MyoRun: Real Time EMG

Design Review Report

Authors: Alex Hong, Tarana Laroia, Kayla Vokt:
Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—**

THE objective of our capstone is to create a system capable of controlling a video game environment using only the electrical signal from forearm muscle action. We decided to create a new form of controlling user movement in a videogame setting using an EMG system with five electrodes arranged on specific muscles on the forearm. The collection of these signals were classified into left, right, up and down movements that function as inputs to the control of the user in an endless-runner style video game. This novel form of video game control creates an exciting experience for the user.

***Index Terms*—**
Arduino
Electromyography (EMG)
Muscle Sensor, Signal to Noise Ratio (SNR)
Unity
User Datagram Protocol (UDP) Streaming

I. Introduction

Decoding and extracting the information contained in bioelectrical signals is a continually studied area of research. Electromyography, or EMG, technology allows us to detect signals generated by skeletal muscles from the surface of the skin. One of the initial areas of application of EMG signals was driving input for the control of powered upper limb prostheses, which came to be known as myoelectric control [1]. Many common computer and videogames today are played with an external controller such as a keyboard or game controller.

For our project, we were interested in applying myoelectric control to the novel area of video game control. This application area is important because it has the potential for broader impact for people with impaired movement skills, as well as has the potential for impact in rehabilitation of motor impairment following stroke [2]. We developed a hardware EMG system using 5 electrode pairs on distinct muscles in the forearm. EMG signals collected from trials of different wrist and forearm movements were used to train and test a machine learning classification model in order to identify hand movements as inputs to the virtual game. Designing a novel game controlling technique interfacing with the human body meant that our system was susceptible to the intricacies of the human body system. To combat this, we collected training data from various subjects in order to make the model robust enough to work for different people.

The software portion of the game had to be developed in parallel, thus in order to test the system independently without the muscle guided signal inputs, a Mock Muscle Controller was designed to simulate game inputs and test the functionality of the frontend of the project.

Our final capstone goal was to develop a surface EMG system for the forearm that controls user movement in a virtual game environment. Through simultaneous game development and machine learning algorithm modification, we created a robust myoelectric control video game. We hoped to meet the specifications of keeping the EMG system impedance under 20kΩ, while meeting high classification accuracy of over 80% and meeting low classification and game latency, keeping the game delay under 500 ms.

II. Design Requirements

Project requirements can be broken down to requirements for the Muscle hardware, Signal Processing, and Game Software.

The first hardware standard requirement we would like to highlight is keeping the impedance of the EMG system under 20kΩ. It is the standard in medicine for impedances to be under 20kΩ for electronic devices, so this was an important metric we set for ourselves. We will measure the impedance of the actual hardware itself, without any electrical input, using the impedance function of a multimeter. In the event that we need to decrease our electrode's impedance, we will employ techniques commonly practiced in EMG labs, including cleaning and abrading the skin, and applying conductive electrolytic paste as needed. This requirement is important to ensure a better SNR, because cleaner signals are easier to process and classify. This metric is in place to ensure classification accuracy.

Second, we require the classification accuracy to be greater than or equal to 80%. This means that four out of every five muscle movements we make will be classified correctly. This will be measured by repetitively testing all possible inputs, and keep track of the output from the classification algorithm. This is an important metric to meet for this project, so we will considerably iterate on the algorithm as it develops through the design process. If we struggle to meet our objective, we will optimize our algorithm and record more training data from diverse samples in order to make the algorithm more robust. If needed, we will add more electrodes into our system because having more features will give our model more information to train on, leading to an improved classification accuracy.

In addition, it is important to reduce the classification delay as much as possible, specifically we expect the delay from utilizing the classification model we develop to fit into the total system delay to be under 250ms. Given that the game delay should not be significantly large, the classification delay should reduce as our model reaches a higher classification accuracy. Collecting and incorporating as much training data as possible into our model will ensure the accuracy and reduce the delay experienced through using our system.

Next we require the Game Delay to be less than or equal to 250 ms. Game Delay is defined to be the time between when messages get sent off through the UDP channel, and when the change in the character's state in the game is visible. Human response time for a negligible latency is known to be 250ms, so this will be our requirement. We have developed the Mock Muscle Controller that will be used to send sample input data

over to Unity. This will be used for a measurement independent of the Hardware, allowing for a specific and accurate measurement of the Software side of performance.

The final metric is the Total System Delay to be less than or equal to 500ms. Total System Delay is the latency of the entire system, which is the time between when a muscle is fired and when the change in the character's state in the game is visible. From the diagram in Fig. 1, when a muscle is fired, the signals are converted to digital inputs, which then goes to ML and DSP models, to the UDP Channel, and finally to the Unity game. The Total System Delay is measuring the entire latency, which ultimately affects the playability and performance of the project.

III. Architecture and/or Principle of Operation

The overall system is a one-way stream from Muscle Activity to Game Action. When the user moves muscles where N electrodes are placed, the Muscle Sensor reads analog EMG readings from the N electrodes and gets amplified and converted into digital signal in Arduino. The raw digital signal of N channels is sent over to Python, and Wavelet Transform, PCA, STFT methods are used to extract features from the signals. Classification Algorithms such as Support Vector Machine (SVM) are used to classify the feature signals into Flexion, Extension, Pronation, Supination. The classification results in bytes are then sent over to Unity in UDP protocol, which is then used to trigger action on the game on active edge.

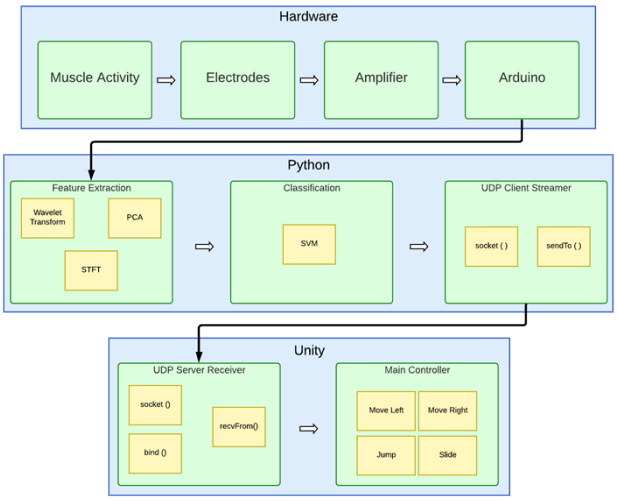


Fig. 1. Overall System Design

The software has an inherent dependency with the hardware because the system is a constant one-directional stream from the hardware to software. Without a hardware side to provide inputs to control the game, the game is unable to be played. While the project's final goal is to get both sides functional, Mock Muscle Controller is developed to test for Software functionality.

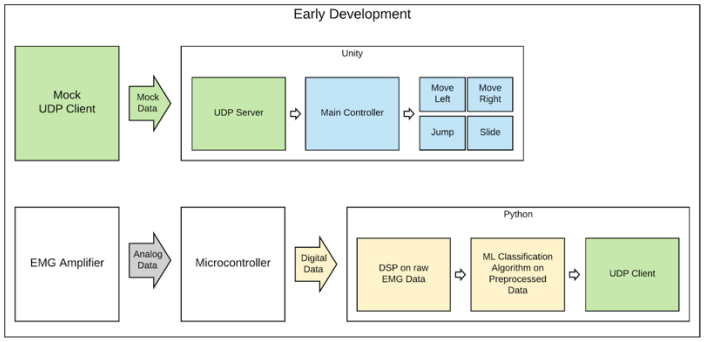


Fig 2. Diagram of Mock Muscle Controller

The Mock Muscle Controller simulates real Classification Results by sending off sequences of bytes that we expect to see from the real classifier. The controller application, written with the pygame library, listens for keyboard inputs - left, right, up, and down arrows - each arrow keys represents the classification result. For example, Up arrow maps to the class Pronation, which maps to byte (0x03).

The Mock Muscle Controller is useful because it allows for parallel development between the Software and the Signals, allowing the team to make progress without having to wait for one another. Another benefit is that such blackbox testing allows the 2 development lines, Software and Signals, to be made in tandem without relying on one another. One can constantly make progress and proceed without having to wait for another. Such separation help the team test and integrate easily. Another benefit of the Mock Muscle Controller is that it can test metrics independent of the Hardware side, and test specifically for the latency in the UDP and the game only.

IV. Design Trade Studies and System Description

A. Hardware Subsystem

The first part of our system begins when the user performs a muscle movement. The activity in their muscles will be read and recorded by the array of muscle sensors on their arms. These signals are read into Arduino IDE using the ADCs on the Arduino Nano microcontroller. From there, they are sent directly into python to be analyzed.

After the signal data has been sent to the python, it will be preprocessed via the Wavelet Transform then the STFT to change the basis of the signal and extract its features. From there, these extracted features will be used as inputs into the SVM algorithm, where they will be classified as a muscle movement mapping to a game control.

Electrodes

A key component of our EMG system is the electrodes we use. For our project, we chose to use the Covidien EMG/EKG pads. Like many electrode pads, they are self-adhesive and come with silver/silver chloride tabs built into them, providing easy integration with our snap electrode wires and amplifier. Unlike most conventional self-adhesive electrode pads, these come with electrolytic gel designed into the electrode itself. This will allow for greater conductivity on the surface of the skin, which will reduce the impedance of our system and lead

to a higher SNR, which will make our signals clear and our classifications more accurate. Without the electrolytic gel, the spatial frequencies between the two electrodes would be more difficult to measure.

Sensors

For our project, we employed the use of the MyoWare Muscle Sensor [3]. This is an embedded EMG system that measures the activity at a single muscle in the body. It is made up of two electrodes, about 2 cm apart, which record the resistance and spatial frequencies between them. This is how muscle activity is measured. The entire system references a ground electrode, meant to be separated from the muscle in question. This allows all noise that the system experiences to be filtered out using the ground electrode as a reference to the rest of the system. It also has a builtin amplifier, which is critical in a system sensitive to noise. Amplifying the signal as close to the source as possible will lead to a higher SNR, which is important for having a high classification accuracy. Letting the signal travel any amount means exposing it to more and more power deterioration and noise, which will result in a weaker, noisier signal. We took care to choose a system with amplifiers patches right at the signal source, taking full advantage of the signal strength. Furthermore, the MyoWare Muscle Sensors are individual embedded systems, which gives us the adaptability to easily shift our electrode placements and make our system modular. We chose to have the ability to place a small number of electrodes in a few key locations. An alternative design we considered was an EMG sleeve, which would put a higher density of electrodes in one place, and have a single amplifier for all of the multiplexed signals. However, this would limit us to working with a single muscle or group of muscles. It would also not have the benefit of amplifying directly at the signal source (albeit fairly close) and would not offer us the modularity of picking what muscles we wanted to use.

Microcontroller

For our project, we needed a microcontroller capable of reading the analog signals from our muscle sensors. We considered a variety of Microcontroller options, including the Raspberry Pi, SparkFun RedBoard, Adafruit Feather M0, and the Arduino Uno/Mega/MKRZero. Ultimately, we decided on using the Arduino Nano. It is capable of reading up to eight analog signals at once, each with an ADC sampling rate of 9.6 kHz, making it ideal for our EMG electrode system. It is also small and headerless, making it easily portable with the flexibility to be made into a wearable device. It has a 5V power rating, which also makes it compatible with our muscle sensors. The Arduino specifically comes with Arduino IDE, and easy support for coding the system. As hardware is not the main focus of our project, we wanted a reliable platform that would allow us to focus on the software and signals aspects of our work; therefore, we chose to use Arduino over other platforms, as it allowed us the flexibility to easily set up our hardware and dedicate our time to collecting and analyzing our data.

B. Data Collection & Classification Subsystem

Electrode Placements

Another critical design decision that we made for this project was where on the forearm we intended to place the electrode pair components for the EMG system. As an initial idea, we intended to use an array of eight electrode pairs along the forearm, beginning 2-3 centimeters from the elbow and extending down to a few inches above the wrist. They would be arranged in a circular formation around the forearm. As gleaned from previous EMG experiment research papers, with a recommended interelectrode difference of 20mm as to avoid crosstalk and enhance SNR [4]. After reviewing additional research papers to discover the best electrode configuration for our use case, we decided to alter our design to utilize five electrode pairs instead of eight. This was motivated by collecting fewer extraneous muscular signals, given the four specific wrist/forearm movements we selected to use to control our game. We decided to place the five electrodes directly on the muscles that are activated during these specific movements, including the Brachioradialis, Flexor Carpi Radialis, Flexor Carpi Ulnaris, Extensor Carpi Ulnaris, Extensor Digitorum, as shown in Fig. 3.

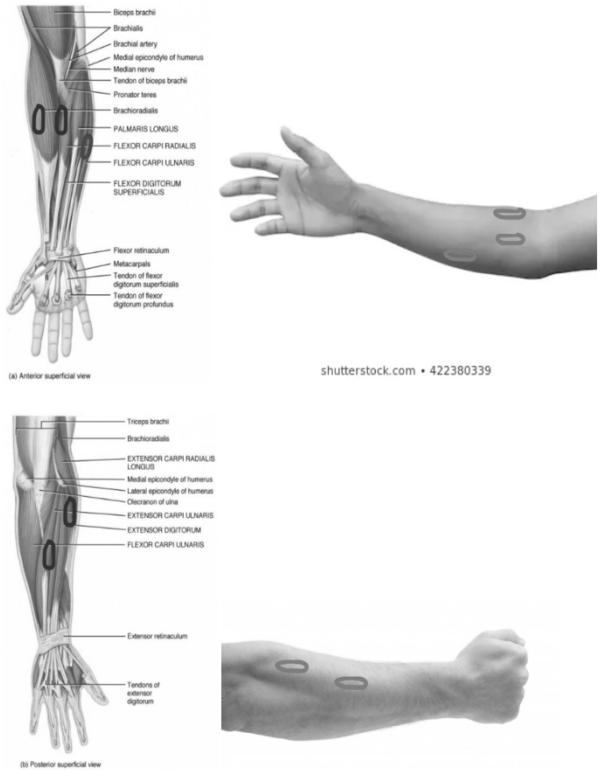


Fig. 3. This figure showcases the location of the five muscles in the forearm that will be recorded as well as the location of where the electrodes will be placed on the surface of the

Feature Extraction

The best features to classify the EMG signals from the forearm were selected through reading through existing literature about EMG classification. The best features that we selected to incorporate were Principal Component Analysis

(PCA) for dimensionality reduction, Wavelet Transform (WT), and Short Time Fourier Transform (STFT) [5]. Other feature extraction methods that were considered were root-mean-square, time-domain features such as mean absolute value and number of zero crossings, however it was determined that the features that we selected, particularly WT, yielded the best classification accuracy [6].

Classification

For our classification algorithm, we wanted a machine learning model that was well suited to our project and resources. We considered a variety of models, including K Nearest Neighbors (KNN), Perceptron, Neural Networks, and Hidden Markov Models (HMMs). Ultimately, we decided on using a Support Vector Machine (SVM). This model in particular is suited to classification (not regression) and works very well even on small amounts of training data. Because of COVID-19, our access to participants to collect a diverse sample of training data has been extremely limited, so we wanted a model that would perform well without having to be trained on thousands of samples to perform well. Because of this restriction, we ruled out using a Neural Network, which is only as good as the training data and can only recognize things it has seen before. SVMs are also a form of unsupervised learning, which work better for signal processing applications such as ours, as it will be capable of doing its own feature extraction. This is another area where SVMs will outperform Neural Networks for our application. Furthermore, SVMs can work on data that is not linearly separable, making them adaptable to a wider variety of situations, especially since our classes share distinct features and muscle patterns. For this reason, this model is easy to adapt into higher dimensional space, unlike HMMs. We will also be primarily analyzing signals in the Fourier basis, making the sequential nature of HMMs not as applicable to us. Another benefit of SVMs is that they do not weigh all features equally, as we know that some features we extract will be more meaningful than others. Particularly, some muscle movements will be more meaningful for certain classifications. A drawback of both Perceptron and KNN is that they provide the same weight to all features, and SVMs are more flexible in that regard.

C. Software Subsystem

Unity System

The Unity game triggers actions from a data stream from UDP. A separate thread receives the data and processes the active edge trigger, to change a state variable. Then, the main thread does actions based on the changed states. Keeping the main thread separate from the UDP Socket is important in keeping the game running smoothly and fast.

Creating new objects in the virtual world and deleting them are expensive processes in Unity because of the rendering, meshes, object colliders and other background mechanisms that need to be used and allocated. Thus, reusing already created objects as much as possible is the preferred practice. This is the approach used to generate roads for the endless Myo-Run game. The map keeps a queued array of road segments of fixed size, and when the player moves past one segment, the segment is popped and pushed back in. This way the roads are being reused, and the only objects that need to be re-created or re-rendered are the obstacles and other feature objects. But these also can be reused to optimize game run-time.

UDP vs. TCP/IP

UDP is used as the communication protocol between Python and Unity to deliver classification results because we need fast delivery. This is at the cost of unreliable delivery compared with TCP/IP, but we are constantly streaming classification results to the game, so there is not as much of a need for reliable delivery.

Modified UDP Client-Server Model

In a typical UDP model, the Client request information and the Server responds back with information. Following this mechanism our initial plan was to have Unity request for data from the Muscle Streamer, so Unity would be the Client, and Muscle Streamer would be the Server. However the problem with this is that before the Streamer can stream anything it needs to wait until the Client sends something. This is unnecessary because we just want the Streamer to stream information to the designated port.

This gets even more unnecessary when either one of the Unity or Muscle Streamer get disconnected. One option is to have Unity constantly request data and wait for response for every transaction. Another option is to have Muscle Streamer start streaming data after the first request, and have a timer for Unity to know whether Unity is disconnected and request data again. Both of the methods are inefficient and they revolve around having Unity being the Client and Muscle Streamer being the Server.

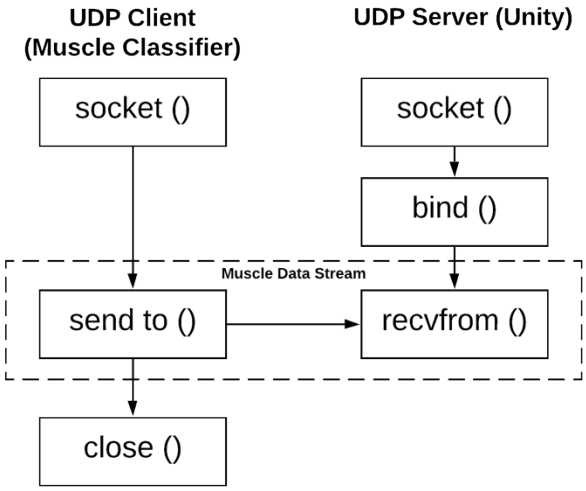


Fig. 4. Modified UDP Streamer Model

Instead our solution is to have the Muscle Streamer to be the Client and Unity to be the Server. Shown in Figure #, Muscle Streamer can just send data to a given port information, without knowing whether the other side has received it. Meanwhile Unity just listens for any incoming messages, receives data and

does not respond back. This cuts down the number of messages by half. If we send N sequences of bytes, we send exactly N packets over, making the time complexity to the bare minimum.

V. Project Management

A. Schedule

The schedule for our project consisted of 4 main phases. The detailed schedule can be found at the end of the report.

Phase 1: Project Proposal and Planning

During this phase, we created our project proposal and arranged our Gantt chart and plan of action for the entire semester project. We ordered one initial MyoWare electrode in order to test it and verify that it would work for our intended project, before comitting to using it for our entire design. Additionally, we conducted a literature review on current state-of-the-art surface EMG classification methods and existing myoelectric controlled video games.

Phase 2: Design and Implementation

This phase consisted of the bulk of our project. The major milestones we completed are outlined below.

Milestone #1: Proof of Concept / Hardware Prep

In this milestone, we worked with a single EMG electrode in order to have a proof of concept test for the pipeline of our project. We streamed the data from a single electrode into an Arduino and sent it over to python in order to prepare for the feature extraction and classification part of the project.

Milestone #2: Basic Integration

With the software development of the game well underway, the Mock Muscle Controller was developed to help us test the game independent of the muscle inputs. The basic integration consists of classifying a binary EMG signal and integrating this with the game.

Milestone #3: Full Implementation

The final milestone for phase 2 consists of developing the classification model from the data collected using the EMG system with five electrodes. The output of the signal is to be integrated with the fully developed game.

Phase 3: Performance Testing and Integration

During this phase, we verify the functionality and accuracy of our system, looking back at our metrics to see how closely we met the requirements we set out for ourselves. The classification accuracy and game delay are tested during this phase. At the end of this phase is the Thanksgiving break, which we took care to plan our schedules and availability around. Following this, we want to ensure that all physical/hardware aspects of our project are completed, as we will be in remote instruction during that time.

Phase 4: Final Report

The final phase of our project is to compile all the results and reflect on the outcome of our project. During this phase, we create a final project demo video as well as write our final report.

B. Budget

TABLE I. Major Components of Bill Of Materials

Product	Amount	Price	Shipping	Total Price Per Part	Vendor
MyoWare Muscle Sensor	5	$34	$0.00	$179	Adafruit
Arduino Nano (w/o headers)	1	$21.55	$7.99	$29.54	Digikey
USB Separator	1	$34.95	$0	$34.95	Adadfruit
Covidien EMG Electrodes	1	$18.99	$7.85	$26.84	Covidien
Misc. (wires, tools, etc)	1	$10.00	$0	$10.00	Digikey
Unity Poly Stickman Pack	1	$9.90	$0	$9.90	Stegobubbles
			TOTAL	$290	
			Budget	$600	
			Remaining	$310	

For our hardware implementation, we use five individual MyoWare Muscle Sensors to detect muscle activity, several electrodes to connect the sensors to a participant's skin, an Arduino to intepret the analog signals of said activity, a USB separator to isolate the participant from a computer and protect them from being electrically shocked, and a miscellaneuos assortment of tools and wires to connect everything and put our system together.

Moveover, we also used part of our budget to purchase a Unity asset of predesigned characters, with certain animations for given actions. We will use these to enhance the graphics experience for the user and make the game more playable.

C. Risk Management

Something important to our project's success is the delay between the user's actions and when the game processes them. This is our total delay, which we have broken down into two distinct parts: classification delay and game delay. The classification delay includes the time that it takes for the signal to be read into the Arduino, sent to python, preprocessed, and classified. The game delay includes the amount of time it takes for the game to receive an input signal and act on it. For our project, we want our total delay to be no more than 500 ms, with no more than 250 ms for both the classification delay and the game delay. However, we have measures to address not being able to meet either of these goals. If our classification delay is too large, we will work to improve our algorithm to make it more efficient. One of our main approaches will be to downsample our data. Because the frequencies being recorded are of human muscle activity, they are quite low, and therefore the Nyquist frequency is also low. This gives us the freedom to downsample from the Arduino's much faster sampling rate of 9.6 kHz. If our game delay is too large, then we will apply this idea of downsampling to our approach of the game. We can reduce the framerate, which will mean the game will not reload itself as frequently, but will have more processing power available. We can also work to optimize the code on the game end, and simplify the procedural generation and complex aspects of the game environment.

An important part to ensuring good signal quality is having low impedance on the electrodes [7]. A lower impedance will lead to stronger signals and a better SNR, which will allow for a higher classification accuracy. If our electrode impedance is too high, then we will address this by employing a variety of tactics common in medical device practices. These include introducing electrolytic gel or conductive paste under the skin electrodes [8], exfoliating and abrading the skin to remove dirt and dead skin cells impeding the signal [9] , and removing body hair impeding the signal. If all of these tactics fail to bring us under our desired 20kΩ impedance; however, we are still able to achieve our desired classification accuracy, then we will have reached an impedance low enough to make our project work.

A critical part of the game experience for the user is the classification accuracy of our algorithm. If we are unable to achieve our desired classification accuracy of 80%, then we have measures in place to aid us into improving that. Firstly, we will record more data, as introducing our machine learning model to as much training data as possible will improve its performance. We will particularly record more training data on a variety of people, so that we can introduce a diverse population set into our game to make it more adaptable and diverse. This way, any participant would be able to play, regardless of whether or not they were used to train the classification algorithm they will be testing. Another approach to improving classification accuracy will be introducing more electrodes, particularly on different muscles on the surface of the arm. Adding another muscle into our system will create another class of features to extract, such that we can make our algorithm more precise in its ability to pattern match. More electrodes will allow for a broader range of feature extraction.

VI. Related Work

While there are not any specific myoelectric control games available on the market, there is the Myo Armband developed by Thalmic Labs which has been used to interface with myoelectric control projects, or work as a handsfree presentation remote, which is currently sold on Amazon [9].

References

[1] Moritani, T., Stegeman, D. and Merletti, R. (2005). Basic Physiology and Biophysics of EMG Signal Generation. In Electromyography (eds R. Merletti and P. Parker). doi:10.1002/0471678384.ch1

[2] Klein, Cliff S et al. “Editorial: Electromyography (EMG) Techniques for the Assessment and Rehabilitation of Motor Impairment Following Stroke.” Frontiers in neurology vol. 9 1122. 18 Dec. 2018, doi:10.3389/fneur.2018.01122

[3] Hermens, H., B. Freriks, R. Merletti, D. Stegeman, J. Blok, G. Rau, C. Disselhorst-Klug, and G. Hägg, European recommendations for surface electromyography, Roessingh Research and Development, Enschede, Netherlands, 1999.

[4] Andrews, A, E Morin, and L McLean. “Optimal Electrode Configurations for Finger Movement Classification Using EMG.” 2009 Annual International Conference of the IEEE Engineering in Medicine and Biology Society 2009 (2009): 2987–2990. Web.

[5] Y. Geng, L. Yu, M. You and G. Li, "A Pilot Study of EMG Pattern Based Classification of Arm Functional Movements," 2010 Second WRI Global Congress on Intelligent Systems, Wuhan, 2010, pp. 317-320, doi: 10.1109/GCIS.2010.125.

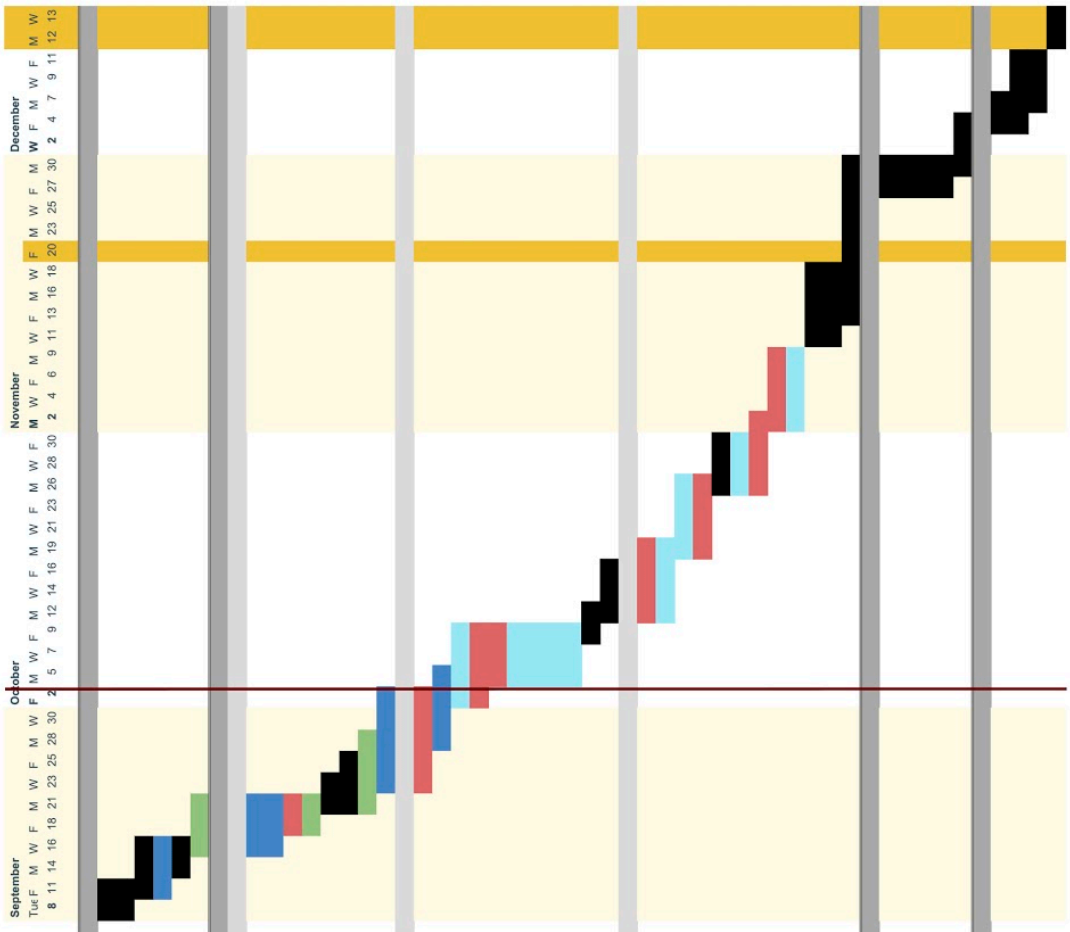
[6] Nunez, P., & Srinivasan, R. (2006-01-26). Electric Fields of the Brain: The neurophysics of EEG. : Oxford University Press. Retrieved 19 Oct. 2020, from https://oxford.universitypressscholarship.com/view/10.1093/acprof:oso/9780195050387.001.0001/acprof-9780195050387.

[7] Brigham, Danielle et al. “Comparison of artifacts between paste and collodion method of electrode application in pediatric EEG.” Clinical neurophysiology practice vol. 5 12-15. 30 Nov. 2019, doi:10.1016/j.cnp.2019.11.002

[8] Lukaski, Henry C, and Micheal Moore. “Bioelectrical impedance assessment of wound healing.” Journal of diabetes science and technology vol. 6,1 209-12. 1 Jan. 2012, doi:10.1177/193229681200600126

[9] https://www.amazon.com/Thalmic-Labs-Gesture-Control-Presentations/dp/B00VHWBH02

18-500 Progress Report: 10/19/2020



	Tasks	Start	End	Team Member	Status
1	**Phase 1: Project Proposal and Planning**				
1.1	Research EEG feasibility (complexity and price)	9/8/2020	9/10/2020	Everyone	Complete
1.2	Project Abstract (due 9/11)	9/14/2020	9/16/2020	Everyone	Complete
1.3	Research EMG Sleeve construction and games	9/9/2020	9/16/2020	Everyone	Complete
1.4	Order proof of concept components	9/11/2020	9/14/2020	Tarana	Complete
1.5	Proposal Proposal Presentation (due 9/20)	9/11/2020	9/20/2020	Everyone	Complete
1.6	Research EMG Muscles	9/16/2020	9/21/2020	Kayla	Complete
	Phase 2: Design and Implementation				
	Milestone #1: Proof of Concept / Hardware Prep				
2.1.1	Test proof of concept components	9/16/2020	9/21/2020	Tarana	Complete
2.1.2	Sandbox Arduino Code/Scope to read or record EMG	9/16/2020	9/21/2020	Tarana	Complete
2.1.3	Make proof of concept game w/ ball	9/19/2020	9/19/2020	Alex	Complete
2.1.4	Design Data Collection Experiment (Electrode Placer	9/18/2020	9/28/2020	Kayla	Complete
2.1.5	View data stream from a single EMG electrode (proo	9/25/2020	9/28/2020	Everyone/Tarana	Complete
2.1.6	Order Prototype Components	9/27/2020	9/29/2020	Everyone	Complete
2.1.7	EMG Analysis Literature Review	9/21/2020	9/28/2020	Kayla	Complete
2.1.8	Solder Headers to MyoWare Muscle Sensors	10/2/2020		Tarana	In Progress
	Milestone #2: Basic Integration				
2.2.1	UDP Connection Socket	9/19/2020	10/1/2020	Alex	Complete
2.2.2	Setup complete system wiring + hardware	10/6/2020		Tarana	In Progress
2.2.3	Collect All EMG Data (1 and 8 electrodes)	10/12/2020		Tarana + Kayla	In Progress
2.2.4	Make Hardware Simulation w/ UDP for Testing	9/25/2020	10/1/2020	Alex	Complete
2.2.5	Start MyoRun Game (Asset Shopping, Generation)	10/1/2020	10/12/2020	Alex	Complete
2.2.6	Assess signal quality	10/12/2020		Tarana + Kayla	In Progress
2.2.7	Read EMG data into MATLAB	10/12/2020		Tarana + Kayla	In Progress
2.2.8	Classify EMG data to distinguish Fire/Not Fire			Tarana + Kayla	Not Started
2.2.9	Assess binary classification quality			Tarana + Kayla	Not Started
2.2.10	Integration Test #1 (1 input, UDP, Unity)			Everyone	Not Started
2.2.11	Slack			Everyone	Not Started
	Milestone #3: Full Implementation				
2.3.1	Make MyoRun Game Part II	10/12/2020		Alex	Not Started
2.3.2	EMG Signal Processing (8 inputs)			Tarana + Kayla	Not Started
2.3.3	EMG Machine Learning (8 inputs)			Tarana + Kayla	Not Started
2.3.4	Test Game with Simulated Hardware			Alex	Not Started
2.3.5	Integration Test #2 (1 input, UDP, Unity)			Everyone	Not Started
2.3.6	Assess classification quality of multiple electrodes			Tarana + Kayla	Not Started
2.3.7	Add all controls and animation effects in game			Alex	Not Started
2.3.8	Items/Cons/Use of extra classes			Alex	Not Started
2.3.9	Optimize EMG Classification Aglorithm			Tarana + Kayla	Not Started
2.3.10	Integration Test #3 (all electrodes, 4 inputs)			Everyone	Not Started
2.3.11	Record Integration Test for Video			Everyone	Not Started
2.3.12	Slack			Everyone	Not Started
	Phase 3: Performance Testing and Integration				
3.1	Testing classification accuracy			Everyone	Not Started
3.2	Testing lagtime between arm movement and game reaction			Everyone	Not Started
3.3	Testing signal to noise ratio			Everyone	Not Started
3.4	Optimization of System			Everyone	Not Started
3.5	Slack			Everyone	Not Started
	Phase 4: Final Report				
4.1	Record Video			Everyone	Not Started
4.2	Final Presentation			Everyone	Not Started
4.3	Edit Video			Everyone	Not Started
4.4	PROJECT DUE (Due 12/12 or 13)			Everyone	Not Started