Electromyographic Control of Computational Systems

J Mehlman, J Swartz: SFSU Engineering 845

Abstract— **Gesture based control of computers is a problem under active research that will benefit wide swaths of users. We created a system built around myoelectric sensors placed on muscle groups. By taking advantage of machine learning algorithms, we are able to successfully control a computer mouse by measuring muscle activation with response times under 240 milliseconds.**

I. Introduction

The default method of interfacing with computational equipment is the keyboard and mouse. These inputs can be challenging for people with even mild disabilities. If not used correctly, improper use can lead to nerve damage which can cause discomfort and disability.

We have captured electromyography data with low-cost hardware, processed the signals with the help of a machine learning algorithm, and then used that information to control a human interface device to operate a mouse for any common computer system.

II. Design and Implementation

A. Hardware and Software Architectures

The system "Fig 1" has the sensors attached to the subject "Fig 6". The signals from the sensor are acquired by the Data Acquisition (DAQ). The DAQ then sends the data via a serial connection to the Signal Processing software (MATLAB). The signals are processed and interpreted to gestures and sends gesture information to the Human Interface Device (HID) over a serial communications port. The HID then commands a computer system to control (in this case) a computer mouse.

Hardware Descriptions:

Sensor Pads: Medical grade conductive pads with bio-compatible conductive gel are used to receive and transmit muscular electrical impulses from the test subject to the sensor.

Sensor Circuit: A premade sensor circuit is used to receive and pre-process the EMG signals from the sensor pads. This off-the-shelf EMG sensor contains functions that prevent utilization of any signal processing outside of a high-low output. To remedy this, the circuit must be modified by removing the undesired functions and sample the raw EMG signal.

Amplifier: Texas Instruments MCP6002 (TI6002) is the amplifier used to receive the raw EMG signal, amplify it, and transmit it to the signal filter. This is a positive signal amplifier and produces 0 VDC whenever the signal is negative. Compensation for this factor is achieved by pulling the baseline (zero) signal to 2.5-3.0 VDC.

Low Pass Filter: An RC filter is used to establish the desired cutoff frequency with the resistor in line between the amplifier and Arduino input to also act as a current limiter.

The program code is built as three major categories. Data Acquisition, Signal Processing, and Human Interface Device. The software flow is shown in "Fig. 3":

- Data Acquisition (DAQ)
  - arduinoDAQ.ino
  - serialComms.cpp
  - serialComms.h
- Signal Processing (MATLAB)
  - The entry point
    - frontEnd.mlx
  - Record data for analysis and training
    - recordDataBlocks.m
    - blockExtract.m
  - Run Realtime capture of data from the DAQ, process data, and send to the HID
    - realTime.m
  - Serial packets receive and parse, Circular buffer for Realtime
    - dataHandeler.m
  - Feature Extraction
    - dataProc.m
    - calcFFT.m
  - Data Analisis tools
    - plotFeatures
- Human Interface Device (HID)
  - HID.ino
  - gestureMap.h
  - serialComms.cpp
  - serialComms.h

The sensors, after filtering and amplification, are connected to the analog inputs of an Arduino Uno R3. The DAQ sends the unprocessed signals along with a timestamp to the Signal Processing suite running in MATLAB.

J B Mehlman is with San Francisco State University, San Francisco, CA 94132 USA (e-mail: jmehlman@sfsu.edu, SFSU ID 903035606).

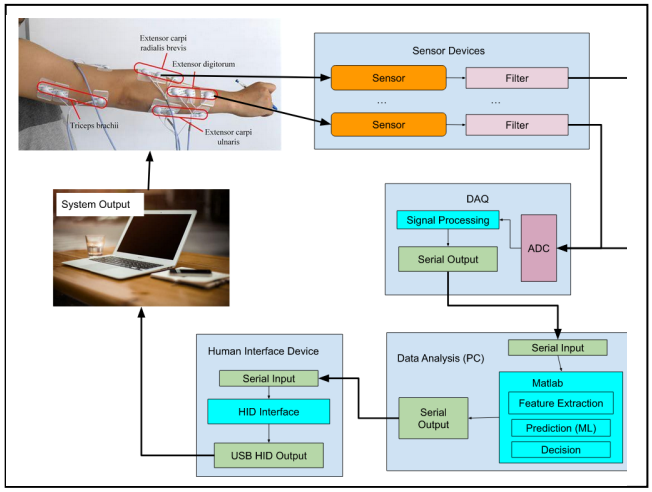
J Swartz is with San Francisco State University, San Francisco, CA 94132 USA (e-mail: jswartz@sfsu.edu SFSU ID 903035606).

If the system is running in training mode. The signals are saved raw as data blocks. The size of the blocks is configurable, for this experiment a block size of 1024 was used, this selection is arbitrary. These packets are then split into the data processing block size, with overlap processing as suggested by Englehart [1]. Shown in "Fig 2" from Englehart. The data is collected with training and testing blocks for each gesture. The data is automatically saved as a MATLAB .m file and saved in a timestamped directory with training and testing subdirectories. The filenames include the date, time, gesture, and save number.

TABLE I. HARWARE COMPONENTS

System	Components		
	Component	*Vendor*	*Part*
Sensor	Instrunemtaion Amplifier	Advancer Tech.	Muscle Sensor Platinum V3
Sensor	Self Adhesive Electrodes	Corvidian	H124SG
Amplifier	Dual Channel Low Power Op Amp	Texas Instruments	TI6002
Filter	Ceramic Capacitors (10%)	1A Manufacturing	EE916A
Filter	Metal Film Resistors (1%)	Bojack	BJ-50
DAQ	Microcontroller	Arduino	UNO R4
HID	Microcontroller	Adafruit	QT Py M0

Figure 1. System Block Diagram.



Once the data is acquired it is processed into features. A vector is made for each time block with N-Features per channel.

When the system is in training mode, the labels are appended to the end of the vector, then stacked into an (N+1) x M matrix for M time blocks. The data features are then ready for machine learning model training and validation.

When the system is running in Realtime, the data comes in through a circular buffer directly into the overlap processing scheme. The Realtime array is sent to the pre-trained model. After the model returns the gesture, the gesture is sent over a serial connection to the HID, which in turn commends the user's computer operation.

Figure 2. Overlap Processing (from Englehert) [1].

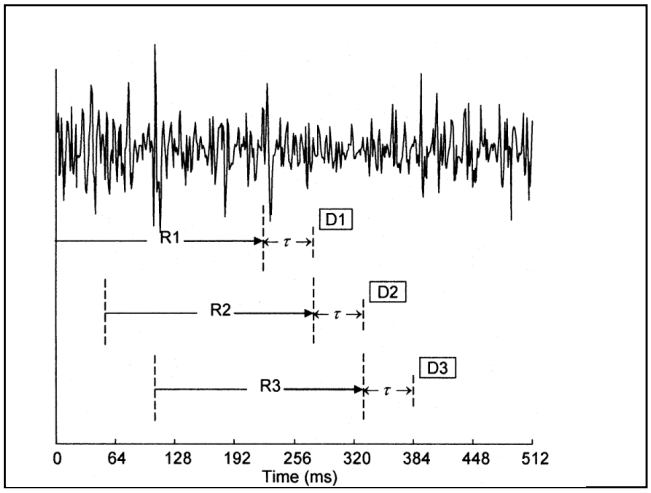
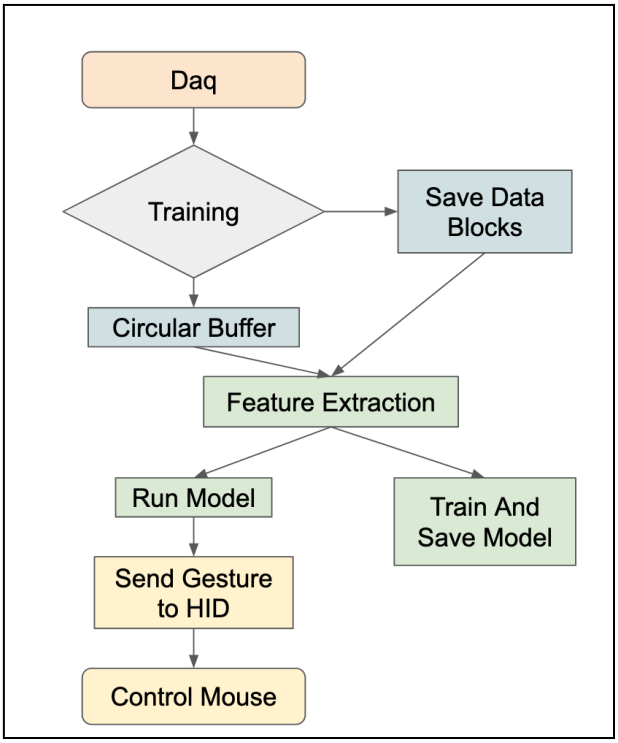


Figure 3. Software Flowchart



B. *Critical Aspects and Requirements*

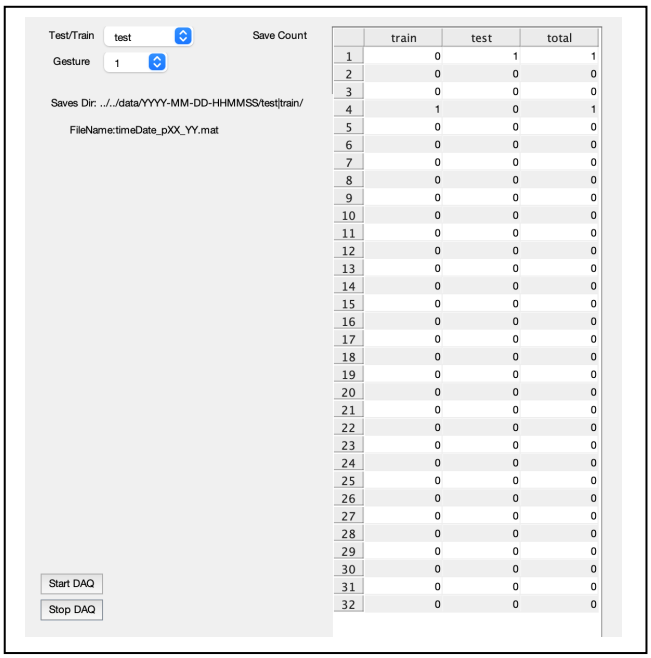
If the system is not easy to use, it is useless. One aspect of usability is the training protocol. This must be smooth and relatively seamless. With this in mind, we created a user interface "Fig 4" to keep track of save and gesture numbers and to create the file structure automatically tagging the gesture number and saving as "test" or "train.

Englehart suggests [1] that 300 ms is sufficient for controlling a real-time system. However, we have used a more aggressive target of 240 ms. Counting from the start of a block,

60 packets at 300 Hz is 200 ms. This leaves up to 40 milliseconds for data processing. Our total time for processing varied from 5 to 20 milliseconds "Table II".

- FFT time: ≅ 50 µS
- Feature Extraction: 100 µS to 10 mS
- Model run: 5 mS to 10 mS

Figure 4. Training User Interface



Data quality and repeatability is a huge issue. The cleaner the data is the better the processing and machine learning will do. Issues that affect our data quality are:

- 60 Hz noise (and first harmonic at 120 Hz) "Fig 5"
- Unknown 5 Hz to 10 Hz and harmonics "Fig 6"
- Subject fatigue.
- DC offset. A DC offset of 2.5 V was used due to the Arduino only having single ended measurements available.
- DC drift

TABLE II. DATA CAPTURE TIMING

DAQ Bandwidth	300 Hz
Data Block Time	200 milliseconds
Nyquist Frequency	150 Hz
Sample Period	3.3 milliseconds
Block Size	60 packets
Pakcet overlap	12 packets
Block Processing time	5 to 20 milliseconds

Figure 5. Subject Fatigue

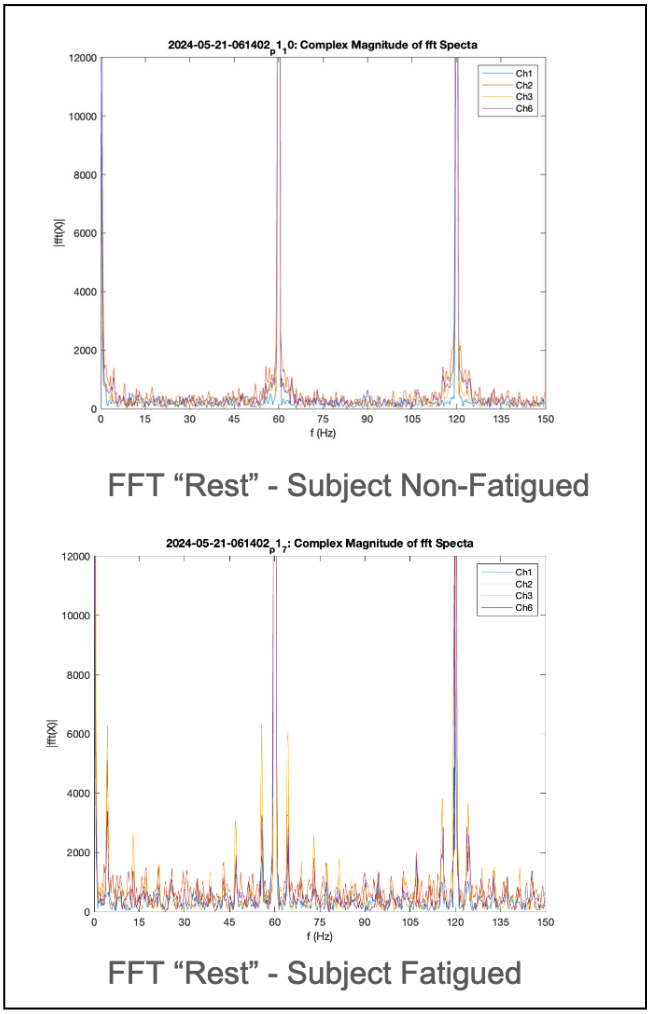
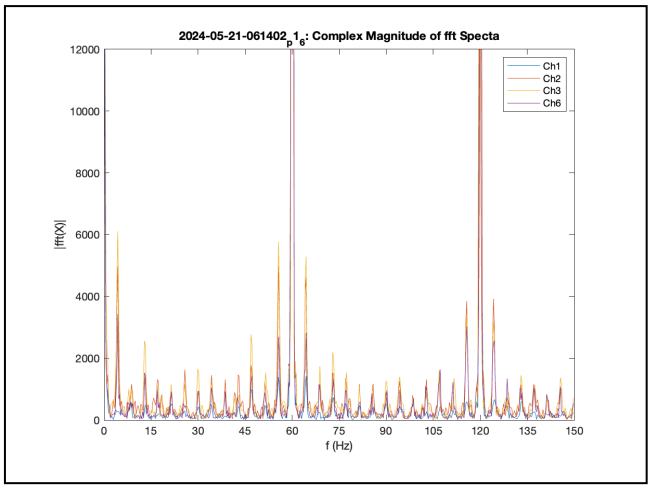


Figure 6. Unknown Noise Spikes Shown for Gesture 0 "rest"



Note that in "Fig 5" the noise broadband noise floor is much lower than the fatigued portion of "Fig 4". However, the narrow, evenly spaced spikes are much more pronounced. 60

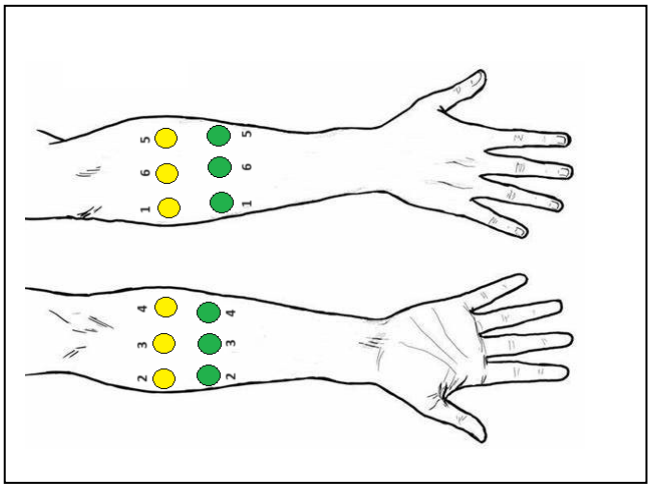
Hz noise and the DC drift were both probably a result of an incorrect ground path.

C. Implementation

This project uses an off-the-shelf, modified instrumentation amplifier circuit. The "as made" circuit includes rectifier, smoother, and gain functions that alter signal output to a variable voltage level as opposed to a raw EMG signal making it unusable for the Matlab training program. Circuit modification is required and includes removing those functions, creating specific amplifiers/filters to suit the needs of the project/data, and mounting all components to a breadboard. The resulting circuit includes a precision instrumentation amplifier (AD8221), low power op amp (TI/MCP6002), and a low pass Resistor-Capacitor (RC) filter tuned to 115 Hz (±2 Hz). Six replicant channels provide inputs to an Arduino R3 (Channels A0-A5).

Current limitations on the Arduino board for all I/O ports are restricted to 200 milliamps (mA). The filter effectively reduces current per Arduino input channel to less than 5 mA. Further reduction of potential strain on the Arduino is through a dual source power supply (Lambda LPD-422A-FM) to all breadboarded components. The Arduino receives 5 Volts Direct Current (VDC) using the USB connection to a computer.

Figure 7. Sensor Position on Forearm



Safety testing is imperative. All circuits and leads, especially those in contact with the test subject, are verified under the most extreme conditions expected. Positive 15 VDC and negative 15 VDC set on the power supply with the current limiters turned off resulted in a maximum of 2.0 mV and undetectable current on the test leads. Medical device standards require direct contact touch current to be less than 100 microamps (µA). When the measured values comply with established safety standards, it is safe to proceed. Voltages on the power supply are reset to positive 5 VDC and negative 5 VDC for training and testing.

Placement of the yellow (mid-muscle) sensor pads are on the right forearm, approximately three inches below the elbow crease, in a symmetric/equally spaced pattern around the forearm. The green (end-muscle) sensor pads are placed in line with their corresponding yellow pads, approximately one inch closer to the hand to reduce cross channel interference.

The low pass filter is designed using 1% tolerance resistors and 10% tolerance capacitors. Tuning is achieved by installing a resistor in series between the amplifier output (V_{IN}) and the load (V_{OUT}). Capacitors are added in parallel between V_{OUT} and ground until the desired cutoff frequency is achieved. The following are the schematic and equation used.

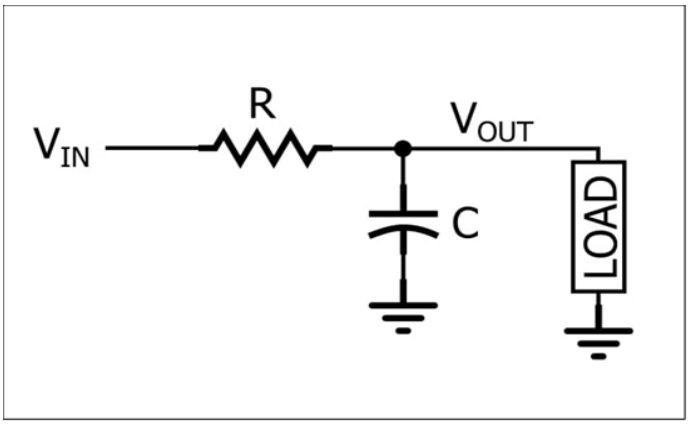


Figure 8. RC Filter Circuit [9]

Equation:

$$f_c = (2\pi RC)^{-1}$$

Where:

$$f_c\text{: } Cutoff\ Frequency\ (Hz)$$
$$R\text{: } Resis\tan c\, e\ (\Omega)$$
$$C\text{: } Capaci\tan c\, e\ (F)$$

Usage

$$f_c = \left[2\pi\left((1000)(1.384\mu F)\right)\right]^{-1} = 115Hz$$

The TI/MCP6002 is configured as a non-inverting operational amplifier using 1% tolerance resistors (R_2 and R_1) to set the gain. The following are the schematic and equation used.

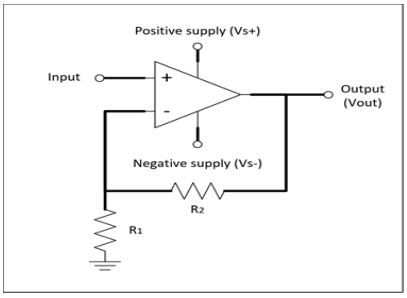


Figure 9. Non-Inverting Amplifier Schematic [9]

Equation:

$$G = R_2/R_1$$

Where:

$$G\text{: } Gain$$
$$R_2\text{: } Feedback\ Resistor\ (Ohms)$$
$$R_1\text{: } Gain\ Resistor\ (Ohms)$$

Usage:

$$G = (10000\ \Omega)/(10\ \Omega) = 1000$$

As discussed earlier, the purchased sensor contains built in features and functions that disguise the raw EMG signals and prevents deep learning. Figure **10** shows the circuit design and Figure **11** demonstrates the resulting output that masks the signal.

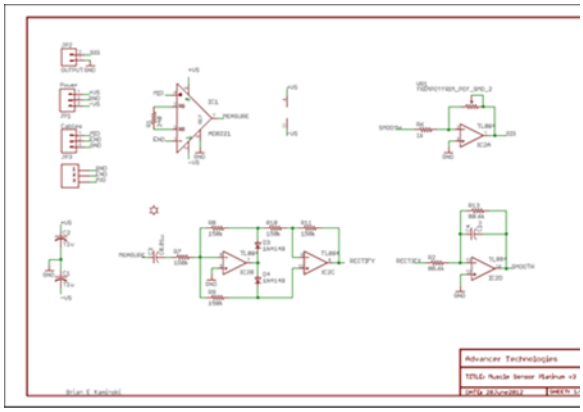


Figure 10. Circuit as Purchased [8]

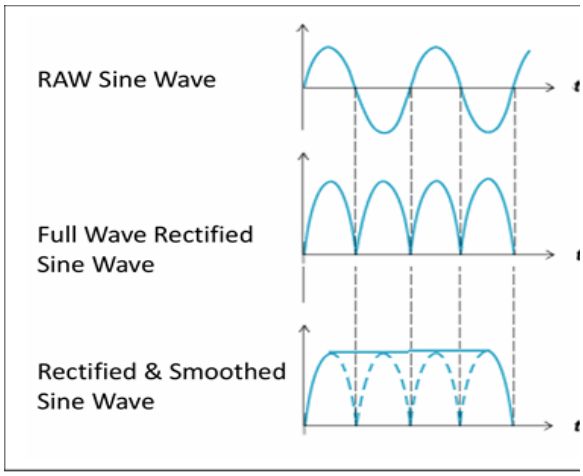


Figure 11. Resulting Signal From Purchased Circuit [8]

After circuit modification (Fig. **12**), the raw EMG signal is amplified and sent through a low pass filter set to a 115 Hz cutoff frequency. The resulting, modified signal is able to be used in the deep learning algorhythims. Each of the sensors used have the same modification.

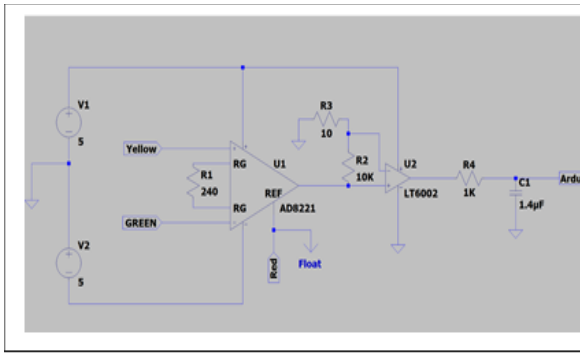


Figure 12. Sensor Circuit Modified (LTSpice Circuit Model)

D. Algorithms

Serial communication is in binary rather than ASCII to increase processing speed and decrease packet size. The protocol from the DAQ to MATLAB is described in "Table III" and from the MATLAB to the HID "Table IV".

TABLE III. DAQ SERIAL PROTOCAL

Datum	***Description***	***Data Type/Size***
SYNC BYTE	0xAA	Uint 8
Time Stamp	Milliseconds Since DAQ Boot	Uint 32
CH 1	Myoelectric data (ADC Counts)	Int 16
…	Myoelectric data (ADC Counts)	Int 16
CH N-1	Myoelectric data (ADC Counts)	Int 16
Ch N	Myoelectric data (ADC Counts)	Int 16
TERMINATE	CRLF	2 x Uint 8

TABLE IV. HID SERIAL PROTOCAL

Datum	***Description***	***Data Type/Size***
SYNC BYTE	0xAA	Uint 8
Data Byte	Gesture Number (Negative for setting)	Int 8
Setting	Action to Apply to Gesture	Char (aka uint8)

We used some features that are in the frequency domain. While it can be beneficial to use frequency-based features, even with clean signals, our data was far from clean. The same data is presented both in the time domain and in the frequency domain "Fig 13". In the time domain, it is very difficult or impossible to see the signal at all due to the large 60 Hz noise. When processing in the frequency domain it is very easy to skip over the noise, by simply only looking in the frequency bands that don't have the noise, we are only looking at the 0 Hz component, and the 65 Hz to 100 Hz band. Note, time constraints make the use of a MATLAB filter challenging, using postprocessing filters proved to take too much time for real-time processing under our 40-millisecond limit. Furthermore, while both software and hardware filters can be beneficial, they do also affect the data in the non-filtered zone, adding frequency dependent delays and attenuation outside of the desired band that are detrimental to feature generation.

The Frequency domain conversions are done with a Fast Fourier Transform (FFT):

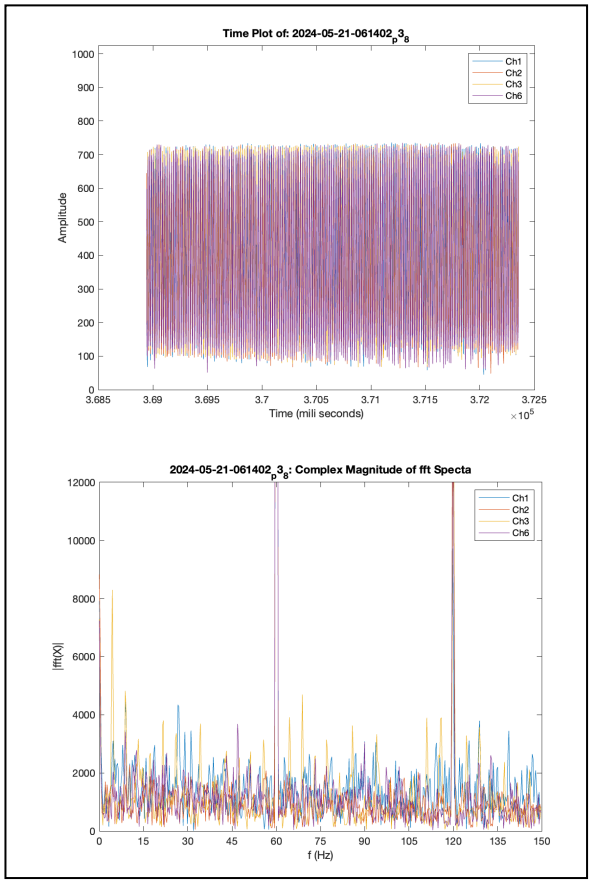
- The signal mean (1) is subtracted from the signal to remove the DC offset.
- A Hann window is applied to reduce spectral leakage
- The magnitude of the signal is calculated from the Real and Imaginary portions of the frequency domain data.

The features calculated are shown in "Table V"

TABLE V. FEATURES AND FORMULA

Feature	*Formula*	
Mean	$\mu = \frac{1}{n}\sum x_i$	(1)
Standard Deviation	$\sqrt{\frac{1}{n}\sum(x_i - \mu)^2}$	(2)
Peak to Peak	Max(x) – Min(x)	(3)
RMS	$\sqrt{\frac{1}{n}\sum(x_i)^2}$	(4)
Mean Absolute Value	$\frac{1}{n}\sum\lvert x_i\rvert$	(5)
FFT Peaks/Locations (0 Hz, peak)	$f(x) = a_0 + \sum_{n=1}^{\infty}\left(a_n \cos\frac{n\pi x}{L} + b_n \sin\frac{n\pi x}{L}\right)$	(6)
Spectral Centroid (65 Hz to 100 Hz)	$From\ i = f_{min} to\ f_{max}: \frac{\sum I_i * f_i}{\sum I_i}$, $where\ I = magnatude\ at\ frequency = f_i$	(7)

Figure 13. Time and Frequency Domain Plots of the same Data



(Gesture 3: “Thumbs Up”)

For all features, except “mean”, the mean is subtracted prior to calculating the feature. This is necessary due to the large DC offset. Also note, that the spectral peaks and locations are not useful due to the impulse train nature of muscle activation [Schwartz] except for the 0 Hz magnitude (6). Furthermore, the Mean Absolute Value (5) proves detrimental to the Machine Learning Algorithm’s ability to predict gestures.

Once the feature set is selected, it is sent to a machine learning algorithm. Directly after the training and testing data is captured the features from the data are sent to a large selection of machine learning algorithms, and the best performing/fastest algorithm is selected. The best performing model for this experiment is a Support Vector Machine (SVM) model. “Table VI” shows the hyperparameters for the test run. It is interesting to note that using principal component analysis was always detrimental to the model’s performance.

TABLE VI. EXAMPLE HYPERPERAMETERS FOR SVM

Parameter	*Value*
Kernal	Quadratic
Kernal Scale	Automatic
Standardize	Yes
Box Constraint Level	1
Multiclass Codeing	One-vs-One
PCA	Disabled

TABLE VII. FULL GESTURE LIST

Gesture Number	*Name*	*Static/Dynamic*
1	rest	Static
2	fist	Static
3	thumb up	Static
4	thumb down	Static
5	point index finger	Static
6	knife hand	Static
7	one	Static
8	two	Static
9	three	Static
10	four	Static
11	five	Static
12	peace	Static
13	bull	Static
14	I love you	Static
15	hello (wave)	Dynamic
16	princess wave	Dynamic
17	ok	Static
18	quiet coyote	Static
19	stop/halt	Static
20	hold beer	Static
21	drink beer	Static
22	hold pen	Static
23	type on keyboard	Dynamic
24	scratch	Dynamic
25	sign language "A"	Static
26	sign language "B"	Static
27	sign language "C"	Static
28	sign language "D"	Static
29	sign language "E"	Static
30	the bird	Static
31	wrist up	Static
32	wrist down	Static

III. EXPERIMENTS

The beginning point is to start with a full set of Gestures “Table VII”, sensors “Fig 7” and Features “Table V”. For the list of gestures at least 2 training and 1 testing run is gathered for each feature using “recordDataBlocks”.

Figure 14. Feature Extraction of 5 Gestures: Mean

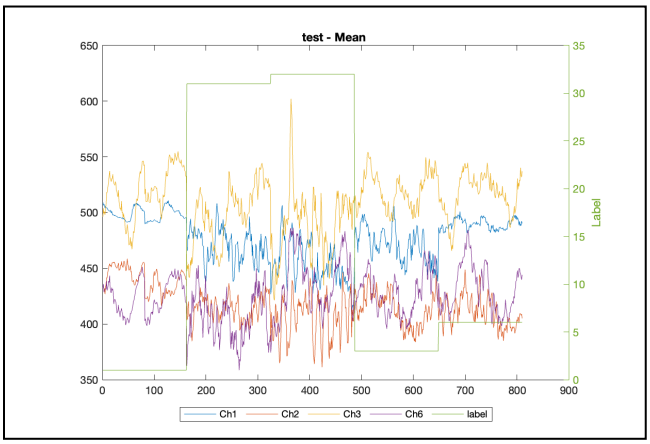


Figure 15. Feature Extraction of 5 Gestures: Standared Deviation

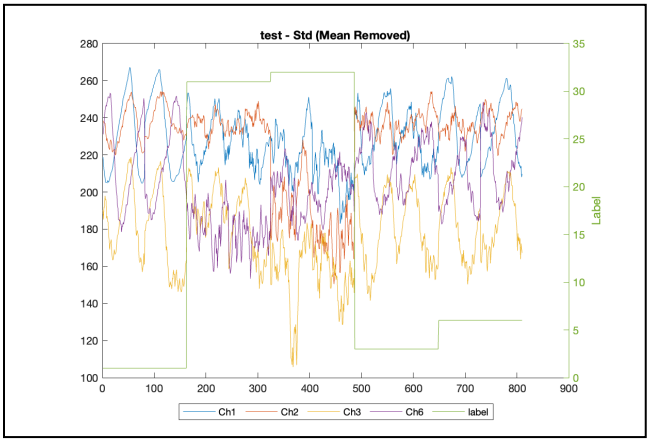


Figure 16. Feature Extraction of 5 Gestures: Peak to Peak

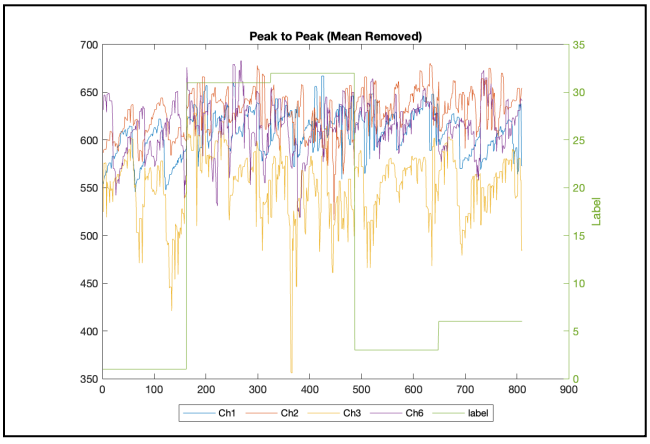


Figure 17. Feature Extraction of 5 Gestures: RMS

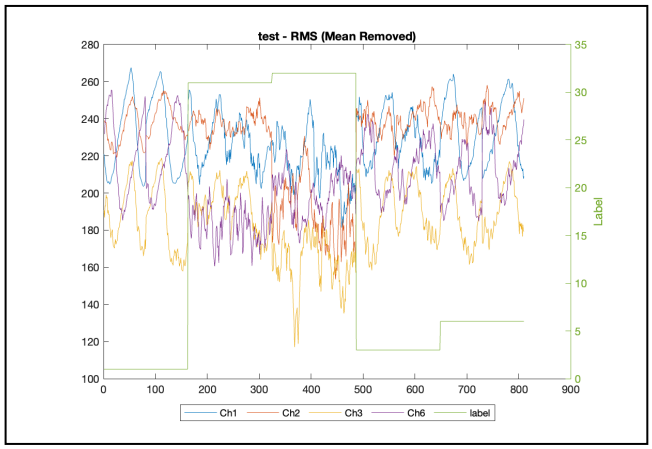


Figure 18. Feature Extraction of 5 Gestures: Mean Absolute Value

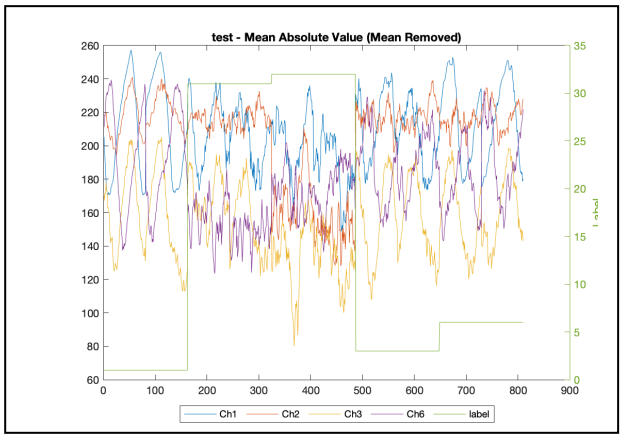


Figure 19. Feature Extraction of 5 Gestures: 0 Hz

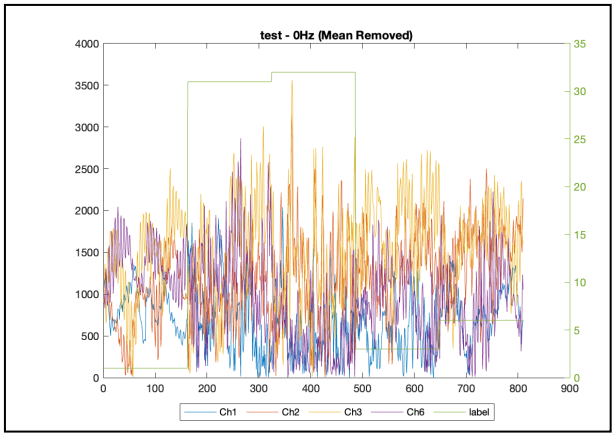
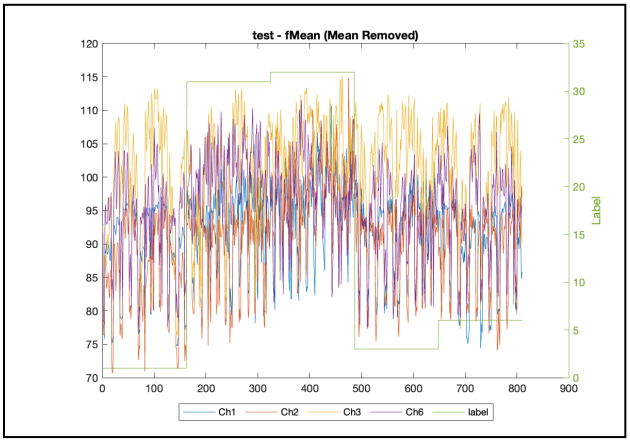


Figure 20. Feature Extraction of 5 Gestures: Spectral Centroid (65 Hz to 100 Hz)



Once the features are generated, the training and testing sets are sent to MATLAB's "clasificationLearner" for model training. A confusion matrix is generated from the testing data, "Figure 21", of the best models. From this matrix gestures are culled that have poor performance, or are confused with other, better performing, gestures. When the gesture list is smaller (approximately culled in half), the features and channels are removed one at a time to determine the effectiveness/detriment of that feature/data channel using several models to compare. Sometimes removing a feature would help one model while hurting another. From this set it was determined that Ch 4 and Ch 5, as well as the feature "Mean Absolute Value" produce lower accuracy models and are removed. This process is repeated until data suggests a best set of 5 Gestures representing the mouse commands, "Table VIII", is achieved while keeping in mind that "rest" must be included as "Stop".

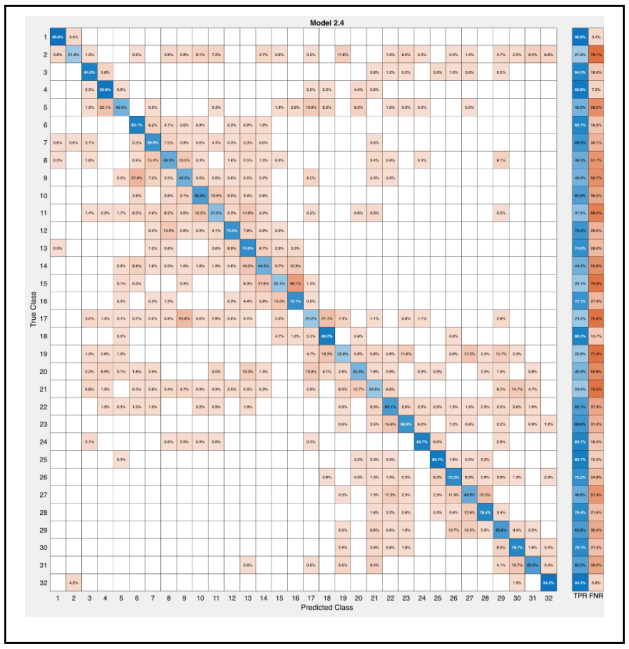
Once the best performing gestures, channel sets, and features are determined, the subject is sat down to train a new model. This time 8 training blocks and 2 testing blocks are recorded only on the selected 5 gestures.

The model is quickly trained and installed. Then the "realTime" section is run and the subject controls the mouse using electromyography generated sent to the model, and the model output is transmitted to the HID, which in turn drives the computer.

TABLE VIII. HID SERIAL PROTOCAL

Numbr	***Gesture***	***Mouse Command***
1	Rest	STOP
31	Wrist Up	UP
32	Wrist Down	Down
3	Thumb Up	LEFT
6	Knife Hand	RIGHT

Figure 21. Full Set Confusion Matrix

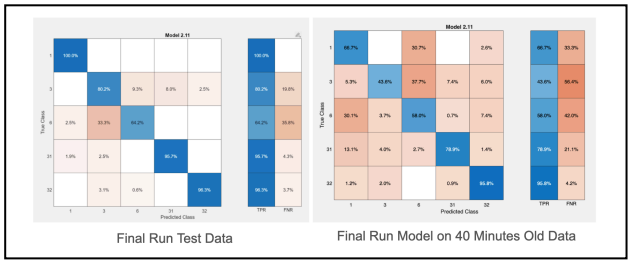


IV. RESULTS AND DISCUSSION

The first run performs well. The test subject is able to control the mouse movement. However, performance quickly falls off. While "Up" and "Down" maintained high controllability, "Stop" and "Right" begins to have difficulties.

We then compare the final training confusion matrix with data recorded 40 minutes prior "Fig 22" and discover that it does a poor job, confusing a commanded "Stop" with "Right" and vice versa.

Figure 22. Confusion Matrix of Final test Compaird with the same model tested on 40 minuts old data



V. TEAM ROLES

James Swartz:

- Sensor Selection
- Filter Design
- Hardware Testing

Joshua Mehlman:

- Systems Architecture
- Signals Analysis
- Software design and implementation
- Machine Learning Model Training

VI. Conclusion and Future work

We successfully created a system able to translate myoelectric muscle signals to computer mouse control. Using a set of off the shelf sensors that came with a processing circuit. We first must modify the signal processing circuit to remove the rectification and 2 Hz low pass filter. Then we add a second amplifier and a 115 Hz low pass filter to avoid aliasing. We then record training data, which is overlap processed and a set of time domain and frequency domain features are extracted. Those features are used to train a machine learning algorithm. The algorithm selected is almost always an SVM.

Once the algorithm training is complete, the myoelectric signals are overlap processed in real time. The set of features determined to give the machine learning algorithm the best results are extracted and sent to it. The output of the algorithms prediction of the subject's current gesture is sent to a Human Interface Device, which in turn commands mouse movements to a computer system. The subject is able to successfully control the mouse in "Up", "Down", "Left", "Right", and "Stop" using gesture alone.

The systems performance is not as good as we would have liked it to be. We are only able to control 5 gestures (including stop). The largest concern we have is that the system needs to be trained just prior to operation, and the performance quickly falls off.

The most significant challenge we have in getting better performance is sensor noise, and drift. There is a large 60 Hz and harmonic noise "Fig 13". This noise is so large that it is impossible to view the signals in time domain, as the 60 Hz component massively envelops the actual signal. There is also a 5 Hz to 10 Hz noise signal that comes and goes with harmonics past our measurement range. We also are fighting a DC drift.

We are using signals directly read from an Arduino. Since the Arduino is a single ended measurement device, we need to add a significant DC offset to avoid clipping the signal or damaging the inputs. This can be remedied by using a dedicated Data Acquisition system with dual ended inputs.

For the noise and DC drift, we will need to find and eliminate the source. The 60 Hz and the DC drift are both probably from an incorrect circuit ground. It would also be beneficial to install a hardware-based band-gap filter around 60 Hz and harmonics. A larger investigation is needed to identify and remedy the 5 Hz to 10 Hz noise.

The data acquisition rate was selected to be 300 Hz. This was selected in response to the limitations of MATLAB's blocking serial read. It was discovered that 300 Hz is the fastest we could take the data in and still have time for processing. At 300 Hz we had 26ms for processing, at 1 kHz we could not keep up at all. This portion should be re-written outside of MATLAB as a non-blocking interrupt-based serial read.

User fatigue was an issue. The subject was audibly complaining about fatigue during training. We should re-think the training protocol, instead of running through the gestures non-stop, do them one at a time, with good rest between when the subject is feeling fatigued. This would more closely match operation, when if the user is tired, they can simply stop for a moment and rest.

There were also two planned features that did not get implemented. The first is that it would be very useful to be able to monitor the signals in the time and frequency domain, real time. This could let us know if there is a sensor or fatigue issue. The second feature that we started to work on is the ability to program gestures to actions on the fly. Once we have a trained gesture set, a gesture can be tagged as "Program". When that gesture is read, the system will go into a program mode where the next gesture is paired with the command the user would like it to be mapped with. The "Program" gesture is then repeated to place the system back in operational mode.

Appendix

Hardware and instrumentation used:

Test Bed	Breadboard	Twin Industries	TW-E41-1060
Power	Power Supply	Lambda	LDP-422A-FM
Measure	Multimeter	Fluke	3000-FC

Power Supply: The Lambda LPD-422A is a dual output, 0-40 VDC power supply with current limiters that span 0-1 A. This equipment is used to power sensors, amplifiers, and filters.

Measurement Equipment: A Fluke Model 3000FC ensures safety of the experiment regarding safe human interaction.

Test Bed: A large breadboard is required for this experiment as it uses six assembled sensor circuits. Any typical breadboard will perform this action but may require additional breadboards.

Acknowledgment

Special thanks to Shannen Swartz for filling in as the test subject at the last moment.

REFERENCES

[1] Englehart, K., & Hudgins, B. (2003). A robust, real-time control scheme for multifunction myoelectric control. IEEE transactions on biomedical engineering, 50(7), 848-854.

[2] Oskoei, M. A., & Hu, H. (2007). Myoelectric control systems—A survey. *Biomedical signal processing and control*, *2*(4), 275-294.

[3] Schwartz, L. (1951), Théorie des distributions, vol. Tome I, Tome II, Hermann, Paris

[4] Chen, B., Chen, C., Hu, J., Nguyen, T., Qi, J., Yang, B., ... & Goitz, H. (2022). A Real-Time EMG-Based Fixed-Bandwidth Frequency-Domain Embedded System for Robotic Hand. Frontiers in Neurorobotics, 16, 880073.

[5] Anasteiv, A., Kadone, H., Marushima, A., … & Ishikawa, E. (2022). Supervised Myoelectrical Hand Gesture Recognition in Post-Acute Stroke Patients with Upper Limb Paresis on Affected and Non-Affected Sides. MDPI Journals, Sensors 2022, 22(22), 8733; https://doi.org/10.3390/s22228733

[6] Wood, B., Tangen, C., Cummings, S. W., et al (2024).Human Muscle System. Britannica Human muscle system | Functions, Diagram, & Facts | Britannica

[7] Jarque-Bou, N., Vergara, M., Sancho-Bru, J., et al (2019) A Calibrated Database of Kinematics and EMG of the Forearm and Hand During Activities of Daily Living. Scientific Data Journal, Article Number 270 (2019). A calibrated database of kinematics and EMG of the forearm and hand during activities of daily living | Scientific Data (nature.com)

[8] Advancer Technologies (2014) Muscle Sensor v3 User's Manual.pdf

[9] Keim, Robert (2019) What Is a Low Pass Filter? A Tutorial on the Basics of Passive RC Filters | All About Circuits (allaboutcircuits.com)