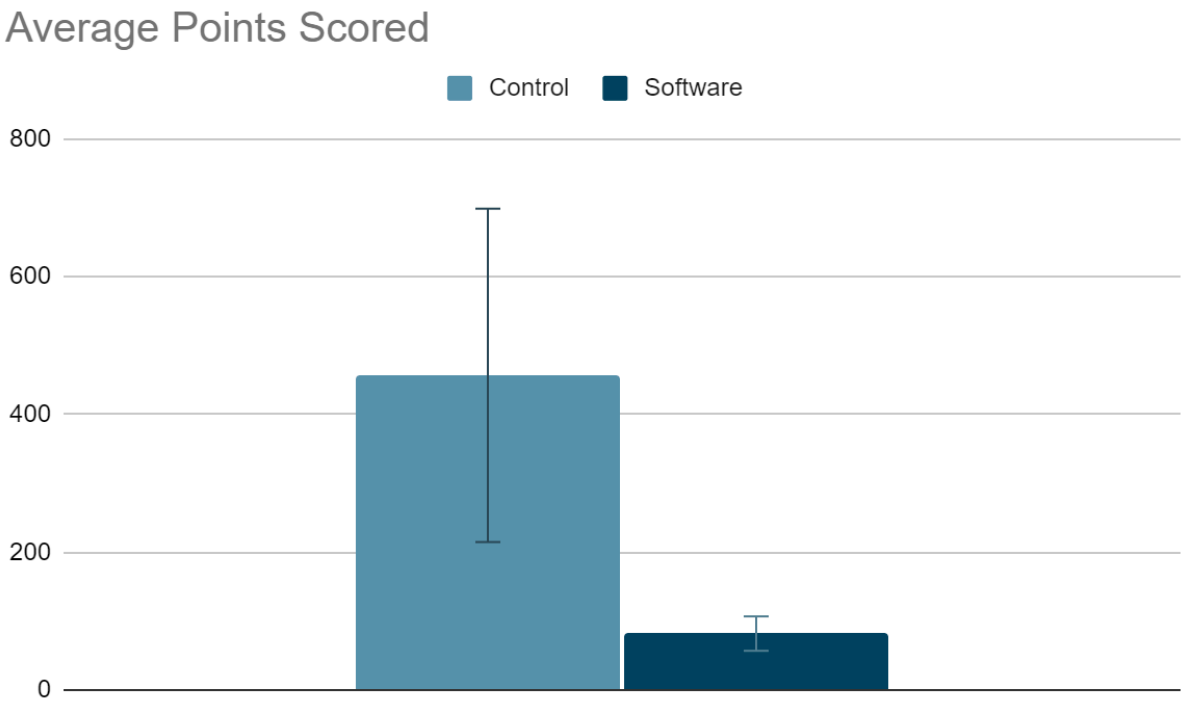
Figure 6.3: Average Number of Advancement over Collection of Trials



In addition, the traditional gameplay trials were done with a user with full operational use of both hands, while this system was designed for a user with the inability to operate one hand. It may be possible that a player with an arm amputation would find it easier to operate the sEMG-mouse interface over the traditional keyboard interface, though, for the project's duration, it was unable to find such a user to test the device with.

For the voice control software, 30 trials were done for each group, control and experimental. In-game score was used to measure progress and was recorded for each trial. The mean scores for the control and experimental group were 456.73 and 81.86 (Figure 6.4) respectively. This was expected, as participants in the control group were already familiar with the keyboard and mouse interface and this was the interface the game was designed for.

Figure 6.4. Average points Scored in Dino Run trials



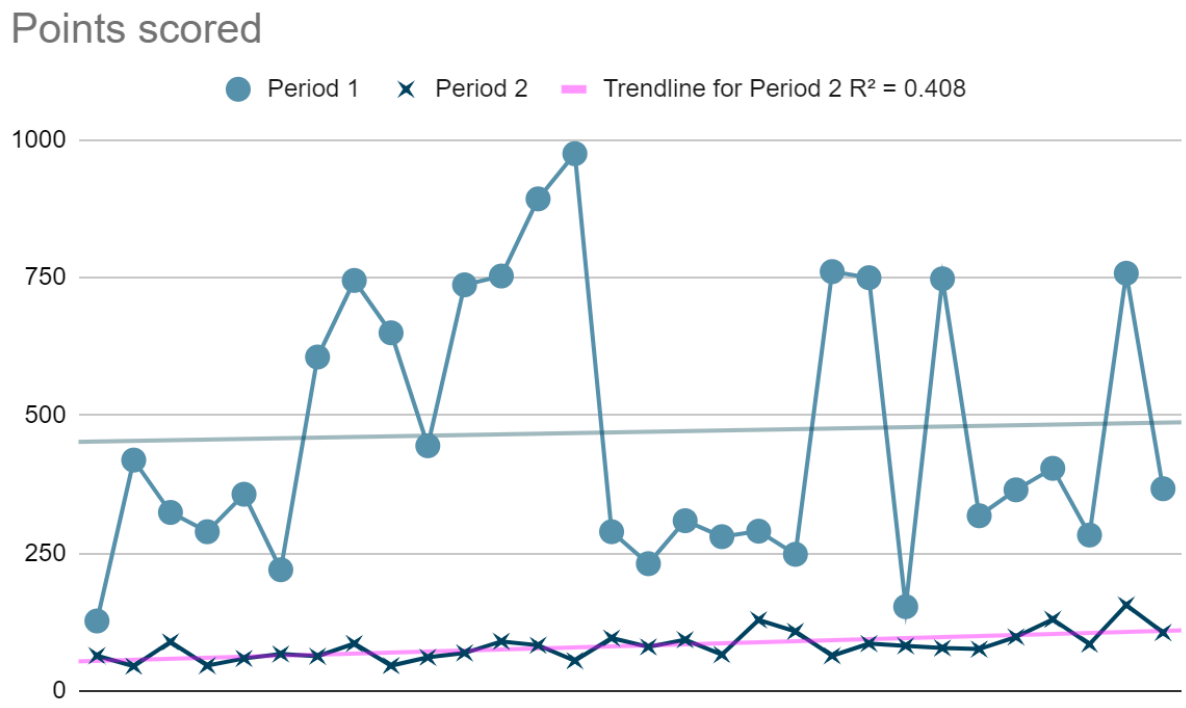
6.4 Observations:

Overall, the sEMG-Mouse interface allows users to play video games through muscle contractions and relaxations, and the voice control software allows for the use of vocalizations for added dimensions of control. While the device does need more adjustment and refinement, a user can use the interface to play a game consistently. The player can move in multiple directions, including diagonally, and when remapped to their desired specifications, the player is capable of making various in-game achievements much like a regular player would. The accumulating muscle strain from the use of the system is still noticeable after long durations. The sEMG-Mouse system is noticeably different from the traditional keyboard-mouse interface, using different muscles and movement patterns, so a user has to adjust over a period of time

and may make more mistakes than a traditional player, as shown in the comparison between the control and the sEMG-Mouse interface.

Additionally, the writers would like to make note of the improvement over time seen in the experimental voice control group (Figure 6.5). While shallow with an R squared value of 0.408, this still shows improvement over time, suggesting the device is learnable and could even potentially be easier to use for certain situations than a traditional input setup.

Figure 6.5. Points scored vs trials (Period 1: Control. Period 2: Experimental)



7. Discussion:

7.1. Low Threshold Calibrations:

The first set of tests were useful for determining baseline voltage thresholds and durations for the purposes of coding key logic and signal validation logic. However, every time

the electrodes on the body were replaced after each session, the voltage thresholds may be altered even due to a slight change in placement. Therefore, recalibration was a near-constant factor in every test and needs to be noted for future developments of the device. In addition, if the electrodes need to be adjusted, the single-use snap electrodes are unsuitable, as they lose adhesion with the skin with every change in placement, resulting in more input noise on the system and less reliability of staying in place.

7.2. Sensitivity Testing:

Sensitivity testing has shown that the sEMG sensors can report false negatives based on software or hardware configuration. Other than the left arm sensor, all sensors had above the targeted 10% of false negatives outputs. This percentage could be improved via further calibration, or more specific valid signal identification than voltage thresholding, or even improvements to the current voltage threshold valid sEMG-signal logic stored on the ESP32. However, the user reported that correcting for false negative outputs were akin to correcting a wrong key press on the keyboard, where the user would generate an sEMG input again for the system to recognize moments later.

7.3. Game Trials:

For early in-game trials, initial control schematics were based on moving forwards, backward, left, and right. However, the user showed an increased preference for utilizing the ‘w’, ‘a’, and ‘d’ keys, with minimal or nonexistent use of the ‘s’ key for moving backward. As such, with a new set of trials, the user was given the option to redesign controls to their preference. In addition, after sEMG sensor calibration, the user could get multiple desired outputs by different voltage thresholds, rather than combined signal inputs. This was used in two instances for the right and left arm. In the right bicep sensor, a high voltage threshold on the right arm would output multiple jumps, while a lower voltage threshold would output a single jump. In the left

bicep sensor, a high voltage threshold would output a ‘sprint’ motion, while a lower voltage threshold would create a ‘walking’ forward motion at lower in-game speeds.

Gameplay trials show that playing with this alternative interface resulted in lower advancement gain compared to traditional playthroughs, but this could be accounted for by both possible improvements to be made to the device and a longer duration of training time required for a user to be capable of controlling a computer through the interface. The experimental group was still capable of the same actions and motions of a traditional player without requiring the use of the left hand for interaction with the computer. In this regard, the project was a success, but various improvements can be made, such as improving signal recognition and sensor quality, removing potential sources of noise or interference, such as the 3.5 mm TRS cables, and replacing them with wireless options, and so on.

Similarly, for the voice control software, gameplay trials initially showed lower scores for the experimental group, while the control group consistently scored higher. This discrepancy can be attributed to several factors. The most substantial factor is undoubtedly the users' familiarity with the traditional control setup. Years of experience using keyboard or gamepad controls provide an inherent advantage for the control group. However, it's encouraging that the experimental group demonstrated improvement over successive trials, showcasing the learnability of the voice control system.

With extended practice and familiarization, it is possible that the experimental group could eventually surpass the control group in terms of score. This potential outcome highlights the intuitive nature of the experimental voice control system. Once a user overcomes the initial unfamiliarity, voice control could enable more fluid and natural interaction with the game environment, potentially leading to gameplay advantages over the traditional control scheme.

It must be noted that at the initial stage, the equipment had multiple flaws. For instance, the sensors are noted to be sensitive in the documentation, where they can detect the activation of any keyboard, mouse, or trackpad directly connected to the computer, when the sensors are directly connected through a computer port. While this issue can be resolved by transmitting recorded voltage values through Bluetooth Low Energy, this does not prevent the sensors from being able to detect and read each other's sEMG signals in particular instances. It is currently unknown if the board is accidentally reading other sensors through the pins, if the audio cables are not isolated enough, or if the sensors can accidentally read each other through the body. The most likely candidate for faults, however, are the audio cables, seeing that the physical contact of audio cables for different sensors holds a high tendency of producing similar outputs on pins, which can be resolved by adjustment of the cables. An immediate resolution would be to transition to a completely wireless version, but this cannot be done with this project due to time constraints.

7.4. Ethical Concerns

Environmental:

The device developed in this project allows amputees to gain access to playing computer games more effectively than without the device. Environmental impacts come from the manufacture of the device and very modest electricity usage commensurate with that used by other low-voltage devices, such as wireless keyboards and mice. Production of the components of the device are manufactured in developing countries and must be shipped to the user location, resulting in impacts on the environment. These environmental impacts can also have negative impacts on the health of people in areas where the materials are sourced and manufactured. Under the global and economic statements, some mitigating outcomes are listed. Once produced and shipped, the device has a long lifespan, meaning that the impact of

production is spread over many hours of use compared to most disposable items. Further, people who turn to computer gaming because of this device might reduce more resource-intensive activities such as driving to see friends. While the prototype sEMG device used disposable ECG Hydrogel Electrodes, the project advocates replacing these with reusable electrodes in the production product.

Social:

Although the project device is designed for amputees who are looking for entertainment, others including their family and friends will also benefit from their increased enjoyment of life and social mobility. In particular, online gaming can be a way to gain and develop friendships, and so enabling amputees to partake is hugely positive not just for the amputee but for their new friends and for their families who benefit from seeing their disabled family members enrich their lives. Other amputees might benefit also as the word spreads about the usefulness of the device, and they also start to use it. One concern might be that amputees using the device might develop a gaming addiction, but for most of the amputee population, it is more likely that their increased social mobility benefits outweigh such a downside. It is well-known that increased social interaction carries many health benefits, not least of which is mental (Johannes et al., 2021), benefitting amputee users. Video gaming was seen as a life-saver for many during the recent COVID pandemic as even able-bodied individuals faced isolation. In addition to mental health benefits, there is evidence that training the muscles for amputees can have additional physical therapy health benefits (Liu et al., 2022).

Global:

The device is a product that will be useful worldwide. Given the relatively higher percentage of amputees in many developing countries in Asia and Africa, it may be of higher use overseas than in the US. The reason for the higher amputation rate in developing countries is a

combination of lower healthcare standards and higher accident rates (McDonald et al., 2020; Yuan et al., 2023). When expanding use to other countries, consideration must be made of the socio-economic implications including the potential for application to other types of devices, such as smartphones, as well as applications for sEMG technology for other types of applications including for work or education. As noted under environmental regulations, components for the device will mostly be manufactured in developing countries where there will be harmful environmental impacts, but such manufacture will create jobs and improve healthcare as a mitigant. Because the environmental impacts are small, and are mitigated by job creation, there should be very limited political concerns about the production of these components. Further, the components are available off-the-shelf, so the manufacturability of the product is also not a major concern.

Economic:

Except where the components for the product are manufactured, the economic impact is likely to be modest. Better mental and physical health outcomes reduce health care costs, for example. However, the application, if extended to other purposes such as education and work-related activities, could make the difference between people not being engaged in the workforce and being able to be meaningfully employed. In fact, digitalization is being tied closely to sustainability and economic advancement by multilateral institutions such as the World Bank (Perez et al., 2023) although there are acknowledged offsets for environmental degradation and resource use. The point though is that for amputees gaining access to the digital world can make a significant difference in their life outcomes.The device created in this project is, however, a precursor for the kinds of devices that would make this contribution.

8. Conclusion:

The project was able to create a multimodal interface for controlling a computer character within a virtual environment. While performance has shown that in its current state it is not able to match with traditional keyboard and mouse, our interface is still a viable alternate option for playing computer games, especially for those with an amputated hand. Multiple possible improvements to the device have been observed and noted in recommendations, where with further development, this alternative interface may become an adequate substitute to the traditional keyboard and mouse, especially for amputees.

9. Recommendations:

There are multiple points of improvement that could be made for the interfaces. For instance, on rare occasions, there is a present signal interference where other sensors' voltage values are being read through the wrong Arduino ports, most likely through the audio cables. In addition, the Arduino Shield ports may fail to function until readjustment of the audio cables. One recommendation to correct these faults would be removing the audio cables and transitioning the wired electrodes to a wireless system, where each sensor is its own contained electronic environment transmitting voltage values over BLE to a central device.

Another recommendation is the transition from disposable electrodes to reusable electrodes. The project had consumed around 200 disposable electrodes during testing and development. While these disposable electrodes were around 20 dollars per pack of 50, recurring costs can be limited by transitioning to reusable electrodes, such as conductive fabric electrodes that can be sewn into sleeves. This would also allow the user to adjust the electrodes as well, for both comfort and to refine acquisition of sEMG signals.

A third recommendation would be to address the fact that a mouse is still needed to interact with a computer game, which limits this product's reach to those with a single functioning hand. Users with dual hand amputation would not be capable of using this interface. As such, an sEMG-Gyroscope interface can be developed, where muscle inputs could be used to control inputs such as left and right clicking, and movement of the hand or head could be used to control the cursor of the computer. In addition, this could also be done by implementing the voice control system, or a combined mix of the two systems discussed above.

For the voice control software, substantial refinements can be made to enhance noise reduction and filtering. Isolating the critical frequency range used for voice commands will significantly improve the software's ability to distinguish between intentional input and background noise. More robust filtering of unnecessary high and low frequencies will refine the audio signal, leading to more reliable key activations and reducing the chance of false positives.

Introducing pitch analysis offers an exciting opportunity to expand the control scheme. By analyzing the tone of the user's voice, the software can differentiate between different commands or even adjust in-game parameters continuously. However, successful implementation hinges on optimizing the software's calculations for faster processing. Any noticeable delay between vocal input and the corresponding action will severely degrade the user experience, making the system feel unresponsive.

Additionally, it's highly recommended that a high-quality microphone be used for optimal results. A clearer input signal reduces the burden on the filtering algorithms, simplifies analysis, and ultimately contributes to a more accurate and responsive system.

The need for recalibration and adjustment is not noticeable until the system becomes faulty, such as not responding to muscle tensions or outputting false readings. A suggestion for

improving this is to readjust or recalibrate sensors. Also, a way to see values other than through the Serial monitor on the IDE, like a side window, would be helpful for troubleshooting.

Finally, calibration of voltage thresholds and remapping key outputs still relied on using the Arduino IDE, to directly reprogram key outputs and redefine voltage thresholds and delays within the software contained on the Arduino Nano ESP32. In the future, once the sensors are placed, the device should be capable of making a live recording of sEMG signals for each sensor, and then prompt the user with a window to input different voltage thresholds and delays based on the live recording. The window should also be capable of changing outputs, based on both voltage thresholds and sEMG signal source.

III. References:

Arduino Sa, 2019, ArduinoBLE, https://github.com/arduino-libraries/ArduinoBLE. (2024)

Amputations. (n.d.). Physiopedia. https://www.physio-pedia.com/Amputations, accessed April 20, 2024.

Bora, D. J., & Dasgupta, R. (2020). Estimation of skin impedance models with experimental data and a proposed model for human skin impedance. *IET Systems Biology*, *14*(5), 230–240. https://doi.org/10.1049/iet-syb.2020.0049

Chai, X., Zhang, Z., Guan, K., Lu, Y., Liu, G., Zhang, T., & Niu, H. (2020). A hybrid BCI-controlled smart home system combining SSVEP and EMG for individuals with paralysis. *Biomedical Signal Processing and Control*, *56*, 101687. https://doi.org/10.1016/j.bspc.2019.101687

Esposito, D., Centracchio, J., Bifulco, P. et al. A smart approach to EMG envelope extraction and powerful denoising for human–machine interfaces. Sci Rep 13, 7768 (2023). https://doi.org/10.1038/s41598-023-33319-4

Fu, Z., Li, H., Ouyang, Z., Liu, X., & Niu, J. (2020). Typing Everywhere with an EMG Keyboard: A Novel Myo Armband-Based HCI Tool. In M. Qiu (Ed.), Algorithms and Architectures for Parallel Processing (Vol. 12452, pp. 247–261). Springer International Publishing. https://doi.org/10.1007/978-3-030-60245-1_17

Johannes Niklas, Vuorre Matti and Przybylski Andrew K. 2021, Video game play is positively correlated with well-being, R. Soc. Open Sci.8202049; https://royalsocietypublishing.org/author/Przybylski%2C+Andrew+K

Konrad, P. (2006). *The ABC of EMG* (1.4). Noraxon U.S.A., Inc. https://www.noraxon.com/wp-content/uploads/2014/12/ABC-EMG-ISBN.pdf

Lehman GJ, McGill SM. The importance of normalization in the interpretation of surface electromyography: a proof of principle. J Manipulative Physiol Ther. 1999 Sep;22(7):444-6. doi: 10.1016/s0161-4754(99)70032-1. PMID: 10519560.

Liu Y, Silva RML, Friedrich JB, Kao DS, Mourad PD, Bunnell AE. Surface Electromyography-Driven Therapeutic Gaming for Rehabilitation of Upper Extremity Weakness: A Pilot Study. Plast Reconstr Surg. 2022 Jul 1;150(1):125-131. doi: 10.1097/PRS.0000000000009208. Epub 2022 May 10. PMID: 35544314; PMCID: PMC9246860.

McDonald CL, Westcott-McCoy S, Weaver MR, Haagsma J, Kartin D. Global prevalence of traumatic non-fatal limb amputation. Prosthet Orthot Int. 2021 Apr 1;45(2):105-114. doi: 10.1177/0309364620972258. PMID: 33274665.

Merletti, R., & Muceli, S. (2019). Tutorial. Surface EMG detection in space and time: Best practices. *Journal of Electromyography and Kinesiology*, *49*, 102363. https://doi.org/10.1016/j.jelekin.2019.102363

Murphy, B.B.; Scheid, B.H.; Hendricks, Q.; Apollo, N.V.; Litt, B.; Vitale, F. Time Evolution of the Skin–Electrode Interface Impedance under Different Skin Treatments. Sensors 2021, 21, 5210. https://doi.org/10.3390/s21155210

Myoware Muscle Sensors - SparkFun Electronics. https://www.sparkfun.com/myoware?ref=msaavedra.com

Getting Started with the MyoWare® 2.0 Muscle Sensor Ecosystem - SparkFun Learn. https://learn.sparkfun.com/tutorials/getting-started-with-the-myoware-20-muscle-sensor-ecosystem/arduino-example-2-transmitting-sensor-data-via-bluetooth---single-sensor

Olmo Md, Domingo R. EMG Characterization and Processing in Production Engineering. Materials. 2020; 13(24):5815. https://doi.org/10.3390/ma13245815

Pérez-Martínez, J., Hernandez-Gil, F., San Miguel, G., Ruiz, D., & Arredondo, M. T. (2023). Analysing associations between digitalization and the accomplishment of the Sustainable Development Goals. Science of The Total Environment, 857, 159700. https://doi.org/10.1016/j.scitotenv.2022.159700

Pham, Hubert (2006). PyAudio Documentation https://people.csail.mit.edu/hubert/pyaudio/docs/#class-pyaudio-stream

Raez MB, Hussain MS, Mohd-Yasin F. Techniques of EMG signal analysis: detection, processing, classification and applications. Biol Proced Online. 2006;8:11-35. doi: 10.1251/bpo115. Epub 2006 Mar 23. Erratum in: Biol Proced Online. 2006;8:163. PMID: 16799694; PMCID: PMC1455479.

Saavedra, L, 2015, USBHIDKeyboard.h, https://github.com/espressif/arduino-esp32/blob/master/libraries/USB/src/USBHIDKeyboard.h. (2024)

Sánchez-Velasco, L. E., Arias-Montiel, M., Guzmán-Ramírez, E., & Lugo-González, E. (2020). A Low-Cost EMG-Controlled Anthropomorphic Robotic Hand for Power and Precision Grasp. *Biocybernetics and Biomedical Engineering*, *40*(1), 221–237. https://doi.org/10.1016/j.bbe.2019.10.002

Sloth, K, 2011, Usb.h, https://github.com/arduino-libraries/USBHost/blob/master/src/Usb.h. (2024)

Smidt KP, Bicknell R. Prosthetics in Orthopedics. [Updated 2023 Jul 24]. In: StatPearls [Internet]. Treasure Island (FL): StatPearls Publishing; 2024 Jan-. Available from: https://www.ncbi.nlm.nih.gov/books/NBK570628/

Urone, P. P., & Hinrichs, R. (2012). 23.11 Reactance, Inductive and Capacitive. In College Physics. OpenStax. https://openstax.org/books/college-physics/pages/23-11-reactance-inductive-and-capacitive?query=reactance&target=%7B“index”%3A0%2C“type”%3A“search”%7D#import-auto-id1169737988958

Vogel, J., Hagengruber, A., Iskandar, M., Quere, G., Leipscher, U., Bustamante, S., Dietrich, A., Hoppner, H., Leidner, D., & Albu-Schaffer, A. (2020). EDAN: An EMG-controlled Daily Assistant to Help People With Physical Disabilities. *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 4183–4190. https://doi.org/10.1109/IROS45743.2020.9341156

Yin, G., Zhang, X., Chen, D., Li, H., Chen, J., Chen, C., & Lemos, S. (2020). Processing Surface EMG Signals for Exoskeleton Motion Control. *Frontiers in Neurorobotics*, *14*, 40. https://doi.org/10.3389/fnbot.2020.00040

Yuan B, Hu D, Gu S, Xiao S, Song F. The global burden of traumatic amputation in 204 countries and territories. Front Public Health. 2023 Oct 20;11:1258853. doi: 10.3389/fpubh.2023.1258853. PMID: 37927851; PMCID: PMC10622756.

Zachary Boddy (October 15, 2023). "Minecraft crosses 300 million copies sold as it prepares to celebrate its 15th anniversary". Windows Central. Archived from the original on October 15, 2023.

Zhang R, Hong Y, Zhang H, Dang L, Li Y. High-Performance Surface Electromyography Armband Design for Gesture Recognition. Sensors (Basel). 2023 May 21;23(10):4940. doi: 10.3390/s23104940. PMID: 37430853; PMCID: PMC10222313.

IV. Appendices

Appendix A: Specifications of Electronic Components

Table A.A: Arduino Uno R4 WiFi specifications - Peripheral Device Arduino Uno R4 WiFi

Specification	Details
Microprocessor	48 MHz Arm® Cortex®-M4 microprocessor with Floating Point Unit
Memory	256 kB Flash Memory 32 kB SRAM 8 kB Data Memory (EEPROM)
Power	Operating voltage for RA4M1 is 5 V Recommended input voltage (VIN) is 6-24 V. Barrel jack connected to VIN pin (6-24 V). Power via USB-C® at 5 V
Peripherals:	Capacitive Touch Sensing Unit (CTSU) USB 2.0 Full-Speed Module (USBFS) 14-bit ADC Up to 12-bit DAC Operational Amplifier (OPAMP)
Secondary MCU:	ESP32-S3-MINI-1-N8
Microprocessor	Xtensa® dual-core 32-bit LX7 microprocessor (with single precision FPU), up to 240 MHz
	384 KB ROM 512 KB SRAM 16 KB SRAM in RTC Up to 8 MB Quad SPI flash
Power:	3.3 V operating voltage
WiFi	802.11 b/g/n Bit rate: 802.11n up to 150 Mbps
Bluetooth	BLE (Bluetooth Low Energy): Bluetooth 5 Speed: 125 Kbps, 500 Kbps, 1Mbps, 2Mbps. Advertising Extensions Multiple Advertisement sets

Table A.B: Arduino Nano ESP32

Specification	Details
Microprocessor	Xtensa® Dual-core 32-bit LX7 Microprocessor
Memory	384 kB ROM 512 kB SRAM 256 kB Flash Memory 32 kB SRAM 8 kB Data Memory (EEPROM)
Power	Operating voltage 3.3 V VBUS supplies 5 V via USB-C® connector VIN range is 6-21 V
Connectivity	Wi-Fi® Bluetooth® LE Built-in antenna 2.4 GHz transmitter/receiver Up to 150 Mbps

Table A.C: MyoWare 2.0 Muscle Sensors:

Specification	Details
Supply Voltage	min. = 2.27V, typical. = +3.3V or +5V, max. = +5.47V
Input Bias Current:	250 pA, max 1 nA
Input Impedance:	800
Common Mode Rejection Ratio (CMMR):	140 dB
Ideal Gain Equations:	Raw (RAW): G = 200 Rectified (RECT): G= 200 Envelope (ENV): G = 200 * R / 1 kOhm R is the resistance of the gain potentiometer in kOhm
Filters:	High-pass Filter: Active 1st order, fc = 20.8 Hz, 2-20dB Low-pass Filter: Active 1st order, fc = 498.4 Hz,2-20dB Envelope Detection: Linear, Passive 1st order, fc = 3.6 Hz,3 -20 dB Rectification Method: Full-wave
Outputs:	EMG Envelope (default), Raw EMG, Rectified EMG

Appendix B: Project Code:

Figure B.A.: sEMG Peripheral Device Code:

```
#include <ArduinoBLE.h>

BLEService sensorDataService("19b10000-e8f2-537e-4f6c-d104768a1214"); // BLE Service named
"sensorDataService"
// BLE Data Characteristic - custom 128-bit UUID, readable, writable and subscribable by
central
// Note, "BLENotify" is what makes it subscribable
BLEUnsignedLongCharacteristic dataCharacteristic("19b10001-e8f2-537e-4f6c-d104768a1214",
BLERead | BLEWrite | BLENotify);

void setup()
{
 Serial.begin(115200);
 // while (!Serial); // optionally wait for serial terminal to open
 Serial.println("MyoWare_MULTIPLE_BLUETOOTH");

 if (!BLE.begin()) { // begin initialization
  Serial.println("starting BLE failed!");
  while (1);
 }
 Serial.println("BLE initialized successfully");

 BLE.setLocalName("MYOWARESENSOR"); // set advertised local name
 BLE.setAdvertisedService(sensorDataService); // set advertised service UUID
 sensorDataService.addCharacteristic(dataCharacteristic);
 // add the characteristic to the service
 BLE.addService(sensorDataService); // add service
 dataCharacteristic.writeValue(0);
 // set the initial value for the characteristic
 BLE.advertise(); // start advertising

}

void loop()
{
 BLEDevice central = BLE.central(); // listen for BLE peripherals to connect

 if (central) // if a central is connected to peripheral
 {
  Serial.print("Connected to central: ");
  Serial.println(central.address()); // print the central's MAC address
```

```
Serial.println("Reading Sensors and writing BLE characteristic values now...");

while (central.connected())
{
 uint16_t sensorValue1 = analogRead(A0);
 // read the input on analog pin A0 / left leg/
 uint16_t sensorValue2 = analogRead(A1);
 // read the input on analog pin A1 / left arm
 uint16_t sensorValue3 = analogRead(A3);
 // read the input on analog pin A3 /right arm/
 uint16_t sensorValue4 = analogRead(A2);
 // read the input on analog pin A2 /right leg/

 uint8_t Val1Byte = map(sensorValue1, 0, 1023, 0, 255);
 uint8_t Val2Byte = map(sensorValue2, 0, 1023, 0, 255);
 uint8_t Val3Byte = map(sensorValue3, 0, 1023, 0, 255);
 uint8_t Val4Byte = map(sensorValue4, 0, 1023, 0, 255);

 uint32_t output = 0;    //output value 32 bit.

 //Sections where data is transcribed into u32bit integer to transmit
 output |= Val1Byte;
 // ---- ---- ---- ---- ---- ---- XXXX XXXX
 output |= (Val2Byte << 8);
 // ---- ---- ---- ---- XXXX XXXX ---- ----
 output |= (Val3Byte << 16);
 // ---- ---- XXXX XXXX ---- ---- ---- ----
 output |= (Val4Byte << 24);
 // XXXX XXXX ---- ---- ---- ---- ---- ----

 delay(10);
 dataCharacteristic.writeValue(output);

 //Test Prints:
 //Serial.print("\t");
 //Serial.print(sensorValue1); // print out the value
 //Serial.print("\t");
 //Serial.println(sensorValue2); // print out the value
 //Serial.print("\t");
 //Serial.print(sensorValue3); // print out the value
 //Serial.print("\t");
 //Serial.println(sensorValue4); // print out the value

}
```

```
  Serial.print(F("Disconnected from central: ")); // when the central disconnects, print it
out
  Serial.println(central.address());
 }
}
```

Figure B.B.: sEMG Central Device Code:

```
#include <ArduinoBLE.h>
#include "USB.h"
#include "USBHIDKeyboard.h"
USBHIDKeyboard Keyboard;
//Initializing Values:
uint8_t val1Byte = 0;
uint8_t val2Byte = 0;
uint8_t val3Byte = 0;
uint8_t val4Byte = 0;

uint16_t value1state = 0;
uint16_t value2state = 0;
uint16_t value3state = 0;
uint16_t value4state = 0;

int spacebarcheck = 0;

void setup() {
 // put your setup code here, to run once:
 Serial.begin(115200);
 Keyboard.begin(); //Begin keyboard.
 // while (!Serial); // optionally wait for serial terminal to open
 Serial.println("MyoWare Multi Sensor Example - BLE Central");

 if (!BLE.begin()) // initialize the BLE hardware
 {
  Serial.println("starting BLE failed!");
  while (1);
 }
 Serial.println("BLE initiallized successfully");
 BLE.scanForUuid("19b10000-e8f2-537e-4f6c-d104768a1214"); // start scanning for peripheral
with specified UUID
 Keyboard.begin();
}
void loop() {
 // Main code here, to run repeatedly:
 BLEDevice peripheral = BLE.available(); // check if a peripheral has been discovered

 if (peripheral) // discovered a peripheral, print out its info to Serial (For Bug Testing
Purposes)
 {
  Serial.print("Found ");
  Serial.print(peripheral.address());
```

```
  Serial.print(" '");
  Serial.print(peripheral.localName());
  Serial.print("' ");
  Serial.print(peripheral.advertisedServiceUuid());
  Serial.println();

  if (peripheral.localName() != "MYOWARESENSOR")
  {
   return;
  }

  BLE.stopScan();

  checkUpdate(peripheral);

  Serial.println("Starting to scan for new peripherals again...");
  BLE.scanForUuid("19b10000-e8f2-537e-4f6c-d104768a1214"); // peripheral disconnected, scan
again
  Serial.println("Scan has begun...");
 }
}

void checkUpdate(BLEDevice peripheral)
{
 Serial.println("Connecting ..."); // Connect to the peripheral notification

 if (peripheral.connect())
 {
  Serial.println("Connected");
 } else {
  Serial.println("Failed to connect!");
  return;
 }

 Serial.println("Discovering attributes ..."); // Discover peripheral attributes
 if (peripheral.discoverAttributes())
 {
  Serial.println("Attributes discovered");
 } else {
  Serial.println("Attribute discovery failed!");
  peripheral.disconnect();
  return;
 }

 // Retrieve the data characteristic
```

```
 BLECharacteristic dataCharacteristic =
peripheral.characteristic("19b10001-e8f2-537e-4f6c-d104768a1214");

 if (!dataCharacteristic)
 {
  Serial.println("Peripheral does not have that characteristic!");
  peripheral.disconnect();
  return;
 } else if (!dataCharacteristic.canWrite())
 {
  Serial.println("Peripheral does not have a writable characteristic!");
  peripheral.disconnect();
  return;
 } else if (!dataCharacteristic.canRead())
 {
  Serial.println("Peripheral does not have a readable characteristic!");
  peripheral.disconnect();
  return;
 } else if (!dataCharacteristic.canSubscribe())
 {
  Serial.println("Characteristic is not subscribable!");
  peripheral.disconnect();
  return;
 } else if (!dataCharacteristic.subscribe())
 {
  Serial.println("subscription failed!");
  peripheral.disconnect();
  return;
 }


 //Change Values here per Sensor:
 int delay_interval = 50;
 int delayValue = 1500;
 int delayValueUpper = 2000;

 //Sensor W/
 bool delay1 = false;
 int counter1 = 0;
 int delayCount1 = 0;
 int count1_signalLength = 500;
 int voltage_thresh_1_upper = 300;
 int voltage_thresh_1_upper_high = 600;
 int voltage_thresh_1_lower = 100;
```

```
 //Sensor A
 bool delay2 = false;
 int counter2 = 0;
 int delayCount2 = 0;
 int count2_signalLength = 500;
 int voltage_thresh_2_upper = 100;
 int voltage_thresh_2_lower = 50;

 //Sensor Spacebar
 bool delay3 = false;
 int counter3 = 0;
 int delayCount3 = 0;
 int count3_signalLength = 500;
 int voltage_thresh_3_upper_high = 300;
 int voltage_thresh_3_upper = 150;
 int voltage_thresh_3_lower = 100;

 //Sensor D
 bool delay4 = false;
 int counter4 = 0;
 int delayCount4 = 0;
 int count4_signalLength = 500;
 int voltage_thresh_4_upper = 100;
 int voltage_thresh_4_lower = 50;

 bool keyW = false;
 bool keyW_Alt = false;
 bool keyS = false;
 bool keyS_Space = false;
 bool keyA = false;
 bool keyD = false;

 bool keySpace = false;
 bool keyShift = false;


 while (peripheral.connected()) // while the peripheral is connected
 {
  if (dataCharacteristic.valueUpdated()) // Check to see if the value of the characteristic
has been updated
  {
   uint32_t received_val = 0;
   dataCharacteristic.readValue(received_val); // note, "readValue(uint32_t& value)" needs
the variable to be passed by reference
```

```
   // parse received_val - this contains all 4 of our ADC values (as each byte)
   //Serial.println(received_val, BIN); // optional print of the entire uint32_t for
debugging. Suppress other prints.
   val1Byte = (received_val & 0x000000FF);
   // read the input on analog pin A0 /left leg/ 'a'
   val2Byte = ((received_val & 0x0000FF00) >> 8);
   // read the input on analog pin A1 /left arm/ 'w'
   val3Byte = ((received_val & 0x00FF0000) >> 16);
   // read the input on analog pin A3 /right arm/ 'spacebar'
   val4Byte = ((received_val & 0xFF000000) >> 24);
   // read the input on analog pin A2 /right leg/ 'd'

   uint16_t sensorValueA = map(val1Byte, 0, 255, 0, 1023); // A
   uint16_t sensorValueW = map(val2Byte, 0, 255, 0, 1023); // - W
   uint16_t sensorValueS = map(val3Byte, 0, 255, 0, 1023); // - Spacebar
   uint16_t sensorValueD = map(val4Byte, 0, 255, 0, 1023); // - D


   // W
   if (delay1 == true){
    delayCount1 += delay_interval;
   }
   if (delayCount1 >= delayValueUpper){
     delay1 = false;
   }
   if (sensorValueW > voltage_thresh_1_upper){
    counter1 += delay_interval;
    if (counter1 >= count1_signalLength && delay1 == false){
     //Voltage Threshold Check:
     if (sensorValueW > voltage_thresh_1_upper_high){
      keyW_Alt = !keyW_Alt;
     }
     else {
      keyW = !keyW;
     }
     delay1 = true;
     delayCount1 = 0;
    }
   }
   if (sensorValueW <= voltage_thresh_1_lower){
    counter1 = 0;
   }

   // A
   if (delay2 == true){
```

```
 delayCount2 += delay_interval;
}
if (delayCount2 >= delayValue){
  delay2 = false;
}
if (sensorValueA > voltage_thresh_2_upper){
 counter2 += delay_interval;
 if (counter2 >= count2_signalLength && delay2 == false){
  keyA = !keyA;
  delay2 = true;
  delayCount2 = 0;
 }
}
if (sensorValueA <= voltage_thresh_2_lower){
 counter2 = 0;
}

// Spacebar
if (delay3 == true){
 delayCount3 += delay_interval;
}
if (delayCount3 >= delayValueUpper){
  delay3 = false;
}
if (sensorValueS > voltage_thresh_3_upper){
 counter3 += delay_interval;
 if (counter3 >= count3_signalLength && delay3 == false){
  if (sensorValueS > voltage_thresh_3_upper_high){
   keyS_Space = !keyS_Space;
  }
  else {
   keyS = !keyS;
  }

  delay3 = true;
  delayCount3 = 0;
 }
}
if (sensorValueS <= voltage_thresh_3_lower){
 counter3 = 0;
}

// D
if (delay4 == true){
 delayCount4 += delay_interval;
```

```
}
if (delayCount4 >= delayValue){
  delay4 = false;
}
if (sensorValueD > voltage_thresh_4_upper){
 counter4 += delay_interval;
 if (counter4 >= count4_signalLength && delay4 == false){
  keyD = !keyD;
  delay4 = true;
  delayCount4 = 0;
 }
}
if (sensorValueD <= voltage_thresh_4_lower){
 counter4 = 0;
}

//Change .press outputs for different keys.
//Logic (W/Space Outputs)

if (keyW == true){
 Keyboard.press('w');
}
else if (keyW_Alt == true){
 Keyboard.press('z');
 Keyboard.press('w');
}
else if (keyW == false || keyW_Alt == false){
 Keyboard.release('w');
 Keyboard.release('z');
 keyW = false;
 keyW_Alt = false;
}
if (keyS == true){
 Keyboard.press(' ');
 delay(1);
 keyS = false;
}
else if (keyS_Space == true){
 Keyboard.press(' ');
}
else if (keyS == false || keyS_Space == false){
 Keyboard.release(' ');
 keyS = false;
 keyS_Space = false;
}
```

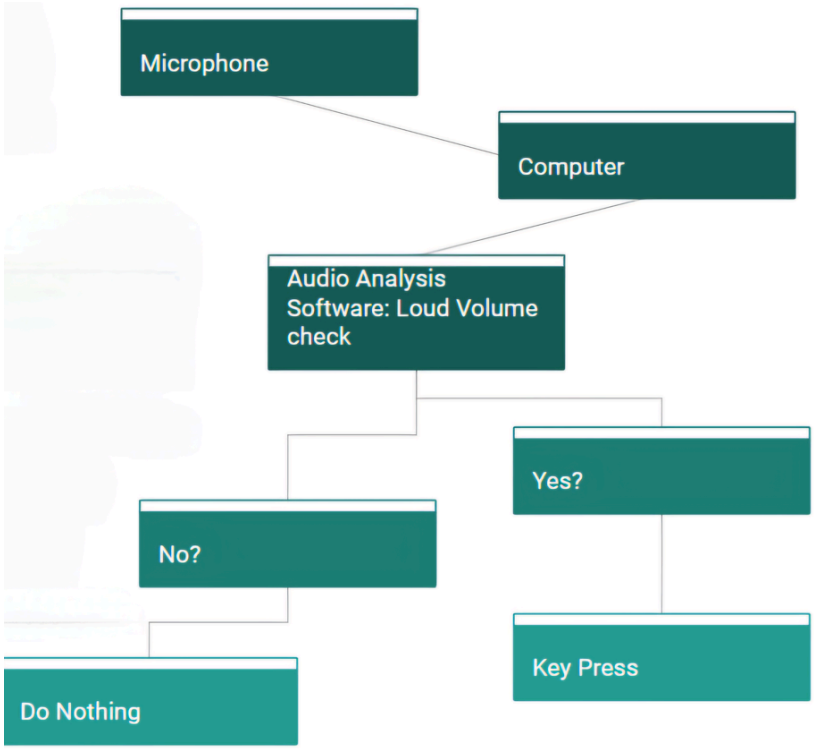
```
   // Space
   if (keyA == true){
    Keyboard.press('a');
   }
   else if (keyA == false){
    Keyboard.release('a');
   }
   if (keyD == true){
     Keyboard.press('d');
    }
   else if (keyD == false) {
    Keyboard.release('d');
   }

   // Serial Print Checks: Uncomment or Comment values you want to check for
calibration/testing purposes. Can output a single line of sEMG values if needed for Trial 2
   // You have to suppress key outputs to prevent accidental typing while testing.
   //Serial.print(keyA);
   // Serial.print("\t");
   // Serial.print(keyD);
   // Serial.print("\t");
   // Serial.print(keyW);
   // Serial.print("\t");
   // Serial.print(keyS);
   // Serial.print("\t");
   // Serial.println(keyShift);

   // Serial.print(spacebarcheck);
   // Serial.print("\t");
   Serial.print(sensorValueA);
   Serial.print("\t");
   Serial.print(sensorValueW);
   Serial.print("\t");
   Serial.print(sensorValueS);
   Serial.print("\t");
   Serial.println(sensorValueD);
   delay(delay_interval);
  }
  delay(1);
 }
 Serial.println("Peripheral disconnected");
 Keyboard.releaseAll();
}
```

Figure B.C. Voice Control Software

```
import matplotlib.pyplot as plt
import numpy as np
import pyaudio
from pyqtgraph.Qt import QtGui, QtCore
import pyqtgraph as pg
import struct
from scipy.fftpack import fft
import sys
import time
import pyautogui


class AudioStream(object):
    def __init__(self):

        # stream constants
        self.CHUNK = 1024 * 4
        self.FORMAT = pyaudio.paInt16
        self.CHANNELS = 1
        self.RATE = 44100
        self.pause = False

        # stream object
        self.p = pyaudio.PyAudio()
        self.stream = self.p.open(
            format=self.FORMAT,
            channels=self.CHANNELS,
            rate=self.RATE,
            input=True,
            output=True,
            frames_per_buffer=self.CHUNK,
        )
        self.init_plots()
        self.start_plot()

    def init_plots(self):

        # x variables for plotting
        x = np.arange(0, 2 * self.CHUNK, 2)
        xf = np.linspace(0, self.RATE, self.CHUNK)

        # create matplotlib figure and axes
        self.fig, (ax1, ax2) = plt.subplots(2, figsize=(15, 7))
        self.fig.canvas.mpl_connect('button_press_event', self.onClick)

        # create a line object with random data
        self.line, = ax1.plot(x, np.random.rand(self.CHUNK), '-', lw=2)

        # format waveform axes
        ax1.set_title('AUDIO WAVEFORM')
        ax1.set_xlabel('samples')
        ax1.set_ylabel('volume')
        ax1.set_ylim(0, 255)
        ax1.set_xlim(0, 2 * self.CHUNK)
        plt.setp(
            ax1, yticks=[0, 128, 255],
            xticks=[0, self.CHUNK, 2 * self.CHUNK],
        )
        plt.setp(ax2, yticks=[0, 1],)

        # format spectrum axes
        ax2.set_xlim(20, self.RATE / 2)

        # show axes
        thismanager = plt.get_current_fig_manager()
        thismanager.window.setGeometry(5, 120, 1910, 1070)
        plt.show(block=False)
```

```
    def start_plot(self):

        print('stream started')
        frame_count = 0
        start_time = time.time()

        while not self.pause:
            data = self.stream.read(self.CHUNK)
            data_int = struct.unpack(str(2 * self.CHUNK) + 'B', data)
            data_np = np.array(data_int, dtype='b')[::2] + 128

            self.line.set_ydata(data_np)
            if (np.max(data_np)) > 250:
                print('loud: a', np.max(data_np))
                pyautogui.press('space')
            else:
                print('normal: b', np.max(data_np))


        else:
            self.fr = frame_count / (time.time() - start_time)
            print('average frame rate = {:.0f} FPS'.format(self.fr))
            self.exit_app()



    def exit_app(self):
        print('stream closed')
        self.p.close(self.stream)

    def onClick(self, event):
        self.pause = True


if __name__ == '__main__':
    AudioStream()
```

Figure B.D Voice Control Software Flowchart



Appendix C: Additional Results:

Figure C.A: Threshold Calibration Left Bicep Trial 1

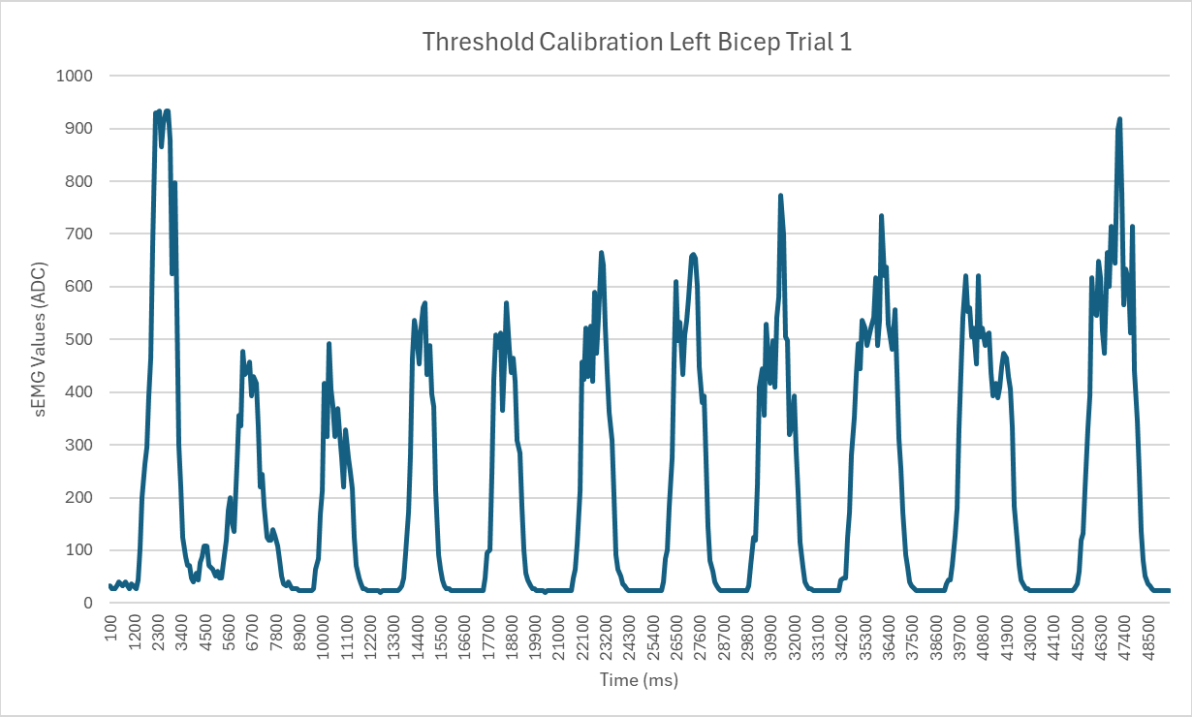


Figure C.B: Threshold Calibration Left Bicep Trial 2

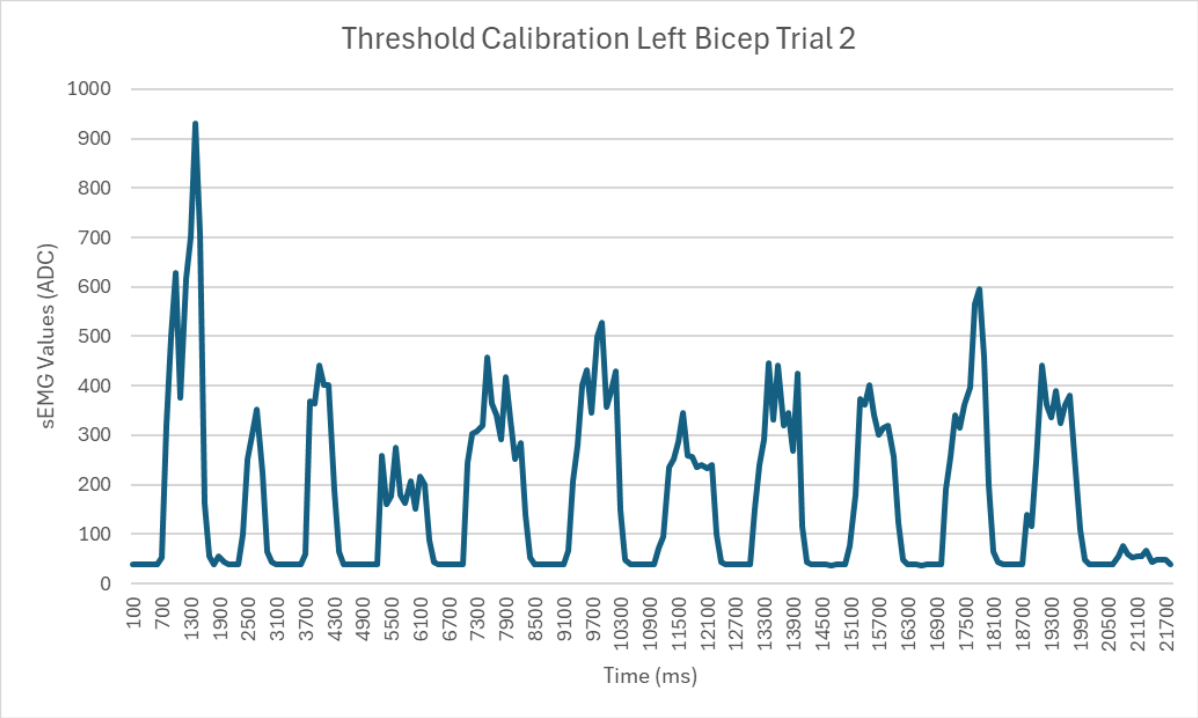


Figure C.C: Threshold Calibration Left Bicep Trial 3

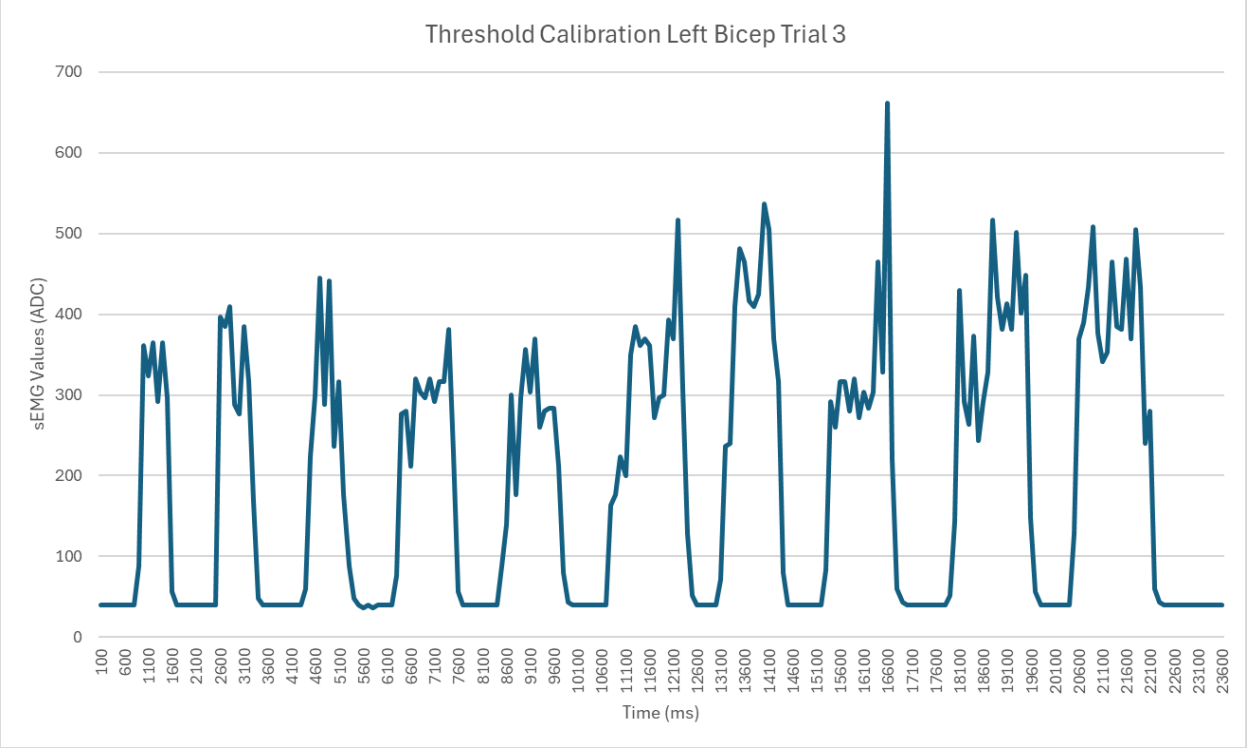


Figure C.D: Threshold Calibration Right Bicep Trial 1

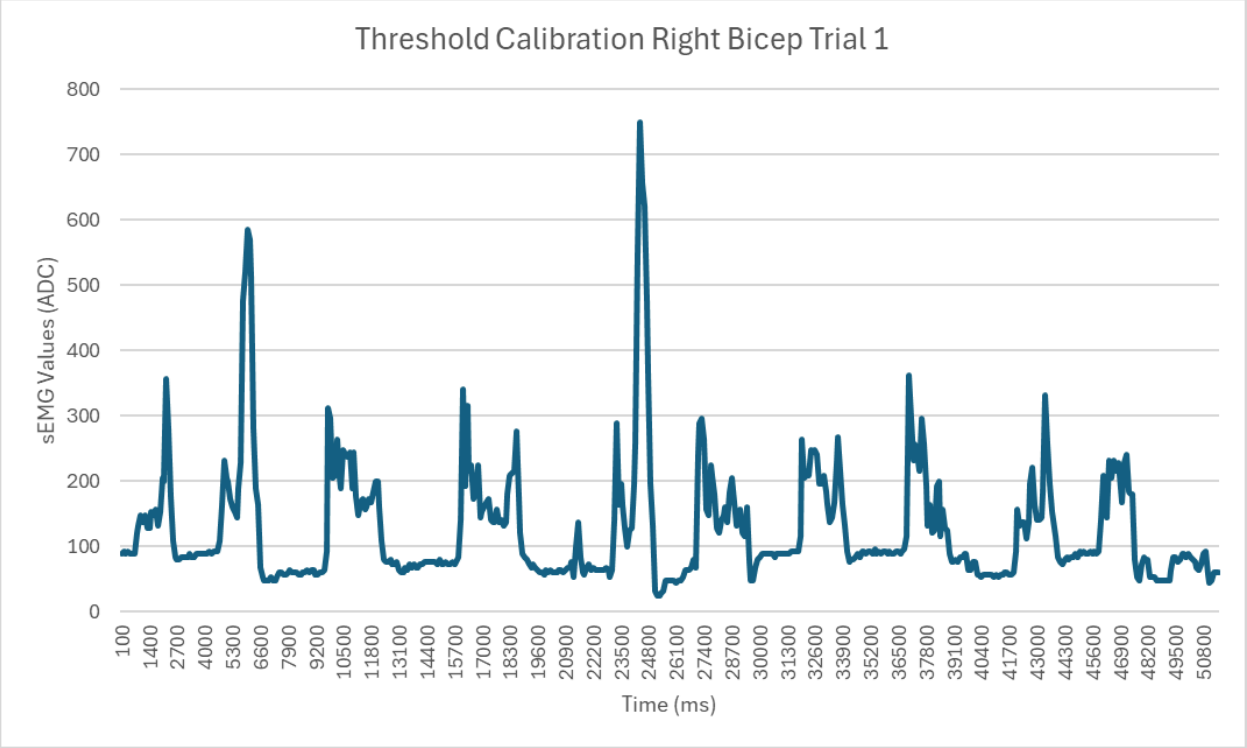


Figure C.E: Threshold Calibration Right Bicep Trial 2

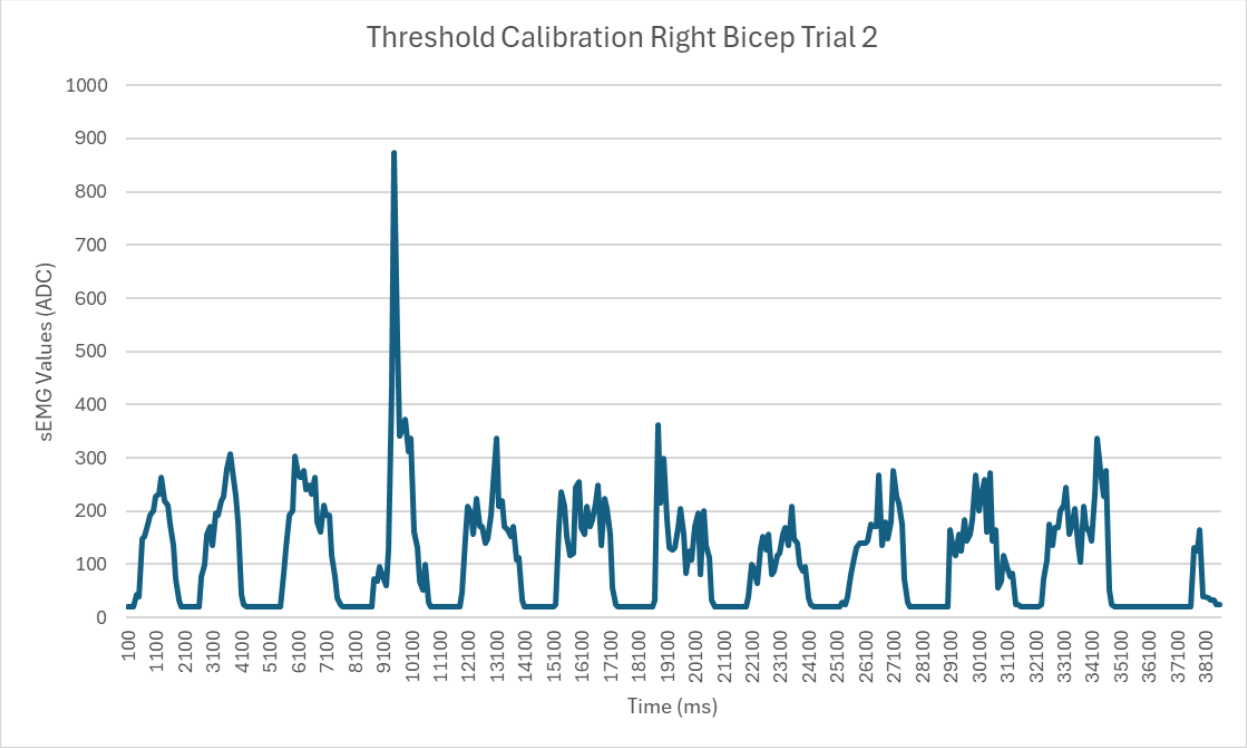


Figure C.F: Threshold Calibration Right Bicep Trial 3

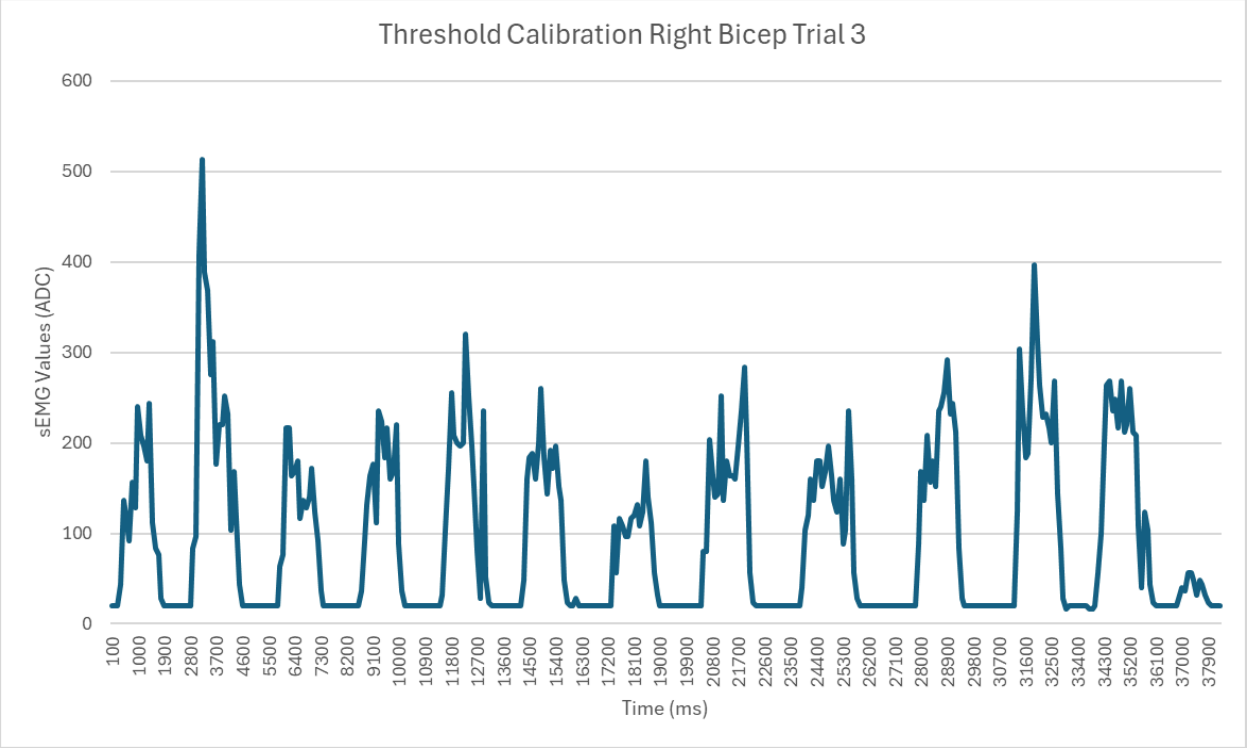


Figure C.G: Threshold Calibration Left Gastrocnemius Trial 1

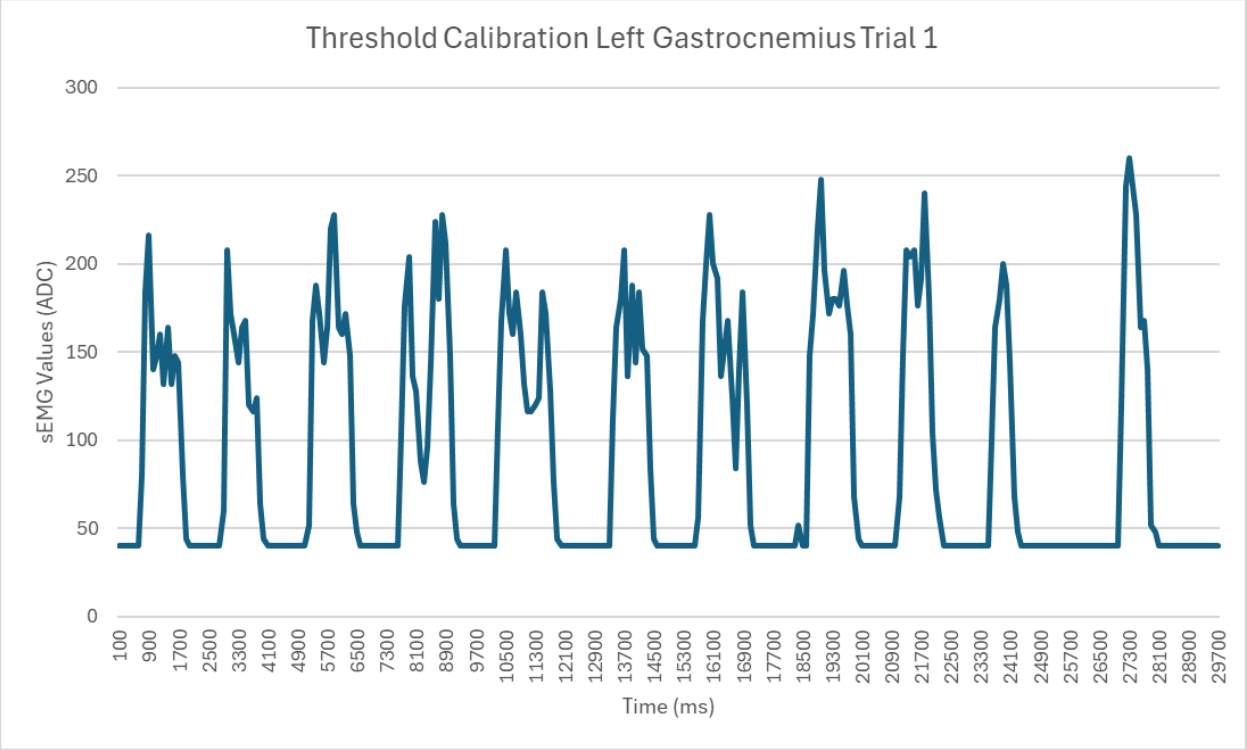


Figure C.H: Threshold Calibration Left Gastrocnemius Trial 2

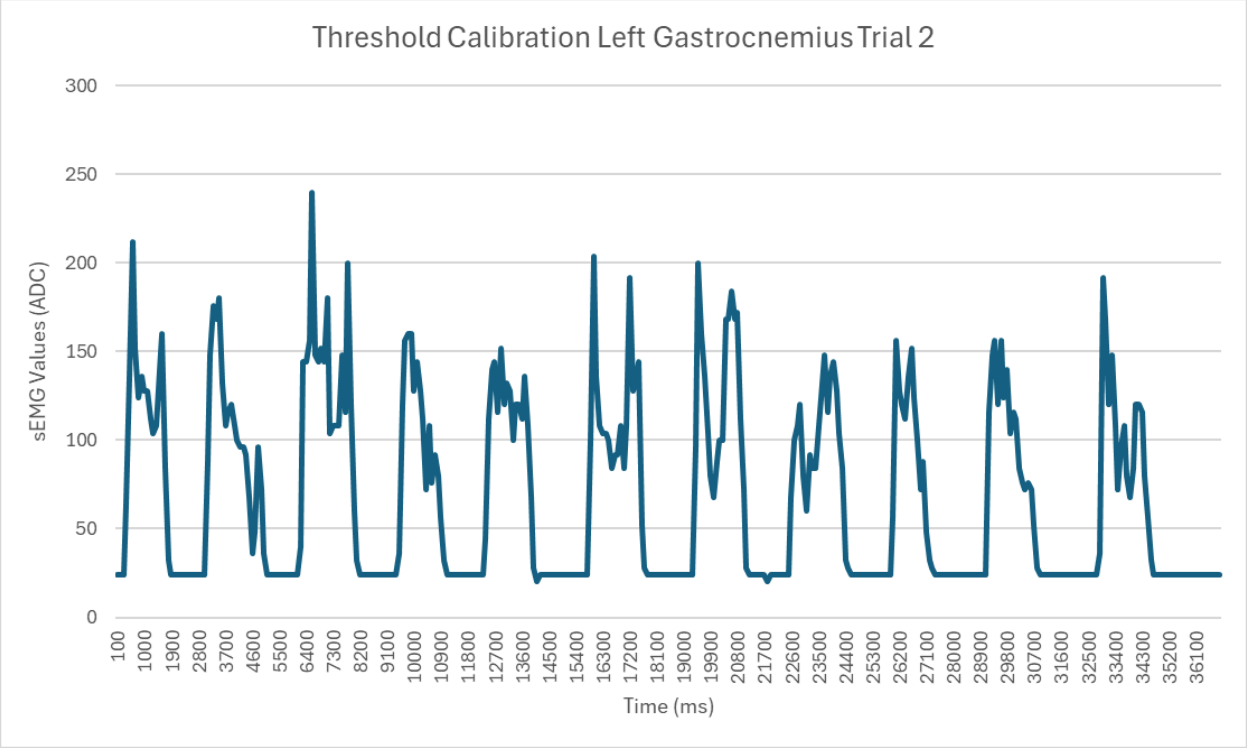


Figure C.I: Threshold Calibration Left Gastrocnemius Trial 3

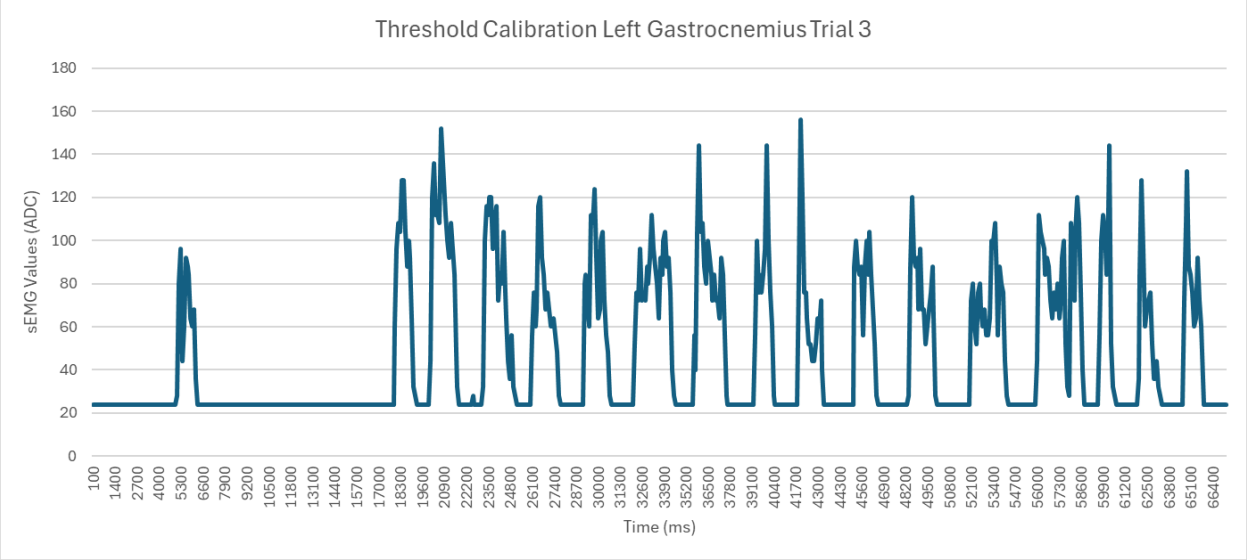


Figure C.J: Threshold Calibration Right Gastrocnemius Trial 1

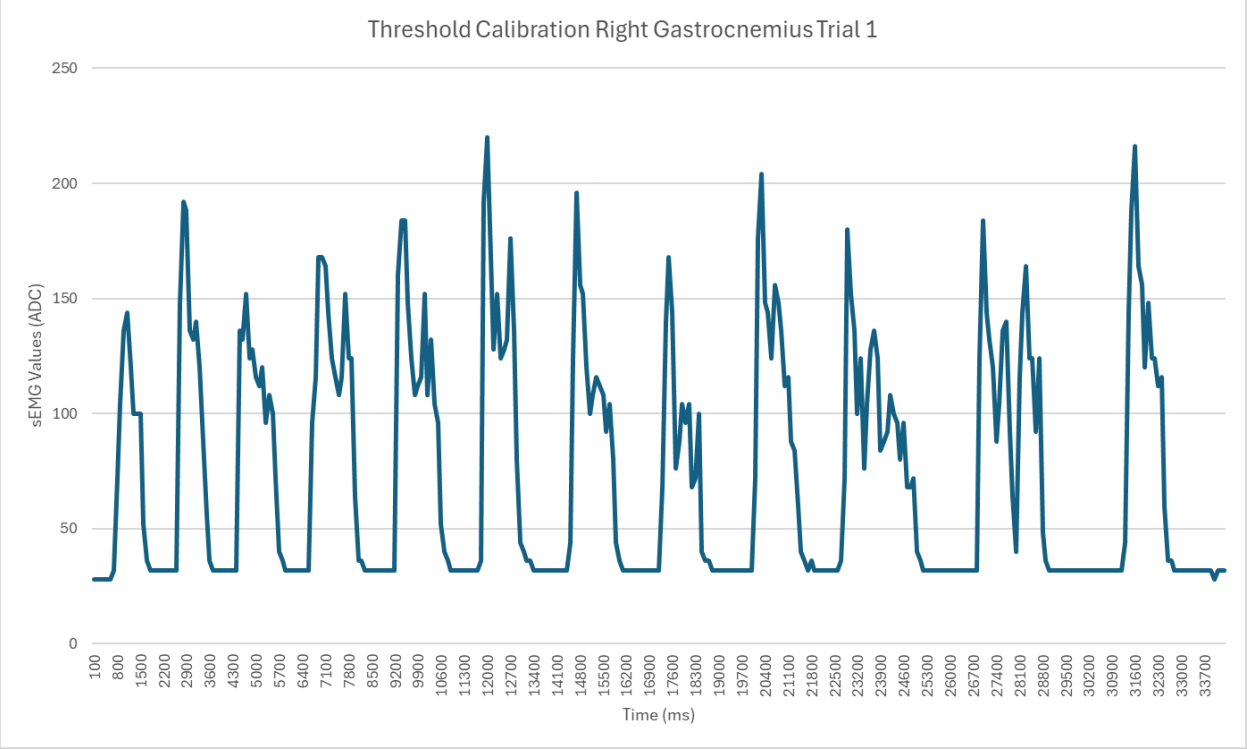


Figure C.J: Threshold Calibration Right Gastrocnemius Trial 2

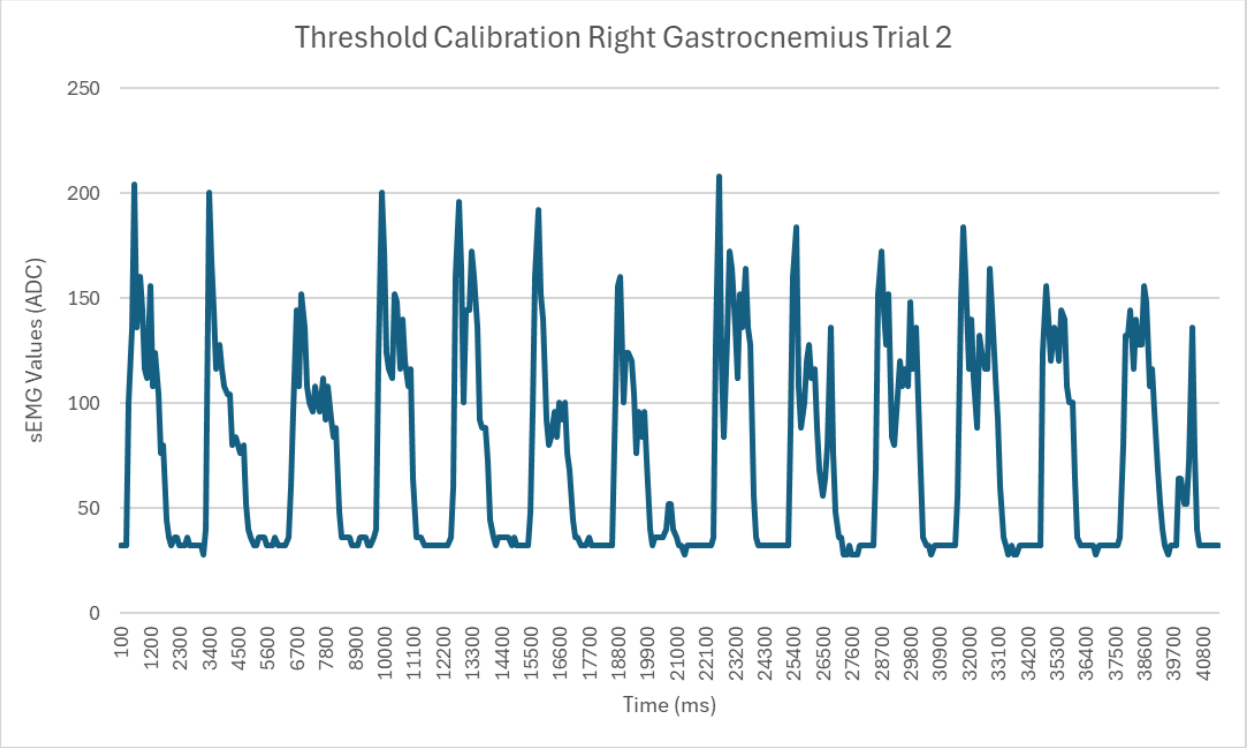


Figure C.H: Threshold Calibration Right Gastrocnemius Trial 3