Department of Electrical & Computer Engineering

Myo-Controlled Robotic Hand

Zackary Minshew
Jheric Byrd
Dr. Fernando Rios

Friday, April 17th, 2020

AKNOWLEDGEMENTS

We would like to express immense gratitude to our supervising professor, Dr. Fernando Rios, Department of Electrical and Computer Engineering, Georgia Southern University for guiding us and for providing intellectual insight throughout this project. We would also like to thank every other professor in the department for all of their help throughout our time here at Georgia Southern University. Finally, we would like to thank all of the resources used listed in the references section for insight on the unknowns of the project.

ABSTRACT

This project explores the use of the surface electromyographic sensors (sEMG) in the Myo® Armband by Thalmic Labs, along with the LattePanda Delta ultimate mini-board by DFRobot, and a feed-forward pattern recognition neural network to control a 3D printed robotic hand. To have the previous mentioned items all coordinate together with ease a user-friendly application was created. The application allows for a more welcoming approach for other individuals to use this technology, rather than having to learn the entirety of the complex project architecture. The Myo armband has eight evenly distributed sEMG sensors that read data when the sensor is touching skin. The artificial neural network was initially trained just for five gestures: Rest, Fist, Fingers & Hand Spread, Wave Hand In, & Wave Hand Out. The goal is to expand these five gestures into at least fifteen gestures. These new gestures would be comprised of the previously mentioned five gestures and then some gestures from the American Sign Language Alphabet. The classified output from the artificial neural network is what determines the command for each of the servos in the robotic hand. This document demonstrates how individuals can achieve similar results to that of advanced technology while saving a significant amount of money.

TABLE OF CONTENTS

LIST OF FIGURES 6
LIST OF TABLES 7
1. INTRODUCTION 8
1.1. Objective of the Project 8
1.2. Organization of the Report 8
2. BACKGROUND 9
2.1. Electromyography 9
2.2. Artificial Neural Network Theory 9
2.3. MYO Armband 10
3. PROJECT GOALS 11
4. METHODOLOGY 12
4.1. Project Materials and Equipment 12
4.1.1. Myo armband by Thalmic Labs 12
4.1.2. LattePanda Delta Tiny Ultimate Windows Device 13
4.1.3. 3D printed robotic hand 13
4.1.4. Six server motors 14
4.1.5. Computer with MATLAB and Arduino software 14
4.2. Experiment and Design Overview 15
4.2.1. Original Gestures Experiment 15
4.2.1.1. Neural Network Architecture 16
4.2.2. First American Sign Language Gestures Experiment 16
4.2.2.1. Neural Network Architecture 16
4.2.3. Final American Sign Language Gestures Experiment 17
4.2.3.1. Neural Network Architecture 17
4.2.4. Preprocessing and Signal Conditioning 18
4.2.5. MATLAB Application and Development 18
5. PROJECT BREAKDOWN AND TASKS 20

6. PROFESSIONAL COMPONENTS 22
6.1. Economic Concerns 22
6.2. Environmental and Sustainability Concerns 22
6.3. Manufacturing Ability Concerns 22
6.4. Ethical, Legal, and Health Concerns 22
7. RESULTS 23
7.1. Original Gestures Experiment 23
7.2. First American Sign Language Gestures Experiment 25
7.3. Final American Sign Language Gestures Experiment 28
8. DELIVERABLES 32
8.1. Research Symposium Poster 32
8.2. Project Webpage 33
8.2.1. Home 33
8.2.2. About 33
8.2.3. Components: Software 34
8.2.4. Components: Hardware 34
8.2.5. Documents 35
8.2.6. Contact 35
8.3. MATLAB Code 36
8.3.1. Custom Application 36
8.3.2. Data Collection Function MATLAB Code 65
8.3.3. Artificial Neural Network Function MATLAB Code 70
8.3.4. Live Streaming Function MATLAB Code 74
8.4. LattePanda Delta Pinout AutoCad Drawing 80
9. CONCLUSION 81
10. REFERENCES 82

LIST OF FIGURES

Figure 1: Myo Armband by Thalmic Labs 12
Figure 2: LattePanda Delta device by DFRobot 13
Figure 3: Dexterous 3D Printed Robotic Hand 14
Figure 4: Architecture of the project function and equipment 14
Figure 5: Five hand gestures used and Myo armband placement ... 15
Figure 6: Original Artificial Neural Network design 16
Figure 7: New Artificial Neural Network design 17
Figure 8: MATLAB Application created for this project 19
Figure 9: Senior Project I Gantt Chart 21
Figure 10: Senior Project II Gantt Chart 21
Figure 11: Validation Fail Check plot of the trained ANN 23
Figure 12: Performance plot of the trained ANN 23
Figure 13: Confusion matrix result from the trained ANN 24
Figure 14: Five American Sign Language hand gestures used 25
Figure 15: NN Tool displaying data from trained ANN 26
Figure 16: Trained ANN Confusion matrix 27
Figure 17: Stem plot showing the classified outputs 27
Figure 18: Data Collection process within the MATLAB application ... 28
Figure 19: Three American Sign Language hand gestures used 28
Figure 20: NN Tool displaying data from trained ANN 29
Figure 21: Trained ANN Confusion matrix 30
Figure 22: Stem plot showing the classified outputs 30
Figure 23: Real-Time gesture classification with Arduino control 31
Figure 24: Real-Time gesture classification with axes plotting 31

LIST OF TABLES

Table 1: Target class array ……..…….. 10

1. INTRODUCTION

1.1 Objective of the Project – what and why it was written

In today's society it is apparent that the role and use of robotics and process automation is becoming increasingly more prominent in many fields and applications, from self-driving automobiles to various types of manufacturing. The expansion of these fields is driven by the persistent and insatiable consumer market, not just in the United States, but in the world. This desire of advancement is typically looking for a means to reduce work or cost. Such as the advancement of robotics in manufacturing that leads to reduced labor and the liability of that labor. A byproduct of reducing the amount of human labor is usually higher profits and the reduction of liability due to cutting insurance and safety requirements need to protect workers. Another desire for these advancements is to reduce error, which could be done by a more advanced autonomously driving vehicle; or to be able to provide someone who has lost a limb with a fully functional prosthetic that emulates a human limb very well. The purpose of this project is to explore a cheap, theoretical alterative to current human prosthetics, and to do this with technology and knowledge used by college students.

1.2 Organization of the Report

This paper presents in detail the methods and results obtained using sEMG from the Myo armband to control a dexterous robotic hand. The paper is organized as follows: Chapter 2 explaining the concepts of electromyography, artificial neural networks, and the myo armband, Chapter 3 states the goals of the project, Chapter 4 describes the methods used to achieve the given results of this project listed in Chapter 7, Chapter 5 lists the tasks throughout this project, Chapter 6 states the professional components of the project, Chapter 8 depicts all the deliverables, and Chapter 9 is the conclusion of this project.

2. BACKGROUND

2.1 Electromyography

An electromyographic signal is the translation of the electrical signals that are transmitted from the motor neurons (nerve cells). The motor neurons send electrical signals to the respective muscle(s) causing it to contract or relax. The EMG signal is a means for diagnostics of the command signals for and individuals' muscles. This information is instrumental in helping doctors diagnose and help patients. There are two different ways to acquire the data of electromyographic signal. One method is use sensors that are placed on the surface of an individual's skin, hence surface electromyographic sensor. The other, more invasive method is to puncture the skin with needles and to stick the needles into muscle tissue to record muscle activity [1]. For this project these surface electromyographic signals are collected and fed into the artificial neural network to train and build this network to process future surface electromyographic signals and provide a classified output, thus predicting hand movement.

2.2 Artificial Neural Network Theory

Artificial Neural Networks are computerized systems that model the neural network structure of animalia brains. Artificial neural networks can be used for the recognition of patterns, prediction of future events based off past trends, and for the classification of data and information by learning from historical data [2]. For this project the neural network created using the Deep-Learning Toolbox in MATLAB is a pattern recognition network that is used to design, train, and classify the selected hand gestures. This pattern recognition network is a feedforward network that is trained to classify input values in accordance to target classes [3]. This means that the target data is comprised of vectors with values of '0' except for the class of which it currently represents that would have a value of '1'. This concept is demonstrated in Table 1.

Table 1: Target class array

Class 1	Class 1	Class 2	Class 2	Class 3	Class 3	Class 4	Class 4	Class 5	Class 5
1	1	0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0	0	0
0	0	0	0	1	1	0	0	0	0
0	0	0	0	0	0	1	1	0	0
0	0	0	0	0	0	0	0	1	1

A feedforward neural network is comprised of a series of layers, with the initial layer being connected to the input of the neural network and then any subsequent layer being connected to the previous layer. The final layer provides the output of the neural network [4].

2.3 MYO Armband

For this project, the Myo armband was used to acquire the surface electromyographic signals. It is a useful tool because it allowed for advanced data collection with a simple means of obtaining that data. The armband has eight surface electromyographic sensors evenly displaced. The signals from these sensors on the armband can be accessed and obtained from a computer via Bluetooth low-energy communication protocol.

3. PROJECT GOALS

The main goal of this project was to explore a cost-efficient alternative to current professional prosthetics. We wanted to achieve this goal with just two Georgia Southern University Electrical Engineering students, relatively cheap resources, and open source technology all within two school semesters. A reason for the desire of reaching this goal is how diverse the uses could be. This could be implemented for a wide array of actions such as sign language or typical daily human actions like eating or grabbing an object.

The first measure of success would be seeing if we achieved a realistic and timely live interaction with the robotic hand mimicking our hand. Then to expand this measure, we would want to expand the amount of gestures used by the end of the project. Which we ultimately did to up to fifteen gestures.

4. METHODOLOGY

4.1 Project Materials and Equipment

The following lists all the materials needed for this project.

- Myo armband by Thalmic Labs
- LattePanda Delta Tiny Ultimate Windows Device
- 3D printed robotic hand
- Six server motors
- Computer with MATLAB and Arduino software

4.1.1. Myo armband by Thalmic Labs

The Myo armband (Fig. 1) is what allowed for the collection and interfacing of the electromyographic signals. The armband is comprised of eight non-invasive surface EMG sensors (sEMG) that detect electronic signals generated by commands from the motor neurons calling for muscle contraction and an inertial measurement unit (IMU). The inertial measurement unit is a device that reads the orientation of the armband and the angular rate using accelerometers and gyroscopes together. The data from each sEMG sensor and the inertial measurement unit is sampled 200 times per second, meaning a frequency of 200 hertz. This is the frequency that the armband communicates to the Bluetooth low-energy dongle that is plugged into a computer's universal serial bus port. The data was accessed in MATLAB using the software package called Myo-Mex Wrapper [5]. This software package converts the source code and functions that are in C++ to MEX files that work directly with MATLAB.

Figure 1. Myo Armband by Thalmic Labs.

4.1.2. LattePanda Delta Tiny Ultimate Windows Device

The LattePanda Delta device (Fig. 2) is going to be used to control the robotic hand, run the deployed artificial neural network, and run the created MATLAB application. The LattePanda Delta device is a “Tiny Ultimate Windows / Linux Device”. It has an Intel Celeron N4100 8th generation processor, 4gb of LPDDR4 2400MHz RAM, 32gb eMMC memory, and it supports WIFI 802.11 and Dual Band Bluetooth 5.0. The user has the option of three different operating systems: Windows 10, Linux, or Ubuntu. In addition to the standard Intel Celeron processor, the Delta also has an Arduino Leonardo co-processor. Between the device’s size, capability of running Windows for the ANN, and the capability of being a microcontroller makes it the ideal component for this project [6].

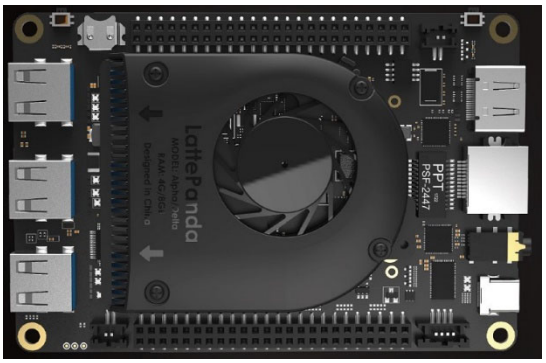


Figure 2. LattePanda Delta device by DFRobot.

4.1.3. 3D printed robotic hand

The artificial robotic hand and arm (Fig. 3) was designed to resemble a human extremity and reenact its motion capability. Based off a design from InMoov, the hand was 3D printed with acrylonitrile butadiene styrene plastic by previous Georgia Southern University Electrical Engineering students. The dexterous hand has motion based off the joints being tensioned with nylon string that is attached to a servo horn, and when the horn rotates it either tightens the string caused the given finger(s) to contract or it rotates the opposite direction causing the finger to extend. There is a servo motor for each finger and one for the wrist. Unfortunately, with this design the motion of the hand’s wrist is limited to one plane dimension [7].

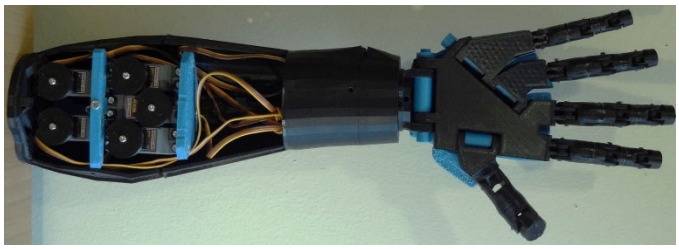


Figure 3. Dexterous 3D Printed Robotic Hand

4.1.4. Six server motors

The six servo motors had 7 kilograms of torque at 4.8 volts and weighed 66.4 grams. Each motor was coreless with metal gears. Metal gears are better than plastic for this project because of the constant use and change of direction that generates wear and heat. Five of the six servo motors each controlled an individual finger and one servo motor controlled the wrist allowing it to move along one axis.

4.1.5. Computer with MATLAB and Arduino software

As previously mentioned, the data from the armband can be accessed on a computer via Bluetooth low-energy dongle. To process this data, MATLAB is needed. To provided control code for the servo motors to run on the Arduino microcontroller, Arduino software is needed. The architecture of this system may be observed in Figure 4.

Figure 4. Architecture of the project function and equipment.

For much of the project design, a computer with 512Gb ssd hard drive, 32 Gb RAM, and an Intel i7 processor. This much power is not necessarily needed, but it does help when it comes to processing a lot of data and training a neural network. However, if the computer was to have too little RAM, such as 4 Gb, this could lead to the system becoming very cumbersome.

4.2 Experiment and Design Overview

For the entirety of the spring semester this experiment was conducted using only two test subjects. One person being a 6’5”, 268 pounds, 23-year-old male and the other person being a 6’2”, 205 pounds, 23-year-old male. Both subjects executed the data collection process in an identical fashion. Both individuals placed the armband on their right forearm, with the Myo logo positioned on top of the brachioradialis.

4.2.1. Original Gestures Experiment

The concept of this experiment was to have five hand gestures that will be classified by the ANN and the prosthetic hand will make a gesture based off the classification from the ANN. The five hand gestures are: Rest, Fist, Spread, Wave In, and Wave Out (Fig. 5). To record the data from the surface electromyographic sensors that is used to train the neural network, the subjects stood vertically erect with a ninety-degree angle of flexion between their lower arm and upper arm.

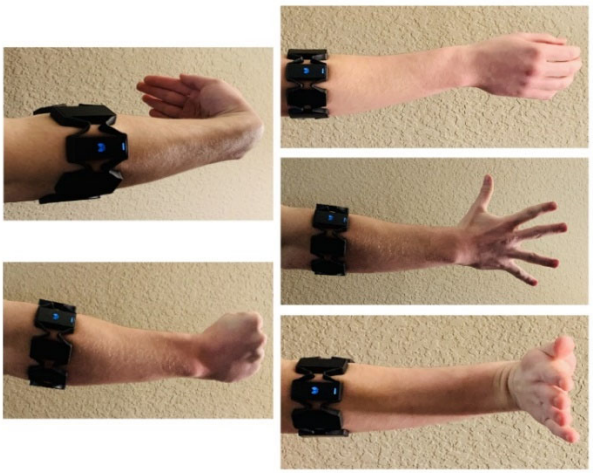


Figure 5. Five hand gestures used and Myo armband placement

4.2.1.1. Neural Network Architecture

An artificial neural network was designed and trained to classify the five original hand gestures. The artificial neural network was a pattern recognition network with eight inputs and five outputs. Once more gestures are permanently added, the amount of outputs will then equal the total number of gestures. For most of the testing, two hidden layers comprised of 125 neurons each were used; along with the epochs being set at 400. There were 120,496 total signals used for the input. For training the artificial neural network, 70% of these signals were used for training while the other 30% was used evenly for validation and training of the neural network. Each input had a target value of one, this is crucial for the gesture classification. The five possible outputs of the artificial neural network were the five different hand gestures (Fig. 6).

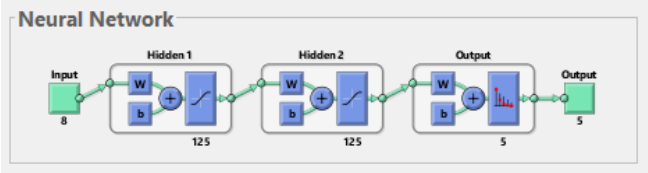


Figure 6. Original Artificial Neural Network design

4.2.2. First American Sign Language Gestures Experiment

Up until the recent isolation due to the coronavirus the experiment has been conducted using only two test subjects. One person being a 6’5”, 268 pounds, 23-year-old male and the other person being a 6’2”, 205 pounds, 23-year-old male. However, it is now only using the latter of the two aforementioned individuals. The armband shall be placed on the right forearm, with the Myo logo positioned on top of the brachioradialis and the subject should stand vertically with the right hand parallel with body pointing towards the ceiling/sky.

4.2.2.1. Neural Network Architecture

An artificial neural network was designed and trained to classify the x amount of selected (5) hand gestures. The artificial neural network was a pattern recognition network with eight inputs and x amount of outputs (depending on the number of desired gestures selected by the user (5)). For the testing, two hidden layers comprised of 125 neurons

each were used; along with the epochs being set at 400 (Fig. 7). There were 71,640 total signals (8955*8) used for the input. For training the artificial neural network, 70% of these signals were used for training while the other 30% was used evenly for validation (15%) and training (15%) of the neural network. Each input had a target value of one, this is crucial for the gesture classification. The possible outputs of the artificial neural network were the x amount of different hand gestures.

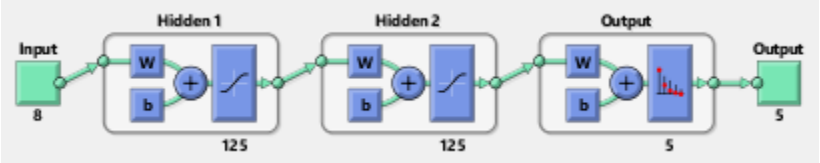


Figure 7. New Artificial Neural Network design

4.2.3. Final American Sign Language Gestures Experiment

Up until the recent isolation due to the coronavirus the experiment has been conducted using only two test subjects. One person being a 6’5”, 268 pounds, 23-year-old male and the other person being a 6’2”, 205 pounds, 23-year-old male. However, it is now only using the latter of the two aforementioned individuals. The armband shall be placed on the right forearm, with the Myo logo positioned on top of the brachioradialis and the subject should stand vertically with the right hand parallel with body pointing towards the ceiling/sky.

4.2.3.1. Neural Network Architecture

An artificial neural network was designed and trained to classify the x amount of selected (3) hand gestures. The artificial neural network was a pattern recognition network with eight inputs (one input for each sEMG sensor on the armband) and x amount of outputs (depending on the number of desired gestures selected by the user (3)). For the testing, two hidden layers comprised of 75 neurons each were used; along with the epochs being set at 150. There were 42,984 total signals (5373*8) used for the input. For training the artificial neural network, 70% of these signals were used for training while the other 30% was used evenly for validation (15%) and training (15%) of the neural network. Each input had a target value of one, this is crucial for the gesture classification. The possible outputs

of the artificial neural network were the x amount of different hand gestures.

4.2.4. Preprocessing and Signal Conditioning

To prepare the data to be entered into the ANN, a preprocessing stage was implemented. When a subject performed a specific gesture for 5 seconds, each EMG sensor was recorded. Then the signal from each channel (sensor) was separated into buffers, each buffer containing 50 samples, therefore a 5 sec length signal contained 20 buffers.

$$\text{Buffers per 5s} = (5 * 200\text{Hz}) / (50 \text{ samples}) = 20$$

Then, each buffer was assigned the corresponding target class and was saved as a sample to be entered in the ANN for training. Using this preprocessing and data conditioning steps, the database generated consisted of a total of 3,674 hand gestures, having an average of 734 data per gesture.

4.2.5. MATLAB Application and Development

An application was created in MATLAB to provide easier interfacing to the armband and artificial neural network. As of now there are three main functions of the app: a data collection process, building/training the artificial neural network function, and a classified gesture output section that also trends the live data from the surface electromyographic sensors on the Myo armband (Fig. 8).

Once the artificial neural network was built, trained, and tested; a function to stream live data from the sensors of the armband to the artificial neural network was created. There are three options for using the live streaming function. The user can either select to only use Arduino control, only use axes plotting, or use both Arduino control and axes plotting. Arduino control is what enables the servo command outputs from MATLAB to the Arduino Leonardo processor. Axes plotting is when the data from the sEMG sensors is plotted depicting the signals in milli-volts with respect to time. The data from the armband was sampled every 350 milliseconds. The classified gesture output (text in red) that is displayed is what will also be the output from the neural network that determines what command to send to the servo motors. For the servo motor control there is a for loop that

was implemented. In this loop the current index of the classified command is compared to the last classified command to see if they are of equal value. If they are then there is no need to repeat the same previous command for that selected motor, so the loop moves on to the next iteration (next motor). Each time the loop is indexed through and issues a valid servo motor command there is a 315 millisecond pause to allow the hardware to catch up to the software. If both plotting axes and Arduino control is enabled, then only one of these actions executes per instance of data sampling in effort to reduce how cumbersome the system can get.

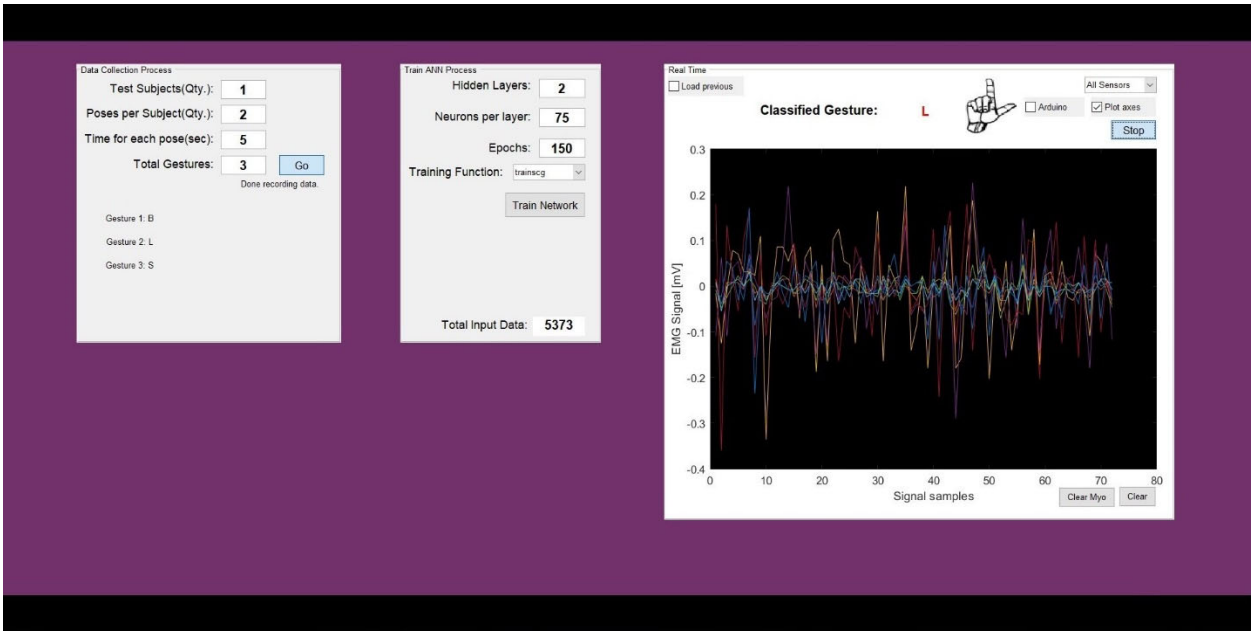


Figure 8. MATLAB Application created for this project

5. PROJECT BREAKDOWN AND TASKS

Our project was broken down into two major categories, hardware & software. There were two programs used for software, MATLAB and Arduino. With MATLAB, we used it for data collection, manipulating the artificial neural network, and classifying the gestures. The hardware consisted of the MYO armband, the LattePanda Delta, and the 3D printed hand. The MYO armband had eight EMG pads that read and collected data from our neural network. It then transmits the data into the MATLAB program using Bluetooth Low-Energy communication protocol via USB dongle. The LattePanda Delta is used to replace an actual desktop PC or laptop. The LattePanda is a miniature computer that is used to run all the programs that are needed to successfully run our project. The LattePanda also has connections like an Arduino board so that it can connect to the servo motors on the artificial hand. The artificial hand is a 3D printed hand that is used to emulate an actual human hand and its movements. The hand's design makes it very realistic. The minimal flaw of the hand is wrist movement as it does not move lateral. The hand was constructed by previous Georgia Southern engineering students.

There were many additional tasks of this project to help facilitate our professional growth and knowledge. For example, we had to write an ethics report and do an ethics presentation. Students also had to not only execute tasks for their projects, but also manage the projects. This was done by setting times and due dates to complete these tasks, which can be observed in the Gantt charts for each respective semester (Fig. 9 & Fig. 10).

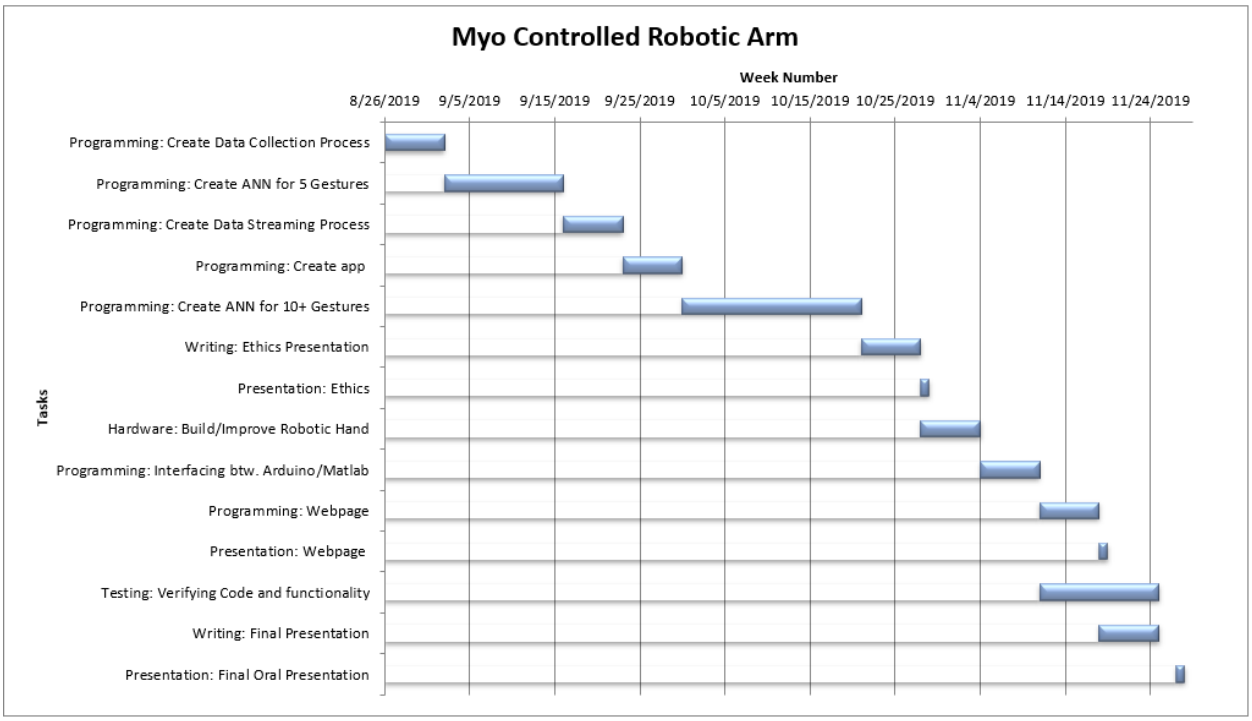


Figure 9. Senior Project I Gantt Chart

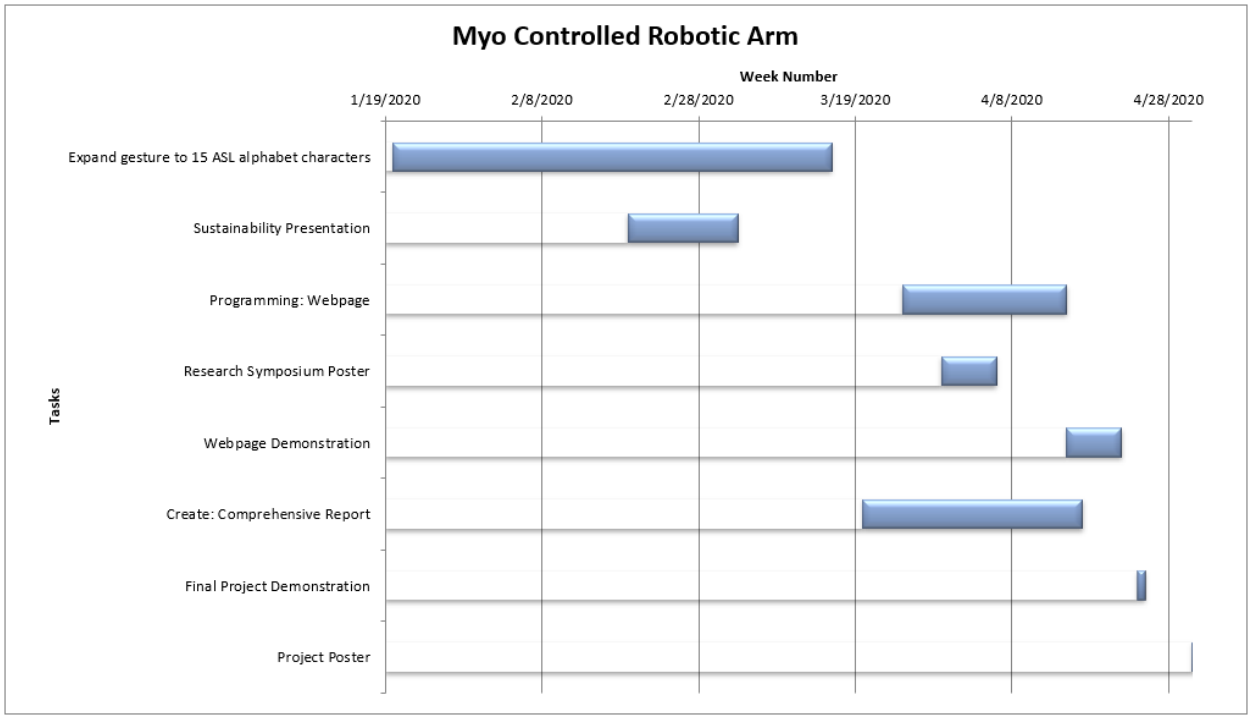


Figure 10. Senior Project II Gantt Chart

6. PROFESSIONAL COMPONENTS

6.1 Economic Concerns

From an economical perspective this project is very friendly, especially when compared to other prosthetics. There are not that many components and they are all relatively cheap. This projects cost does not exceed $700 USD.

6.2 Environmental and Sustainability Concerns

There is the environmental concern with this project being the robotic hands material is all plastic. The production of plastic is not environment friendly. However, there is the alternative of using recycled material which will help the environment.

6.3 Manufacturing ability Concerns

This prototype was hand crafted individually using a 3D printer. However, I believe there is potential for mass production using molds or casts for many of the hand components. Also, it is plausible for manufacturing to use 3D printing. For example, an aerospace company that makes parts for jet engines used a special 3D printer to print out a super nickel alloy part that is normal forged.

6.4 Ethical, Legal, and Health Concerns

This project is ethical and the device it legal. The process or application would be patentable, although the Myo armband was designed by Thalmic Labs and the hand was designed by InMoov so those items are already patented. Regarding the health aspect of this project there are no concerns except for the data aspect. The user must understand that collection of data (sEMG signals) are required for this process and they must agree to it.

7. RESULTS

7.1 Original Gestures Experiment

From the validation check-fail plot of the neural network it can be observed that there were only a handful of validation failures that exceeded two for each individual epoch (Fig. 11). These uncharacteristic errors occurred in the range of 300-350 epochs.

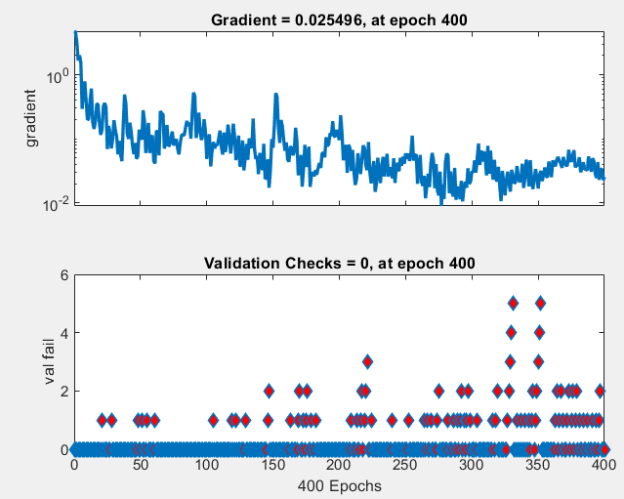


Figure 11. Validation Fail Check plot of the trained ANN

The performance of the artificial neural network was calculated using the cross-entropy function (Fig. 12). This function operates by penalizing errors in classification [8].

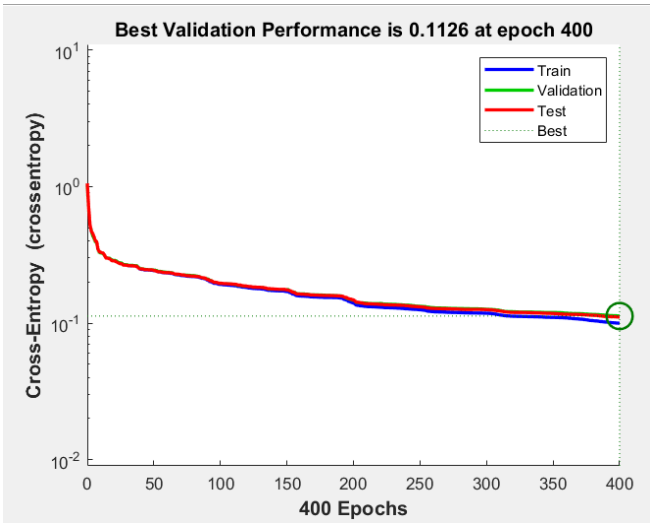


Figure 12. Performance plot of the trained ANN

The results of the accuracies from the trained neural network are positive results considering two different people were involved and not just one. Adding more people creates more diversity of the data in the database, which leads to greater room for error or confusion by the artificial neural network. The five output classes from class one to class five were the five original gestures: rest position, making a fist, spreading all five fingers out, waving the hand in (flex at the wrist), and waving the handout.

From the confusion matrix (Fig. 13) it may be observed that the overall accuracy of the trained neural network was 79.5%. Class one (hand in rest position) had a slightly below average accuracy rate at 73.5%. This could be due to the overall magnitude of the signals from the electromyographic sensors is relatively low since there is no major muscle contraction. Class two (making a fist) had a very good accuracy rate of 98.5%. Whereas class three (spreading of the fingers) had a very low accuracy rate of 65.9%. This could be in part of the similarities of the muscle contraction between making a fist and spreading of the fingers. Based off the data in the confusion plot, the output class four was classified as the target of class three and vise-versa. This demonstrates a similarity in the muscle group contraction.

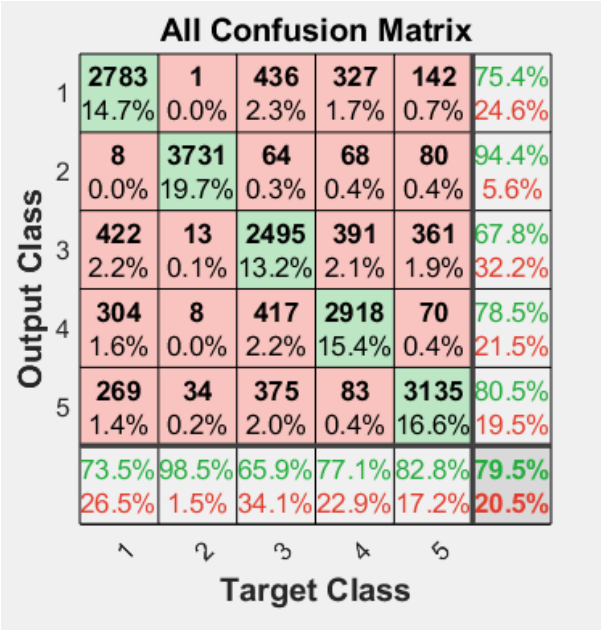


Figure 13. Confusion matrix result from the trained ANN

7.2 First American Sign Language Gestures Experiment

The data provided to the neural network for training and testing was the result of recording five gestures for ten seconds each. The five gestures for this specific experiment where the letters A, C, F, W, & Y of the American Sign Language (Fig. 14). The results of the accuracies from the trained neural network are significantly lower compared to previous tests that involved gestures that required a significant amount more of muscle flexion in the forearm.

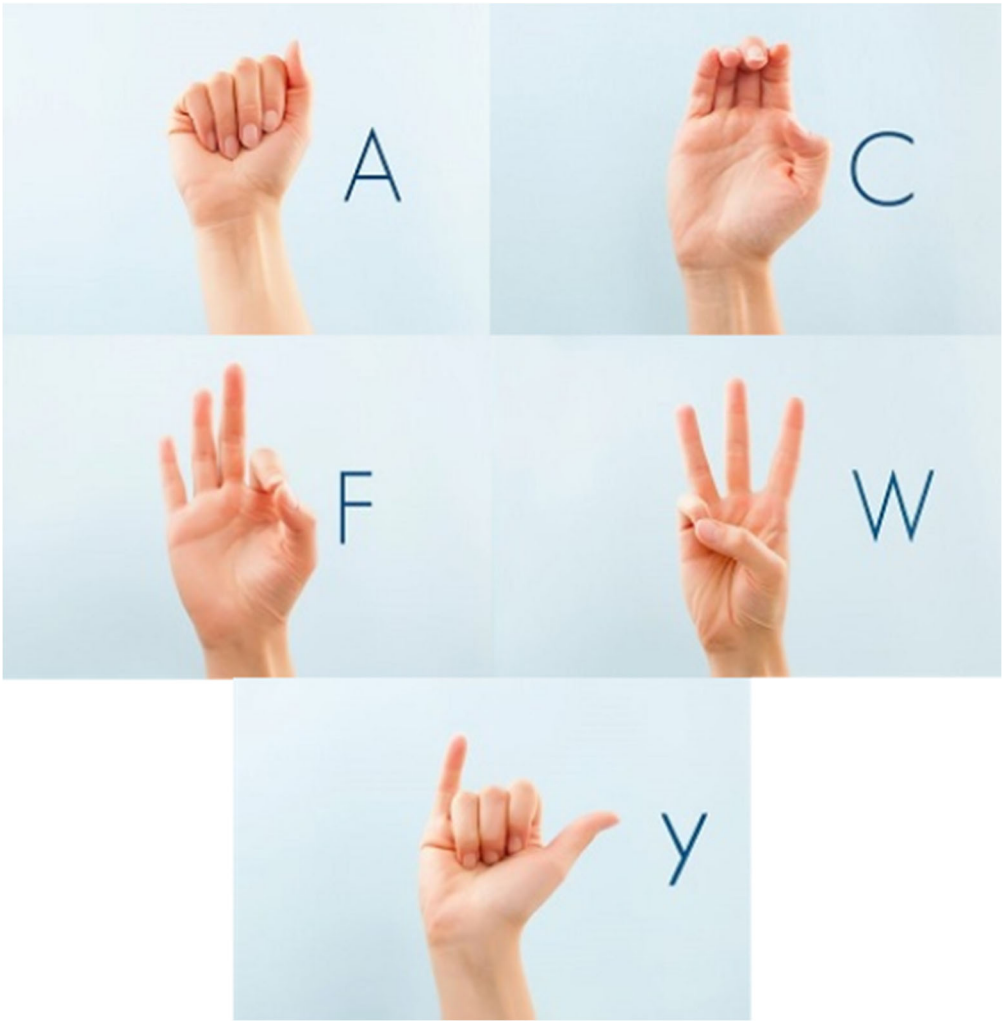


Figure 14. Five American Sign Language hand gestures used

The neural network used the “dividerand” algorithm for data division, “scaled conjugate gradient” algorithm for training the network, and “crossentropy” algorithm for the network’s performance. For this test, the network was able to reach the maximum epochs (400 iterations) without reaching the maximum validation checks that was set at fifty (Fig. 15).

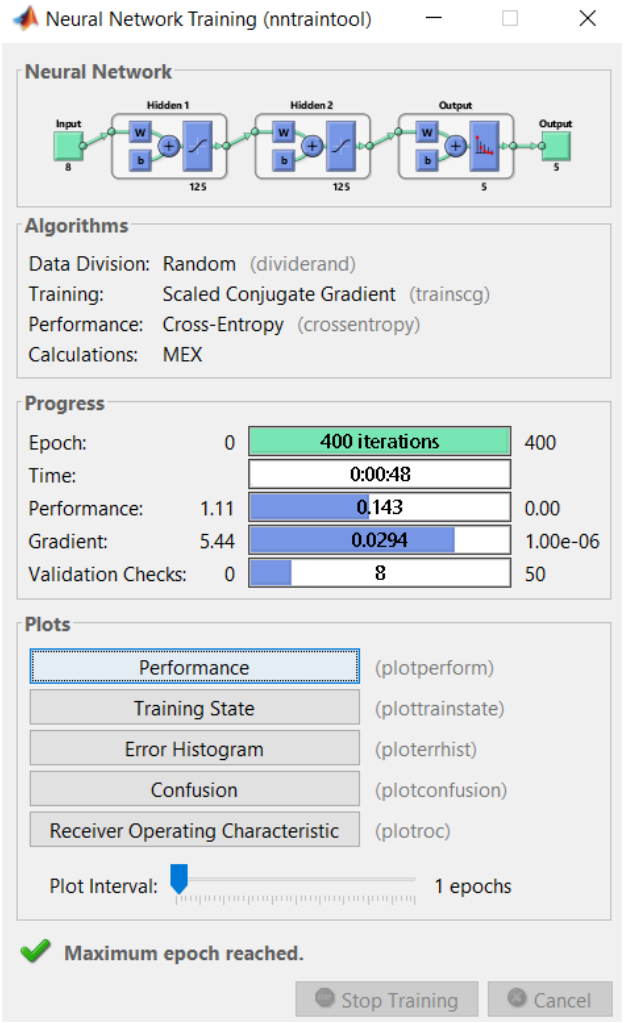


Figure 15. NN Tool displaying data from trained ANN

From the confusion matrix (Fig. 16) it may be observed that the overall accuracy of the trained neural network was 68.8%. Class one (gesture A) had the highest accuracy rate at 73.5%. Class two (gesture C) had an accuracy rate of 69.8%. Class three (gesture F) had an accuracy rate of 67.4%. Class four (gesture W) had an accuracy rate of 73.1%. Class five (gesture Y) had the worst accuracy rate at 60.6%.

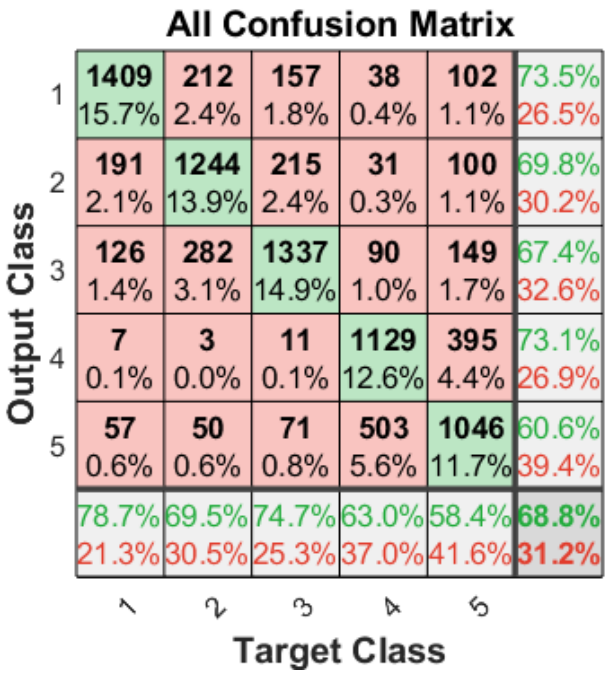


Figure 16. Trained ANN Confusion Matrix Plot

Looking at the confusion matrix again, one can see that the neural network had significant difficulty distinguishing the difference between classes four and five. This may also be observed in the stem plot (Fig. 17) where the red represents incorrectly targeted outputs and the green shows the outputs that were correctly targeted.

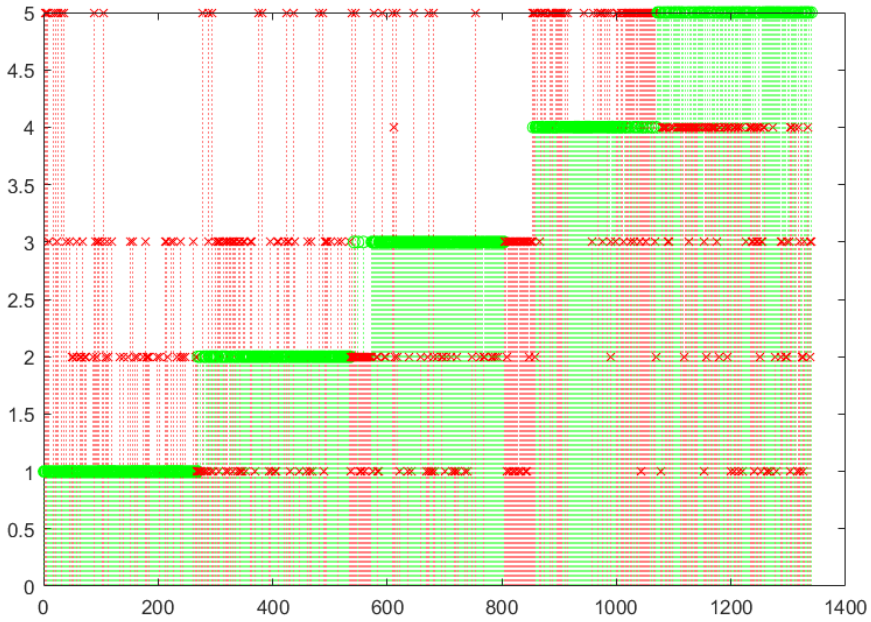


Figure 17. Stem plot showing the classified outputs

7.3 Final American Sign Language Gestures Experiment

The data provided to the neural network for training and testing for the final experiment was the result of recording three gestures two times for five seconds each (Fig. 18). The three gestures for this specific experiment where the letters B, S, & L of the American Sign Language Alphabet (Fig. 19). The results of the accuracies from the trained neural network are significantly higher (12.8% increase in accuracy) compared to the previous tests that involved five gestures of the American Sign Language Alphabet

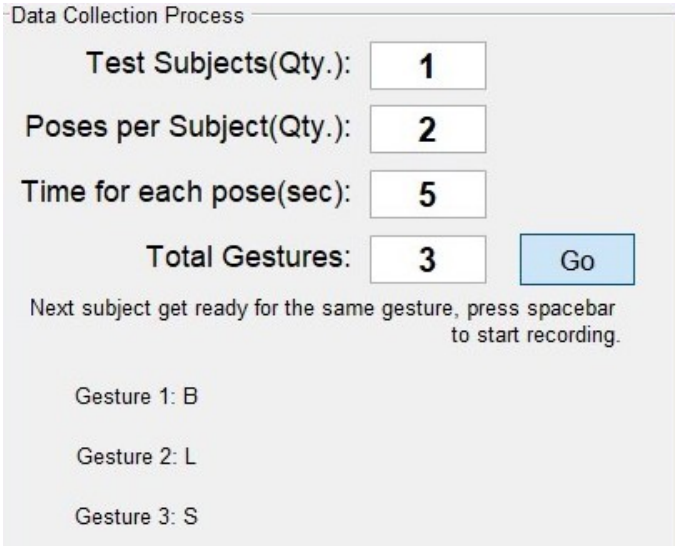


Figure 18. Data collection process within the MATLAB application

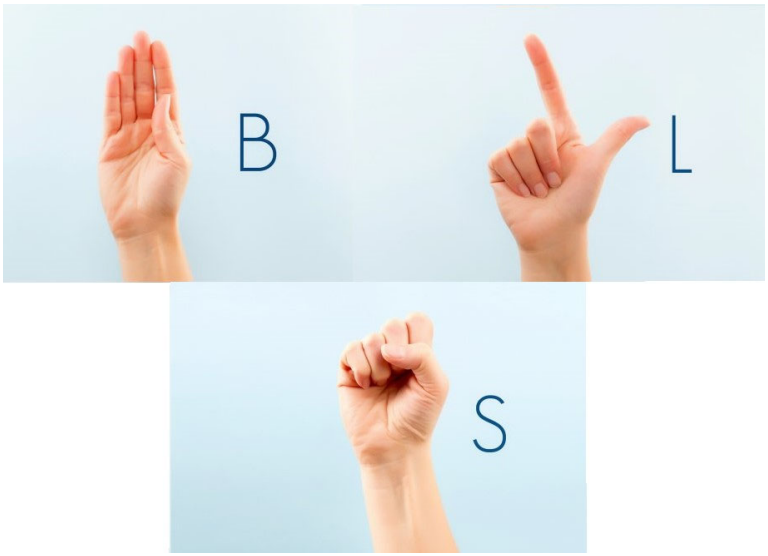


Figure 19. Three American Sign Language hand gestures used

The neural network used the “dividerand” algorithm for data division, “scaled conjugate gradient” algorithm for training the network, and “crossentropy” algorithm for the network’s performance. For this test, the network was able to reach the maximum epochs (150 iterations) without reaching the maximum validation checks that was set at fifty (Fig. 20).

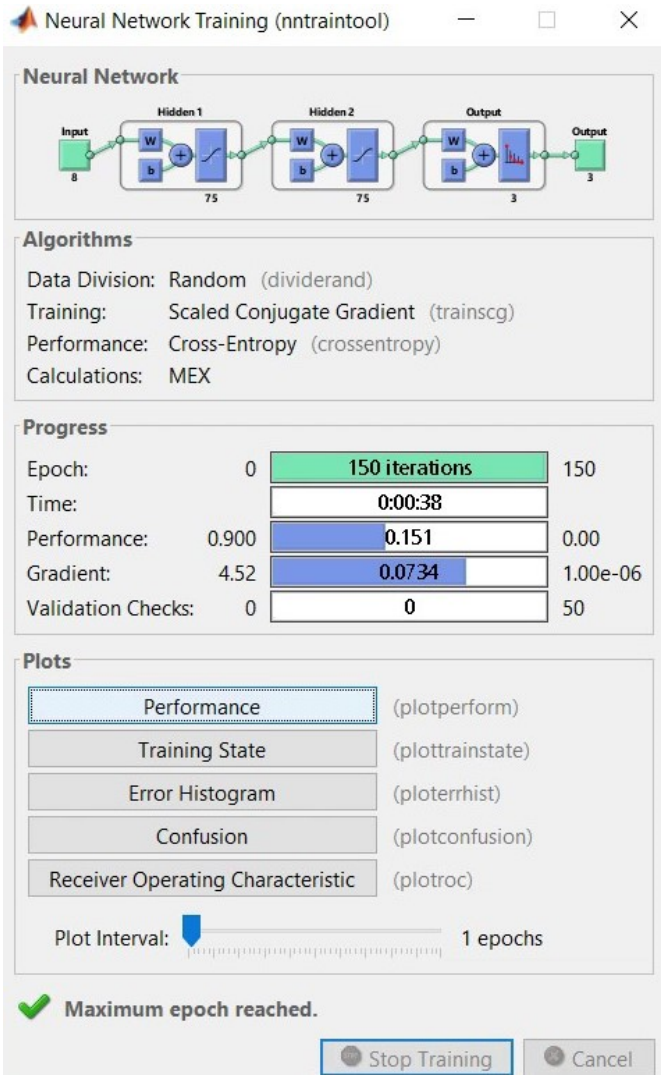


Figure 20. NN Tool displaying data from trained ANN

From the confusion matrix (Fig. 21) it may be observed that the overall accuracy of the trained neural network was 81.6%. Class one (gesture B) had the highest accuracy rate at 91.4%. Class two (gesture L) had an accuracy rate of 70.0%. Class three (gesture S) had an accuracy rate of 87.8%.

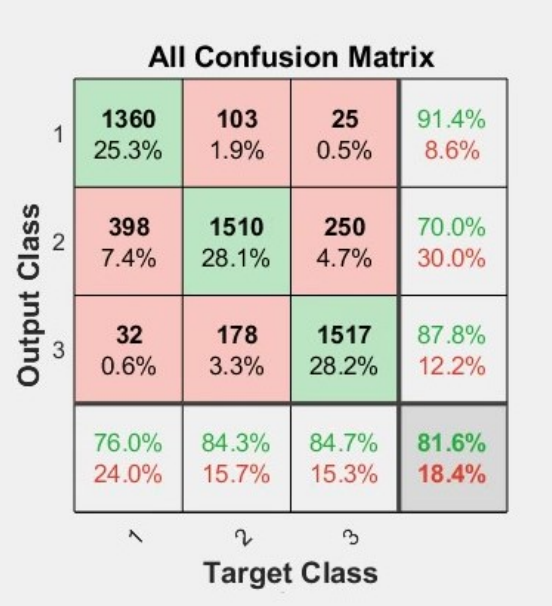


Figure 21. Trained ANN Confusion matrix

Looking at the confusion matrix again, one can see that the neural network had significant difficulty distinguishing the difference between class two and classes four and five. This may also be observed in the stem plot (Fig. 22) where the red represents incorrectly targeted outputs and the green shows the outputs that were correctly targeted.

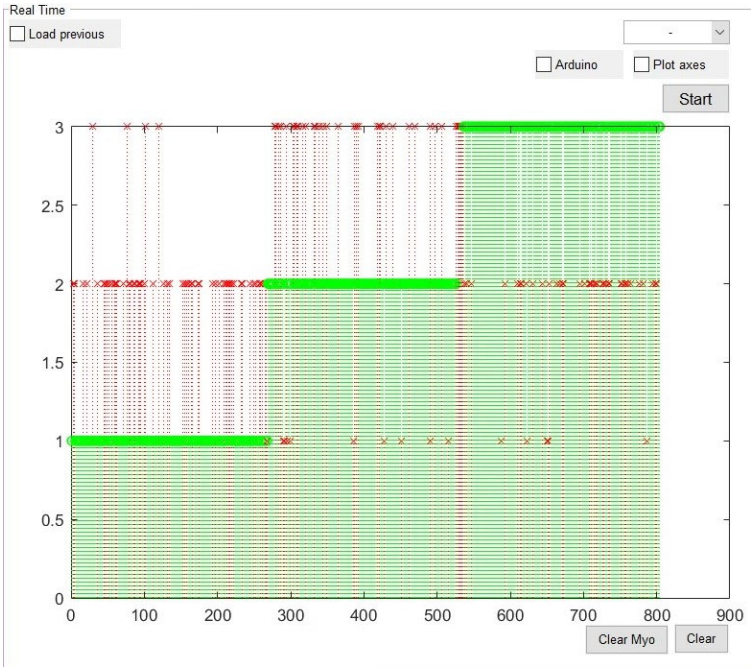


Figure 22. Stem plot showing the classified outputs

Upon the completion of collecting data and training the artificial neural network, the live streaming and gesture classification portion of the simulation was performed. There are three options to do this live simulation. The user has the option to only use Arduino control (Fig. 23), only use axes plotting (Fig. 24), or use axes plotting and Arduino control. However, it is not recommended to use both the Arduino control and axes plotting at the same time because this is very cumbersome and will result in poorer system results.

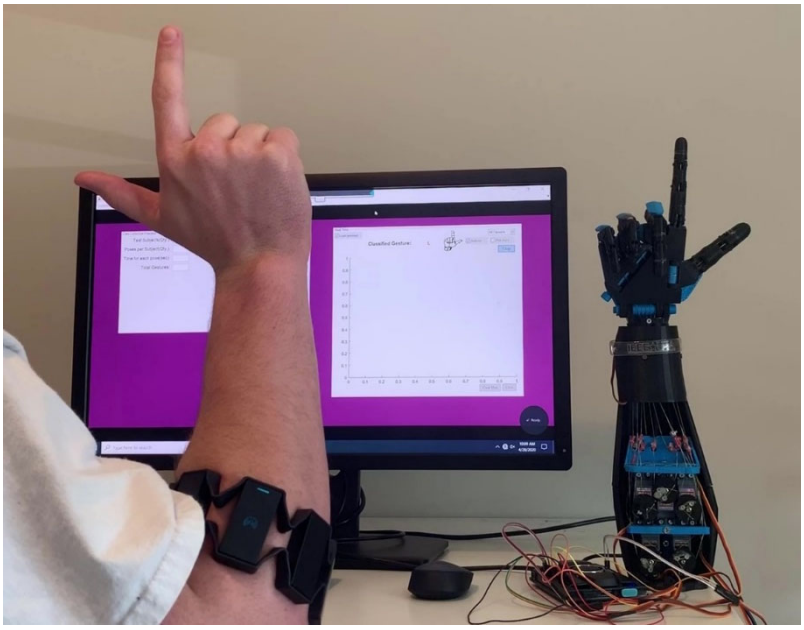


Figure 23. Real-Time gesture classification with Arduino control enabled

Figure 24. Real-Time gesture classification with axes plotting enabled

8. DELIVERABLES

8.1 Research Symposium Poster

109

MYO Armband Controlled Robotic Arm

Jheric Byrd and Zackary Minshew
Department of Electrical and Computer Engineering

Faculty Mentor: Dr. Fernando Rios-Gutierrez

Introduction.

In today's society it is apparent that the role and use of robotics and process automation is becoming increasingly more prominent in various fields and applications, from self-driving automobiles to various types of manufacturing. The expansion of these fields is driven by the persistent insatiable consumer market, not just in the United States, but in the world. This desire of advancement is typically looking for a means to reduce work or cost. Such as the advancement of robotics in manufacturing that leads to reduced labor and the liability of that labor. A byproduct of reducing the amount of human labor is usually higher profits and lower liability (insurance and safety). Another desire for these advancements is to reduce error, which could be done by a more advanced autonomously driving vehicle; or to be able to provide someone who has lost a limb with a fully functional prosthetic that emulates a human limb very well.

Implementation.

This project explores the use of the eight surface electromyographic sensors (sEMG) on the Myo® Armband (*Thalmic Labs*) that provide the data to train a feed-forward pattern recognition artificial neural network. The LattePanda Delta ultimate mini-board (*DFRobot*) is used to execute the classified output from the neural network as a physical output to control a 3D printed robotic hand. This project demonstrates cost alternative options to that of advanced medical prosthetics while saving a significant amount of money.

Background.

An electromyographic signal is a signal that is transmitted from the motor neurons, which send electrical signals to the respective muscle(s) causing it/them to contract or relax. EMG signals provide diagnostics of the command signals for muscles. This information is instrumental in helping doctors to diagnose and help patients. There are two different types of electromyographic signal. One type is placed on the surface of the skin, hence surface electromyographic sensor. The other, more invasive type is to puncture the skin with needles and to stick the needles into muscle tissue to record muscle activity [1].

For this project the Myo armband was used to acquire the sEMG signals. The armband allowed for advanced data collection with a simple means of obtaining that data. The Myo has eight surface electromyographic sensors that are evenly displaced. The signals from these sensors can be obtained from a computer via BLE dongle (*Bluetooth low-energy communication protocol*).

Artificial Neural Networks are computerized systems that model the neural network structure of animalia brains. Neural networks can be used for the recognition of patterns, prediction of future events based off past trends, and for the classification of data and information by learning from historical data [2]. For this project the Deep-Learning Toolbox in MATLAB is used to design, train, and classify the selected hand gestures.

Methods.

A. Experimental Design Process

Up until the recent isolation due to the coronavirus the experiment has been conducted using only two test subjects. One person being a 6'5", 268 pounds, 23-year-old male and the other person being a 6'2", 205 pounds, 23-year-old male. However, it is now only using the latter of the two aforementioned individuals. The armband shall be placed on the right forearm, with the Myo logo positioned on top of the brachioradialis and the subject should stand vertically with the right hand parallel with body pointing towards the ceiling/sky.

B. Signal Conditioning and Data Processing

The artificial neural network required a buffer to be implemented for the data collection process because the startup time was inconsistent which led to variations of signal values in the database. This was implemented by a simple preprocessing task that would run two timed trials of the data collection process upon start up, and then deleting that gathered data because it is not desired.

C. Artificial Neural Network Architecture

An artificial neural network was designed and trained to classify the x amount of selected (5) hand gestures. The artificial neural network was a pattern recognition network with eight inputs and x amount of outputs (depending on the number of desired gestures selected by the user (5)). For the testing, two hidden layers comprised of 125 neurons each were used; along with the epochs being set at 400 (Fig. 1). There were 71,640 total signals (*8955*8*) used for the input. For training the artificial neural network, 70% of these signals were used for training while the other 30% was used evenly for validation (*15%*) and training (*15%*) of the neural network. Each input had a target value of one, this is crucial for the gesture classification. The possible outputs of the artificial neural network were the x amount of different hand gestures.

Figure 1.) ANN Architecture

D. Application Development and Implementation

An application was created in MATLAB to provide easier interfacing to the armband and artificial neural network. As of now there are three main functions of the app: a data collection process, building/training the artificial neural network function, and a classified gesture output section that also trends the live data from the surface electromyographic sensors on the Myo armband (Fig. 2).

Figure 2.) App created for this project

Once the artificial neural network was built, trained, and tested; a function to stream live data from the sensors of the armband to the artificial neural network was created. The data from the armband was sampled every 350 milliseconds. The classified gesture output (text in red) that is displayed is what will also be the output from the neural network that determines what command to send to the servo motors that control the hand.

Results.

The data provided to the neural network for training and testing was the result of recording five gestures for ten seconds each. The five gestures for this specific experiment where the letters A, C, F, W, & Y of the American Sign Language. The results of the accuracies from the trained neural network are significantly lower compared to previous tests that involved gestures that required a significant amount more of muscle flexion in the forearm.

The neural network used the "dividerand" algorithm for data division, "scaled conjugate gradient" algorithm for training the network, and "crossentropy" algorithm for the network's performance. For this test, the network was able to reach the maximum epochs (400 iterations) without reaching the maximum validation checks that was set at fifty (Fig. 3).

Figure 3.) NN Tool displaying data from trained ANN

From the confusion matrix (Fig. 4) it may be observed that the overall accuracy of the trained neural network was 68.8%. Class one (gesture A) had the highest accuracy rate at 73.5%. Class two (gesture C) had an accuracy rate of 69.8%. Class three (gesture F) had an accuracy rate of 67.4%. Class four (gesture W) had an accuracy rate of 73.1%. Class five (gesture Y) had the worst accuracy rate at 60.6%.

Figure 4.) Trained ANN Confusion matrix

Looking at the confusion matrix again, one can see that the neural network had significant difficulty distinguishing the difference between classes four and five. This may also be observed in the stem plot (Fig. 5) where the red represents incorrectly targeted outputs and the green shows the outputs that were correctly targeted.

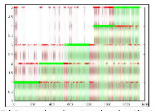


Figure 5.) Stem plot showing the classified outputs

Future Work.

The hopes of this project are to continue to explore the cost alternative options for human prosthetic control via live human body signals (EMG in this case). Upon the completion of adding more gestures a goal would be to incorporate the use of the inertial measurement unit that has accelerometers and a gyroscope for the sensing components.

Discussion.

- One of the top findings was the significant difference in accuracy between gestures requiring more forearm muscle flexion such as waving the hand in or making a fist versus gestures for alphabetic letters of the American Sign Language, which do not necessarily require much muscular force from the forearm.
- This is significant because it exposes a weak point if this was to be used for something such as sign language. If this was used in real life, there would be more error using it to perform sign language versus using it to grab an object or to hold something that requires a distinct force of the forearm muscle.
- This work is important because it is using open source technology and cheaper resources to show a possible alternative of low to middle end prosthetics that still cost anywhere from $5000-$25,000. This project's cost was approximately $500, which is significantly cheaper than any medical prosthetic currently on the market today.

We would like to thank Dr. Rios and the entire Electrical and Computer Engineering Department faculty for all the knowledge, guidance, and support in which they provided us throughout our time here at Georgia Southern University.

References

[1] William Morrison, "Electromyography (EMG)". Internet: https://www.healthline.com/health/electromyography. 2018 [Accessed Mar. 23, 2020].

[2] "What Is a Neural Network?". Internet: https://www.mathworks.com/discovery/neural-network.html. 2019 [Accessed Mar. 23, 2020].

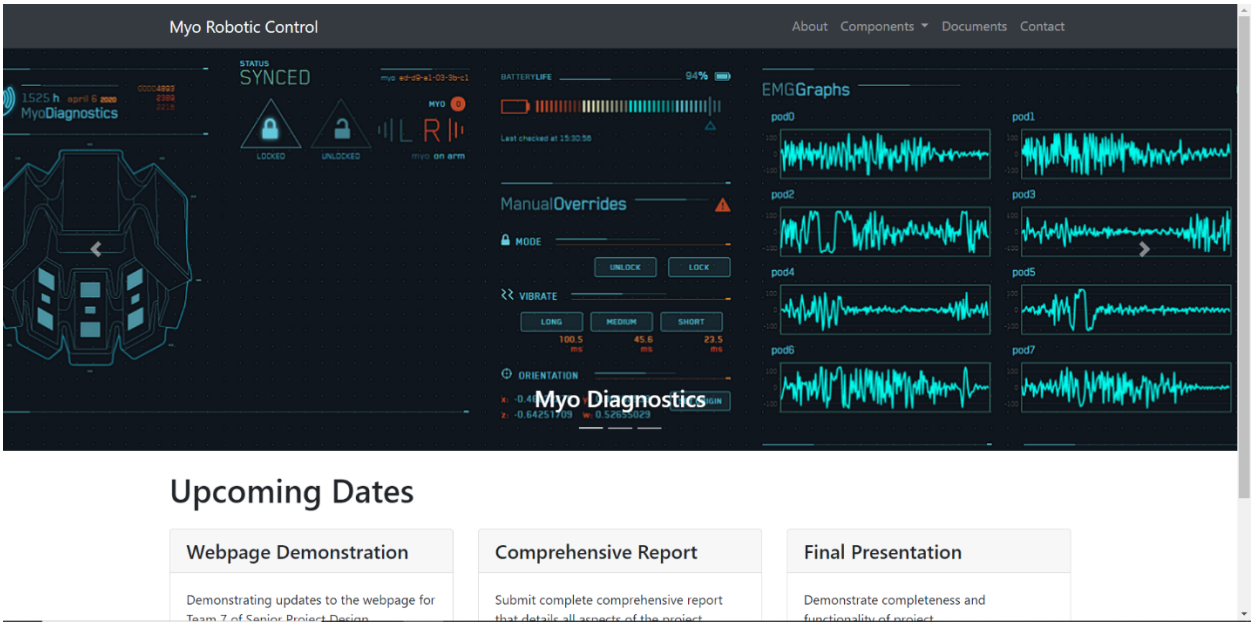
GEORGIA SOUTHERN UNIVERSITY
ALLEN E. PAULSON COLLEGE OF ENGINEERING AND COMPUTING

2020 Student Research Symposium
Statesboro, Georgia
April 24, 2020

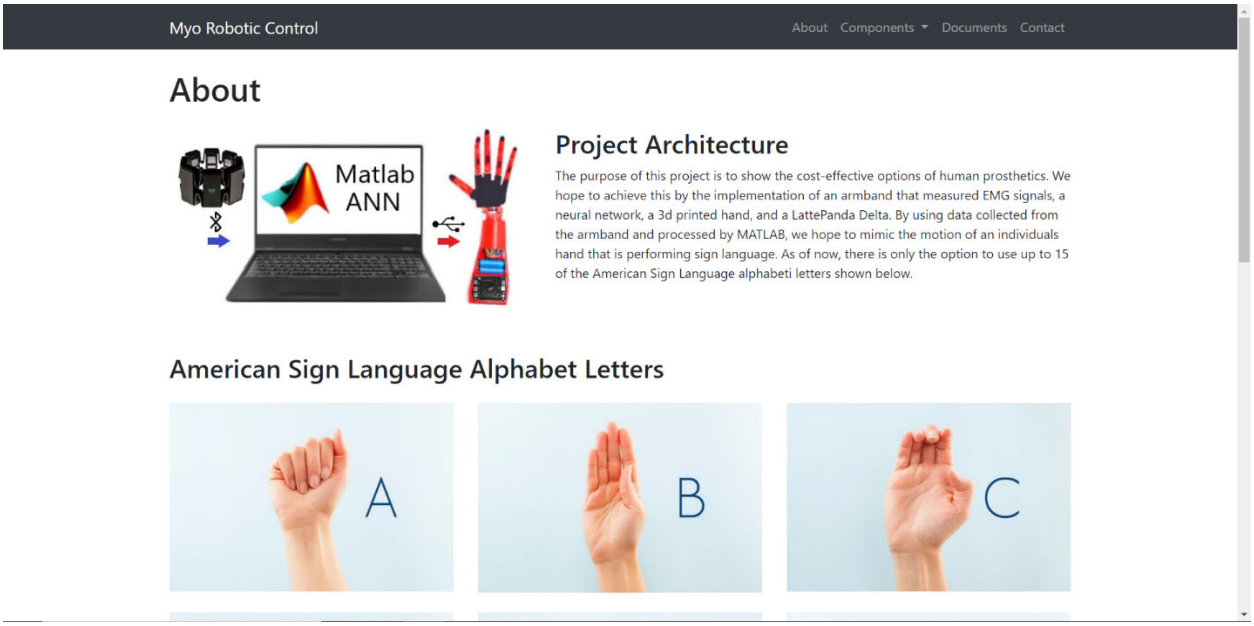
GEORGIA SOUTHERN UNIVERSITY

8.2 Project Webpage

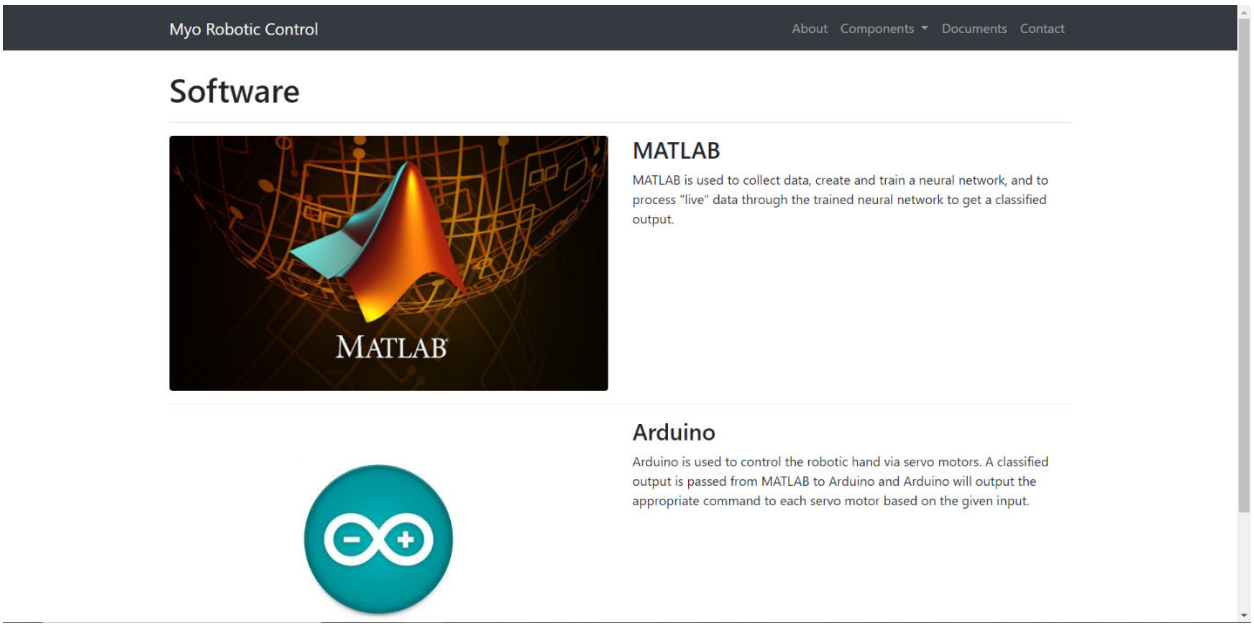
8.2.1. Home Page



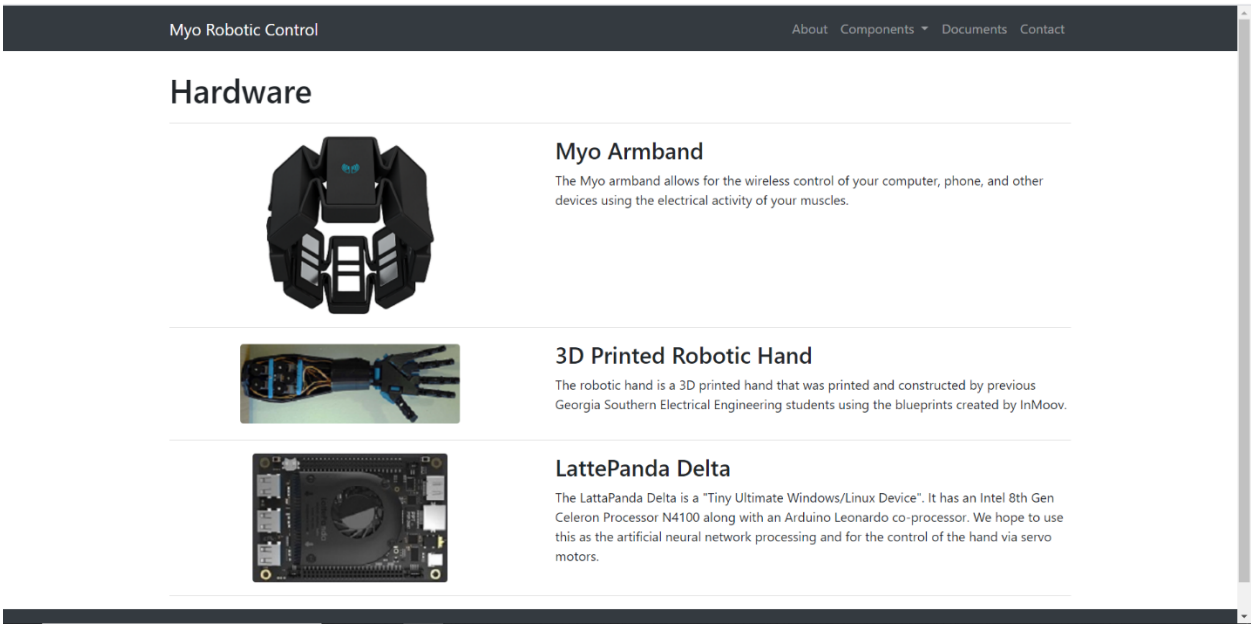
8.2.2. About Page



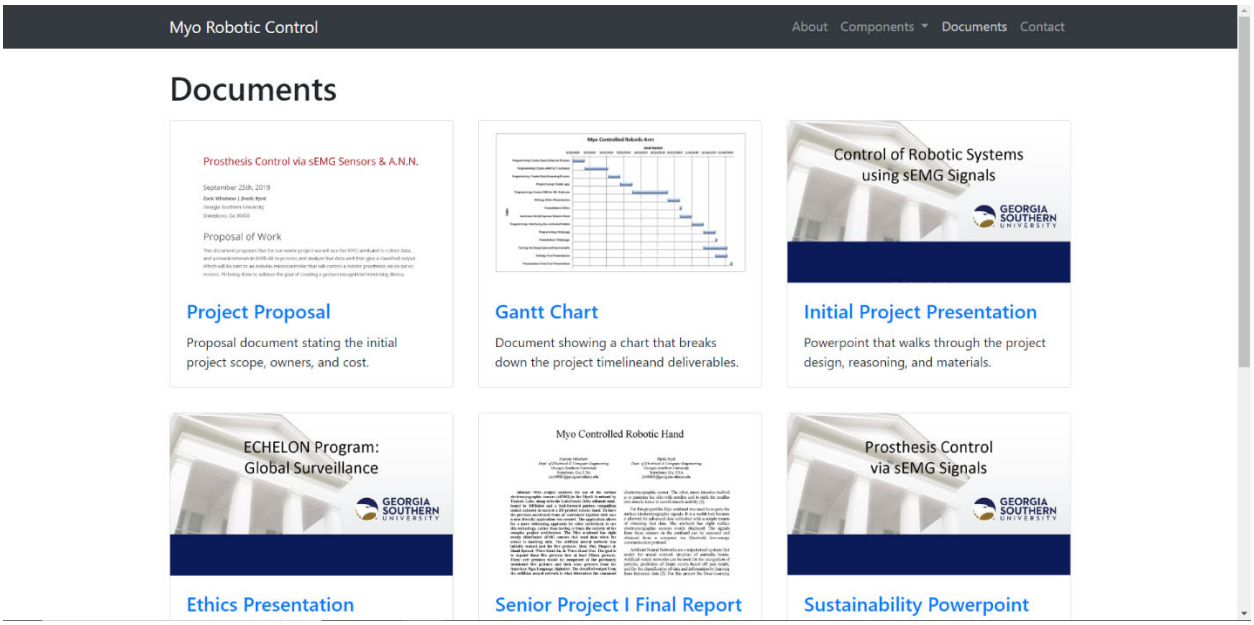
8.2.3. Components: Software Page



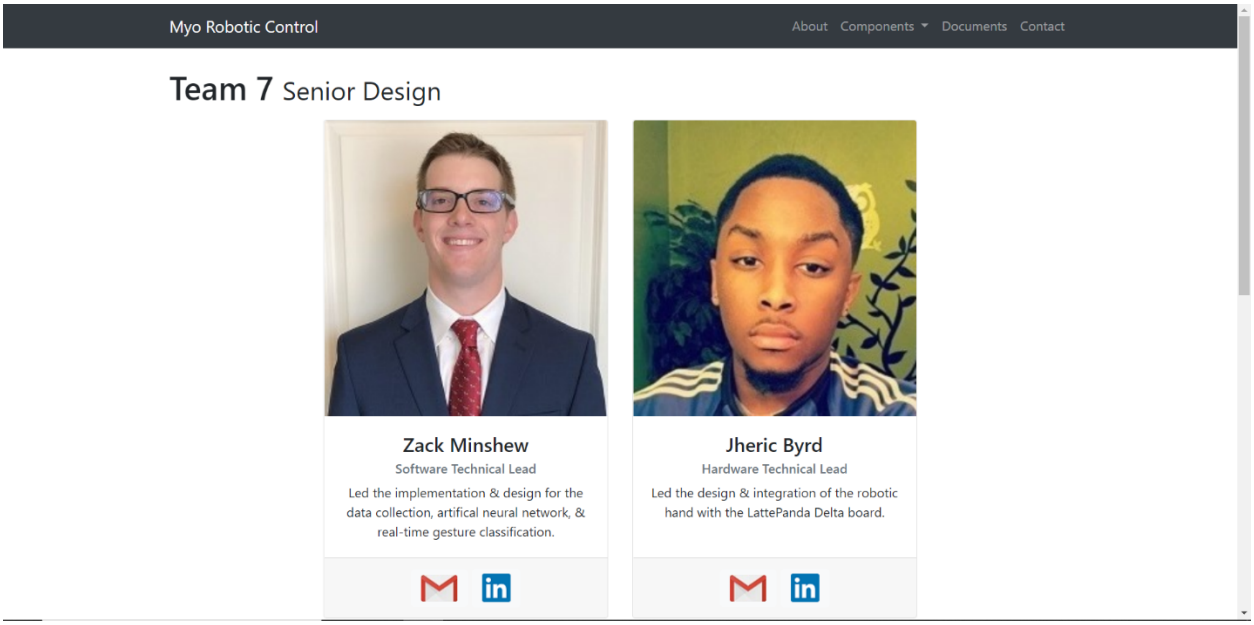
8.2.4. Components: Hardware Page



8.2.5. Documents



8.2.6. Contact Page



8.3 MATLAB Code

8.3.1. Custom Application

```
function varargout = app(varargin)
% APP MATLAB code for app.fig
% Edit the above text to modify the response to help app
% Last Modified by GUIDE v2.5 27-Apr-2020 18:23:02
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @app_OpeningFcn, ...
                   'gui_OutputFcn',  @app_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before app is made visible.
function app_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to app (see VARARGIN)

% Choose default command line output for app
handles.output = hObject;
set(hObject, 'WindowState','maximized')
```


```
set(hObject,'Name','Robotic Control using MYO & sEMG','NumberTitle','off');

list = {'popupmenu25', 'popupmenu26', 'popupmenu27', 'popupmenu28', 'popupmenu29', 'popupmenu30',
...
    'popupmenu31', 'popupmenu32', 'popupmenu33', 'popupmenu34', 'popupmenu35', 'popupmenu36',
...
    'popupmenu37', 'popupmenu38', 'popupmenu39', 'popupmenu40', 'text8', 'text32', 'text33',  ...
    'text34', 'text35', 'text36', 'text37', 'text38', 'text39', 'text40', 'text41', 'text42', ...
    'text43', 'text44', 'text45','text46','text47', 'axes11', 'togglebutton2'};

for index = 1:length(list)
set(handles.(list{index}),'visible','off');
end

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes app wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = app_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;



function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```


```
% Hints: get(hObject,'String') returns contents of edit1 as text
%        str2double(get(hObject,'String')) returns contents of edit1 as a double

% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit2_Callback(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes during object creation, after setting all properties.
function edit2_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit3_Callback(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes during object creation, after setting all properties.
function edit3_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
```


```
end

function edit4_Callback(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes during object creation, after setting all properties.
function edit4_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
% Hints: get(hObject,'String') returns contents of edit5 as text
%        str2double(get(hObject,'String')) returns contents of edit5 as a double

function edit5_Callback(hObject, eventdata, handles)
% hObject    handle to edit5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes during object creation, after setting all properties.
function edit5_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit7_Callback(hObject, eventdata, handles)
% hObject    handle to edit7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes during object creation, after setting all properties.
function edit7_CreateFcn(hObject, eventdata, handles)
```


```
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

m = str2num(get(handles.edit1,'String')); %get the value of from edit1
n = str2num(get(handles.edit2,'String'));  %get the value of from edit2
o = str2num(get(handles.edit3,'String'));   %get the value of from edit3
pArr = cellstr(get(handles.popupmenu40,'String'));

idx = get(handles.popupmenu1,'Value');
items = get(handles.popupmenu1,'String');
q = items{idx};

p = TrainANN(m,n,o,q,pArr,handles); %get p from TrainANN
axes(handles.axes1)

set(handles.text10,'String',p) %set p in text10

% --- Executes on button press in togglebutton1.
function togglebutton1_Callback(hObject, eventdata, handles)
% hObject    handle to togglebutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of toggl
button_state = get(hObject,'Value');
if button_state == 1
    set(hObject,'String','Stop');
 else
    set(hObject,'String','Start');
end
```


```
set(handles.axes11,'visible','on');
str = get(handles.popupmenu2,'Value');
val = get(handles.popupmenu2,'String');
r = val{str};

boxP = get(handles.checkbox11,'Value');
boxA = get(handles.checkbox12,'Value');
boxR = get(handles.checkbox13,'Value');

cla(handles.axes1)  %clear axes
axes(handles.axes11)
axis off

posAr = cellstr(get(handles.popupmenu40,'String'));
% Save the handles structure.
guidata(hObject,handles)

%call the function LiveStreaming
[a,b] = LiveStreaming(r,posAr,boxP,boxA,boxR,handles)

if boxP == 1
axes(handles.axes1)
plot(b);
end

%set a in the textbox
set(handles.text6,'String','Classified Gesture:')
set(handles.text7,'String',a)


% --- Executes on button press in togglebutton2.
function togglebutton2_Callback(hObject, eventdata, handles)
% hObject    handle to togglebutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of togglebutton2
```


```
numPos = str2num(get(handles.edit14,'String'));      % # test subjects
numPeop = str2num(get(handles.edit12,'String'));     % poses per subject
time = str2num(get(handles.edit13,'String'));        % time
posArr = cellstr(get(handles.popupmenu40,'String')); % cell array of selected poses
%posTotal = str2num(get(handles.edit18,'String'));

%[strng,ii,done] = DataCollector(time,numPos,numPeop,posTotal,handles);
[strng,ii,done] = DataCollector(time,numPos,numPeop,posArr,handles);

set(handles.text16,'String',strng)

if done == 1
   set(handles.text8,'String',num2str(ii))
else
   set(handles.text8,'String',"")
end

% --- Executes on button press in pushbutton4.
function pushbutton4_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
KLEAR;

set(handles.edit1,'String',' ')
set(handles.edit2,'String',' ')
set(handles.edit3,'String',' ')
set(handles.edit12,'String',' ')
set(handles.edit13,'String',' ')
set(handles.edit14,'String',' ')
set(handles.edit18,'String',' ')
set(handles.text7,'String',' ')
set(handles.text8,'String',' ')
set(handles.text10,'String',' ')
set(handles.text16,'String',' ')
cla(handles.axes1)  %clear axes
```


```
axes(handles.axes11)
axis off

% --- Executes on selection change in popupmenu1.
function popupmenu1_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes during object creation, after setting all properties.
function popupmenu1_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in pushbutton7.
function pushbutton7_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
KLEAR2;


% --- Executes on selection change in popupmenu2.
function popupmenu2_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes during object creation, after setting all properties.
function popupmenu2_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```


```
% --- Executes on selection change in popupmenu25.
function popupmenu25_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu25 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns popupmenu25 contents as cell array
%        contents{get(hObject,'Value')} returns selected item from popupmenu25
List = {'popupmenu26', 'popupmenu27', 'popupmenu28', 'popupmenu29', 'popupmenu30', 'popupmenu31',
...
       'popupmenu32', 'popupmenu33', 'popupmenu34', 'popupmenu35', 'popupmenu36', 'popupmenu37',
...
       'popupmenu38', 'popupmenu39'};

sel = cellstr(get(hObject,'String'));
val = get(hObject,'value');
if val > 1
var = sel{val};

arr{1} = var;
str = char(arr);
set(handles.popupmenu40,'String',str)

set(handles.popupmenu25,'visible','off')
set(handles.text32,'visible','on')
set(handles.text32,'String',strcat("Gesture 1: ",var))

sel{val} = num2str([]);
A =  sel(~cellfun('isempty', sel));

for index = 1:length(List)
set(handles.(List{index}),'String',A);
end

h = str2num(get(handles.edit18,'String'));
if h > 1
    set(handles.popupmenu26,'visible','on')
```


```
else
    set(handles.togglebutton2,'visible','on')
end
else
end

% --- Executes during object creation, after setting all properties.
function popupmenu25_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes on selection change in popupmenu26.
function popupmenu26_Callback(hObject, eventdata, handles)

List = {'popupmenu27', 'popupmenu28', 'popupmenu29', 'popupmenu30', 'popupmenu31', ...
        'popupmenu32', 'popupmenu33', 'popupmenu34', 'popupmenu35', 'popupmenu36', 'popupmenu37',
...
        'popupmenu38', 'popupmenu39'};
sel = cellstr(get(hObject,'String'));
val = get(hObject,'value');
if val > 1
var = sel{val};

arr = cellstr(get(handles.popupmenu40,'String'));
arr{2} = var;
str = char(arr);
set(handles.popupmenu40,'String',str)

set(handles.popupmenu26,'visible','off')
set(handles.text33,'visible','on')
set(handles.text33,'String',strcat("Gesture 2: ",var))

sel{val} = num2str([]);
A =  sel(~cellfun('isempty', sel));
```


```
for index = 1:length(List)
set(handles.(List{index}),'String',A);
end

h = str2num(get(handles.edit18,'String'));
if h > 2
   set(handles.popupmenu27,'visible','on')
else
   set(handles.togglebutton2,'visible','on')
end
else
end

% --- Executes during object creation, after setting all properties.
function popupmenu26_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end



% --- Executes on selection change in popupmenu27.
function popupmenu27_Callback(hObject, eventdata, handles)

List = {'popupmenu28', 'popupmenu29', 'popupmenu30', 'popupmenu31','popupmenu32', 'popupmenu33',
...
      'popupmenu34', 'popupmenu35', 'popupmenu36', 'popupmenu37','popupmenu38', 'popupmenu39'};

sel = cellstr(get(hObject,'String'));
val = get(hObject,'value');
if val > 1
var = sel{val};

arr = cellstr(get(handles.popupmenu40,'String'));
arr{3} = var;
str = char(arr);
```


```
set(handles.popupmenu40,'String',str)

set(handles.popupmenu27,'visible','off')
set(handles.text34,'visible','on')
set(handles.text34,'String',strcat("Gesture 3: ",var))
sel{val} = num2str([]);
A =  sel(~cellfun('isempty', sel));

for index = 1:length(List)
set(handles.(List{index}),'String',A);
end

h = str2num(get(handles.edit18,'String'));
if h > 3
   set(handles.popupmenu28,'visible','on')
else
   set(handles.togglebutton2,'visible','on')
end
else
end

% --- Executes during object creation, after setting all properties.
function popupmenu27_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end



% --- Executes on selection change in popupmenu28.
function popupmenu28_Callback(hObject, eventdata, handles)

List = {'popupmenu29', 'popupmenu30', 'popupmenu31','popupmenu32', 'popupmenu33',  ...
       'popupmenu34', 'popupmenu35', 'popupmenu36', 'popupmenu37','popupmenu38', 'popupmenu39'};

sel = cellstr(get(hObject,'String'));
val = get(hObject,'value');
```


```
if val > 1
var = sel{val};

arr = cellstr(get(handles.popupmenu40,'String'));
arr{4} = var;
str = char(arr);
set(handles.popupmenu40,'String',str)

set(handles.popupmenu28,'visible','off')
set(handles.text35,'visible','on')
set(handles.text35,'String',strcat("Gesture 4: ",var))

sel{val} = num2str([]);
A =  sel(~cellfun('isempty', sel));

for index = 1:length(List)
set(handles.(List{index}),'String',A);
end

h = str2num(get(handles.edit18,'String'));
if h > 4
   set(handles.popupmenu29,'visible','on')
else
   set(handles.togglebutton2,'visible','on')
end
else
end

% --- Executes during object creation, after setting all properties.
function popupmenu28_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end


% --- Executes on selection change in popupmenu29.
```


```
function popupmenu29_Callback(hObject, eventdata, handles)

List = { 'popupmenu30', 'popupmenu31','popupmenu32', 'popupmenu33', 'popupmenu34',  ...
        'popupmenu35', 'popupmenu36', 'popupmenu37','popupmenu38', 'popupmenu39'};

sel = cellstr(get(hObject,'String'));
val = get(hObject,'value');
if val > 1
var = sel{val};

arr = cellstr(get(handles.popupmenu40,'String'));
arr{5} = var;
str = char(arr);
set(handles.popupmenu40,'String',str)

set(handles.popupmenu29,'visible','off')
set(handles.text36,'visible','on')
set(handles.text36,'String',strcat("Gesture 5: ",var))

sel{val} = num2str([]);
A =  sel(~cellfun('isempty', sel));

for index = 1:length(List)
set(handles.(List{index}),'String',A);
end

h = str2num(get(handles.edit18,'String'));
if h > 5
   set(handles.popupmenu30,'visible','on')
else
   set(handles.togglebutton2,'visible','on')
end
else
end

% --- Executes during object creation, after setting all properties.
function popupmenu29_CreateFcn(hObject, eventdata, handles)
```


```
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on selection change in popupmenu30.
function popupmenu30_Callback(hObject, eventdata, handles)

List = { 'popupmenu31','popupmenu32', 'popupmenu33', 'popupmenu34', 'popupmenu35',  ...
        'popupmenu36', 'popupmenu37','popupmenu38', 'popupmenu39'};

sel = cellstr(get(hObject,'String'));
val = get(hObject,'value');
if val > 1
var = sel{val};

arr = cellstr(get(handles.popupmenu40,'String'));
arr{6} = var;
str = char(arr);
set(handles.popupmenu40,'String',str)

set(handles.popupmenu30,'visible','off')
set(handles.text37,'visible','on')
set(handles.text37,'String',strcat("Gesture 6: ",var))

sel{val} = num2str([]);
A =  sel(~cellfun('isempty', sel));

for index = 1:length(List)
set(handles.(List{index}),'String',A);
end

h = str2num(get(handles.edit18,'String'));
if h > 6
    set(handles.popupmenu31,'visible','on')
else
```


```
    set(handles.togglebutton2,'visible','on')
end
else
end

% --- Executes during object creation, after setting all properties.
function popupmenu30_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes on selection change in popupmenu31.
function popupmenu31_Callback(hObject, eventdata, handles)

List = { 'popupmenu32', 'popupmenu33', 'popupmenu34', 'popupmenu35',  ...
        'popupmenu36', 'popupmenu37','popupmenu38', 'popupmenu39'};

sel = cellstr(get(hObject,'String'));
val = get(hObject,'value');
if val > 1
var = sel{val};

arr = cellstr(get(handles.popupmenu40,'String'));
arr{7} = var;
str = char(arr);
set(handles.popupmenu40,'String',str)

set(handles.popupmenu31,'visible','off')
set(handles.text38,'visible','on')
set(handles.text38,'String',strcat("Gesture 7: ",var))

sel{val} = num2str([]);
A =  sel(~cellfun('isempty', sel));

for index = 1:length(List)
```


```
set(handles.(List{index}),'String',A);
end

h = str2num(get(handles.edit18,'String'));
if h > 7
   set(handles.popupmenu32,'visible','on')
else
   set(handles.togglebutton2,'visible','on')
end
else
end

% --- Executes during object creation, after setting all properties.
function popupmenu31_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end



% --- Executes on selection change in popupmenu32.
function popupmenu32_Callback(hObject, eventdata, handles)

List = { 'popupmenu33', 'popupmenu34', 'popupmenu35',  ...
        'popupmenu36', 'popupmenu37','popupmenu38', 'popupmenu39'};

sel = cellstr(get(hObject,'String'));
val = get(hObject,'value');
if val > 1
var = sel{val};

arr = cellstr(get(handles.popupmenu40,'String'));
arr{8} = var;
str = char(arr);
set(handles.popupmenu40,'String',str)

set(handles.popupmenu32,'visible','off')
```


```
set(handles.text39,'visible','on')
set(handles.text39,'String',strcat("Gesture 8: ",var))

sel{val} = num2str([]);
A =  sel(~cellfun('isempty', sel));

for index = 1:length(List)
set(handles.(List{index}),'String',A);
end

h = str2num(get(handles.edit18,'String'));
if h > 8
   set(handles.popupmenu33,'visible','on')
else
   set(handles.togglebutton2,'visible','on')
end
else
end

% --- Executes during object creation, after setting all properties.
function popupmenu32_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end



% --- Executes on selection change in popupmenu33.
function popupmenu33_Callback(hObject, eventdata, handles)

List = { 'popupmenu34', 'popupmenu35', 'popupmenu36', ...
        'popupmenu37','popupmenu38', 'popupmenu39'};

sel = cellstr(get(hObject,'String'));
val = get(hObject,'value');
if val > 1
var = sel{val};
```


```
arr = cellstr(get(handles.popupmenu40,'String'));
arr{9} = var;
str = char(arr);
set(handles.popupmenu40,'String',str)

set(handles.popupmenu33,'visible','off')
set(handles.text40,'visible','on')
set(handles.text40,'String',strcat("Gesture 9: ",var))

sel{val} = num2str([]);
A =  sel(~cellfun('isempty', sel));

for index = 1:length(List)
set(handles.(List{index}),'String',A);
end

h = str2num(get(handles.edit18,'String'));
if h > 9
   set(handles.popupmenu34,'visible','on')
else
   set(handles.togglebutton2,'visible','on')
end
else
end

% --- Executes during object creation, after setting all properties.
function popupmenu33_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end


% --- Executes on selection change in popupmenu34.
function popupmenu34_Callback(hObject, eventdata, handles)
```


```
List = { 'popupmenu35', 'popupmenu36', ...
        'popupmenu37','popupmenu38', 'popupmenu39'};

sel = cellstr(get(hObject,'String'));
val = get(hObject,'value');
if val > 1
var = sel{val};

arr = cellstr(get(handles.popupmenu40,'String'));
arr{10} = var;
str = char(arr);
set(handles.popupmenu40,'String',str)

set(handles.popupmenu34,'visible','off')
set(handles.text41,'visible','on')
set(handles.text41,'String',strcat("Gesture 10: ",var))

sel{val} = num2str([]);
A =  sel(~cellfun('isempty', sel));

for index = 1:length(List)
set(handles.(List{index}),'String',A);
end

h = str2num(get(handles.edit18,'String'));
if h > 10
   set(handles.popupmenu35,'visible','on')
else
   set(handles.togglebutton2,'visible','on')
end
else
end

% --- Executes during object creation, after setting all properties.
function popupmenu34_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
```


```
    set(hObject,'BackgroundColor','white');
end


% --- Executes on selection change in popupmenu35.
function popupmenu35_Callback(hObject, eventdata, handles)

List = { 'popupmenu36', 'popupmenu37','popupmenu38', 'popupmenu39'};

sel = cellstr(get(hObject,'String'));
val = get(hObject,'value');
if val > 1
var = sel{val};

arr = cellstr(get(handles.popupmenu40,'String'));
arr{11} = var;
str = char(arr);
set(handles.popupmenu40,'String',str)

set(handles.popupmenu35,'visible','off')
set(handles.text42,'visible','on')
set(handles.text42,'String',strcat("Gesture 11: ",var))

sel{val} = num2str([]);
A =  sel(~cellfun('isempty', sel));

for index = 1:length(List)
set(handles.(List{index}),'String',A);
end

h = str2num(get(handles.edit18,'String'));
if h > 11
    set(handles.popupmenu36,'visible','on')
else
    set(handles.togglebutton2,'visible','on')
end
else
```


```
end

% --- Executes during object creation, after setting all properties.
function popupmenu35_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on selection change in popupmenu36.
function popupmenu36_Callback(hObject, eventdata, handles)

List = { 'popupmenu37','popupmenu38', 'popupmenu39'};

sel = cellstr(get(hObject,'String'));
val = get(hObject,'value');
if val > 1
var = sel{val};

arr = cellstr(get(handles.popupmenu40,'String'));
arr{12} = var;
str = char(arr);
set(handles.popupmenu40,'String',str)

set(handles.popupmenu36,'visible','off')
set(handles.text43,'visible','on')
set(handles.text43,'String',strcat("Gesture 12: ",var))

sel{val} = num2str([]);
A =  sel(~cellfun('isempty', sel));

for index = 1:length(List)
set(handles.(List{index}),'String',A);
end

h = str2num(get(handles.edit18,'String'));
```


```
if h > 12
   set(handles.popupmenu37,'visible','on')
else
   set(handles.togglebutton2,'visible','on')
end
else
end

% --- Executes during object creation, after setting all properties.
function popupmenu36_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end



% --- Executes on selection change in popupmenu37.
function popupmenu37_Callback(hObject, eventdata, handles)

List = {'popupmenu38', 'popupmenu39'};

sel = cellstr(get(hObject,'String'));
val = get(hObject,'value');
if val > 1
var = sel{val};

arr = cellstr(get(handles.popupmenu40,'String'));
arr{13} = var;
str = char(arr);
set(handles.popupmenu40,'String',str)

set(handles.popupmenu37,'visible','off')
set(handles.text44,'visible','on')
set(handles.text44,'String',strcat("Gesture 13: ",var))

sel{val} = num2str([]);
A =  sel(~cellfun('isempty', sel));
```


```
for index = 1:length(List)
set(handles.(List{index}),'String',A);
end

h = str2num(get(handles.edit18,'String'));
if h > 13
   set(handles.popupmenu38,'visible','on')
else
   set(handles.togglebutton2,'visible','on')
end
else
end

% --- Executes during object creation, after setting all properties.
function popupmenu37_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end



% --- Executes on selection change in popupmenu38.
function popupmenu38_Callback(hObject, eventdata, handles)

List = {'popupmenu39'};

sel = cellstr(get(hObject,'String'));
val = get(hObject,'value');
if val > 1
var = sel{val};

arr = cellstr(get(handles.popupmenu40,'String'));
arr{14} = var;
str = char(arr);
set(handles.popupmenu40,'String',str)
```


```
set(handles.popupmenu38,'visible','off')
set(handles.text45,'visible','on')
set(handles.text45,'String',strcat("Gesture 14: ",var))

sel{val} = num2str([]);
A =  sel(~cellfun('isempty', sel));

for index = 1:length(List)
set(handles.(List{index}),'String',A);
end

h = str2num(get(handles.edit18,'String'));
if h > 14
   set(handles.popupmenu39,'visible','on')
else
   set(handles.togglebutton2,'visible','on')
end
else
end

% --- Executes during object creation, after setting all properties.
function popupmenu38_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end



% --- Executes on selection change in popupmenu39.
function popupmenu39_Callback(hObject, eventdata, handles)

sel = cellstr(get(hObject,'String'));
val = get(hObject,'value');
if val > 1
var = sel{val};

arr = cellstr(get(handles.popupmenu40,'String'));
```


```
arr{15} = var;
str = char(arr);
set(handles.popupmenu40,'String',str)

set(handles.popupmenu39,'visible','off')
set(handles.text46,'visible','on')
set(handles.text46,'String', strcat("Gesture 15: ",var))
set(handles.togglebutton2,'visible','on')

else
end

% --- Executes during object creation, after setting all properties.
function popupmenu39_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit18_Callback(hObject, eventdata, handles)

posTotal = str2num(get(handles.edit18,'String'));

set(handles.togglebutton2,'visible','off')
list = {'popupmenu25', 'popupmenu26', 'popupmenu27', 'popupmenu28', 'popupmenu29', 'popupmenu30',
...
        'popupmenu31', 'popupmenu32', 'popupmenu33', 'popupmenu34', 'popupmenu35', 'popupmenu36',
...
        'popupmenu37', 'popupmenu38', 'popupmenu39'};
for index = 1:length(list)
set(handles.(list{index}),'visible','off');
end

List = {'text32', 'text33', 'text34', 'text35', 'text36', ...
        'text37', 'text38', 'text39', 'text40', 'text41', ...
        'text42', 'text43', 'text44', 'text45','text46'};
```


```
for index = 1:length(List)
set(handles.(List{index}),'visible','off');
end

if posTotal > 15
  set(handles.text47,'visible','on')
  set(handles.edit18,'String',"")
  set(handles.popupmenu25,'visible','off')
else
    set(handles.text47,'visible','off')
    set(handles.popupmenu25,'visible','on')
end

set(handles.popupmenu40,'String'," ")

% --- Executes during object creation, after setting all properties.
function edit18_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function edit14_Callback(hObject, eventdata, handles)
% hObject    handle to edit14 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes during object creation, after setting all properties.
function edit14_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function edit12_Callback(hObject, eventdata, handles)
```


```
% hObject    handle to edit12 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes during object creation, after setting all properties.
function edit12_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit13_Callback(hObject, eventdata, handles)
% hObject    handle to edit13 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes during object creation, after setting all properties.
function edit13_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes on selection change in popupmenu40.
function popupmenu40_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu40 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes during object creation, after setting all properties.
function popupmenu40_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```


```
% --- Executes when figure1 is resized.
function figure1_SizeChangedFcn(hObject, eventdata, handles)
% hObject    handle to figure1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)



% --- Executes on button press in checkbox11.
function checkbox11_Callback(hObject, eventdata, handles)
% hObject    handle to checkbox11 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of checkbox11



% --- Executes on button press in checkbox12.
function checkbox12_Callback(hObject, eventdata, handles)
% hObject    handle to checkbox12 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of checkbox12



% --- Executes on button press in checkbox13.
function checkbox13_Callback(hObject, eventdata, handles)
% hObject    handle to checkbox13 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of checkbox13
```


8.3.2. Data Collector Function

```
function [strng,ii,done] = DataCollector(time,numPos,numPeop,posArr,handles)

% Deletes existing excel files that contain data
for j = 1:length(posArr)

strFile = string(posArr{j});
filename = strcat(strcat('EMGdata_',strFile),'.xlsx');

if exist(filename,'file') == 2
  delete(filename);
end
end
%%
install_myo_mex;

sdk_path = 'C:\myo-sdk-win-0.9.0'; % root path to Myo SDK
build_myo_mex(sdk_path); % builds myo_mex
countMyos = 1;

% Instantiate MyoMex
mm = MyoMex(countMyos)
m1 = mm.myoData(1);

pause(0.1); % wait briefly for the first data frame to come in
% data properties sampled on the IMU time base
m1.timeIMU
m1.quat
m1.rot
m1.gyro
m1.gyro_fixed
m1.accel
m1.accel_fixed
m1.pose
m1.pose_rest
m1.pose_fist
m1.pose_wave_in
```


```
m1.pose_wave_out
m1.pose_fingers_spread
m1.pose_double_tap
m1.arm
m1.arm_right
m1.arm_left
m1.arm_unknown
m1.xDir
m1.xDir_wrist
m1.xDir_elbow
m1.xDir_unknown

% data properties sampled on the EMG time base
m1.timeEMG
m1.emg
%% Wasted for loop. For whatever reason, upon initial data streaming there
%is a flux of data.

    strng = 'Loading...';

    if ~isempty(handles)
            set(handles.text16,'String',strng)
    end

   drawnow()

h = waitbar(0,'Initializing...');

for hh = 1:2

if hh == 1
 m1.startStreaming();

pause(1);
   waitbar(.2,h)
pause(time*.4);
    waitbar(.4,h)
```


```
pause(time*.4);
   waitbar(.6,h)


elseif hh == 2

pause(time*.4);
  waitbar(.8,h)
pause(time*.4);
   waitbar(1,h)

end
m1.stopStreaming();
m1.clearLogs()
end
close(h)
%% Start the data collection process

% numPos  =  % # test subjects
% numPeop =  % poses per subject
% time    =  % time
% posArr  =  % cell array of selected poses

%subj = #people * #pose per person
subj = numPos * numPeop;

for i = 1:length(posArr)

   %strng = 'Prepare to record data. Press spacebar when ready.';
   stR = strcat('Prepare to record data for: ', strcat(" ", string(posArr{i})));
   strng = strcat(stR ,strcat(" ", ', Press spacebar to start recording.'));

   if ~isempty(handles)
     set(handles.text16,'String',strng)
   end
   drawnow()
   pause
```


```
for ii = 1:subj

    if subj > 1
        str1 = strcat('Logging data for: ', strcat(" ", string(posArr{i})));
        str2 = strcat("(",strcat(num2str(ii),")"));
        strng = strcat(str1, strcat(" ", str2));
    else
        strng = strcat('Logging data for: ', strcat(" ", string(posArr{i})));
    end

    if ~isempty(handles)
         set(handles.text16,'String',strng)
        % set(handles.text8,'String',num2str(ii))
    end

drawnow()

m1.startStreaming();
pause(time);
m1.stopStreaming();

if subj == 1 && i ~= length(posArr)
  strng = 'Get ready for next gesture';
  done = 1;
elseif subj ~=1 && ii == 1
  strng = 'Next subject get ready for the same gesture, press spacebar to start recording.';
  done = 1;
elseif subj ~= 1 && ii == subj && i ~= length(posArr)
   strng = 'Get ready for next gesture, press spacebar.';
   done = 1;
elseif i == length(posArr) && ii == subj
   strng = 'Done recording data.';
   done = 0;
end

if ~isempty(handles)
```


```
    set(handles.text16,'String',strng)
    set(handles.text8,'String'," ")
end

drawnow()

if subj > 1
    pause
else
    pause(3)
end

end

    if ~isempty(handles)
        set(handles.text16,'String',strng)
        set(handles.text8,'String','')
    end

   drawnow()

  if i == 1
    sadie = length(m1.emg_log)-200;
    Train = m1.emg_log(2:1:sadie,:);
  else
     Train = m1.emg_log(2:1:sadie,:);
  end

strFile = string(posArr{i});
filename = strcat(strcat('EMGdata_',strFile),'.xlsx');
xlswrite(filename,Train);

m1.clearLogs()
end

mm.delete();
clear mm
```


8.3.3. Artificial Neural Network Function

```
function p = TrainANN(m,n,o,q,pArr,handles)

arL = length(pArr);

for j = 1:arL

strFile = string(pArr{j});
filename = strcat(strcat('EMGdata_',strFile),'.xlsx');

if exist(filename,'file') == 2
   if j > 1
      Features = [Features,xlsread(filename)'];
   else
      Features = xlsread(filename)';
   end
end
end
j=1;

% Combine all the Training Features into one varaible
% Features = [fist,rest,spread,waveIn,waveOut];
% Total amount of collected data
p = length(Features);
%After getting the features we have created Target labels for these features
flen = length(Features);

Labels=zeros(arL,flen);

for j = 1:arL
   if j == 1
      Labels(1,1:((j/arL)*flen-1))=1;
   elseif j == arL
       Labels(j,(((j-1)/arL)*flen):flen)=1;
   else
      Labels(j,(((j-1)/arL)*flen):((j/arL)*flen-1))=1;
   end
```


```
end
% Features is the input data assigned to variable x for further usage
% Labels is the Target data assigned to variable t for further usage
x=Features;
t=Labels;

% Locate the data in the RAM for usage, not necessary. Must be 2 powers
setdemorandstream(391418381)



% Allow for selection of Neural Network training function from App input
trainFcn = q;

% to determine hidden layer size and neurons in each layer
for i = 1:m
   hiddenLayerSize(i) = [n];
end



net = patternnet(hiddenLayerSize, trainFcn);

net.divideFcn = 'dividerand';  % Divide data randomly
net.divideMode = 'sample';  % Divide up every sample
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;

% Speciy how many epochs. Epoch means how many times the
%data is passed to the network for training
net.trainparam.epochs = o;

%update the accuracy after how many epochs
net.trainParam.show = 1;
%specify the learning rate of our model
net.trainParam.lr = 0.1;
%total Check validation of trianing of our model
net.trainParam.max_fail = 50;
```


```
% now everything is done its time to train the network
[net,tr] = train(net,x,t);

% it shows the trianing process
nntraintool

% seprate the features and lables for testing
testX = x(:,tr.testInd);
testT = t(:,tr.testInd);

% pass these features to the network we have trained it will gives us the
% predicted labels in testIndices
% Test the Network
testY = net(testX);
testIndices = vec2ind(testY);

tInd = tr.testInd;
tstOutputs = net(x(:,tInd));
tstPerform = perform(net,t(:,tInd),tstOutputs);

tind = vec2ind(testT);
yind = vec2ind(testY);
percentErrors = sum(tind ~= yind)/numel(tind);


%Plots of our network
% For a list of all plot functions type: help nnplot
%{
figure(1)
plotperform(tr)
figure(2)
plottrainstate(tr)
figure(3)
plotroc(testT,testY)
figure(4)
plotconfusion(testT,testY)
```


```
figure(5)
plotregression(testT,testY)
figure(6)
[M, I] = max(testY, [], 1);
stem(1:length(M),I,':xr')
%}

[M, I] = max(testY, [], 1);
axes(handles.axes1)
%stem(1:length(M),I,':xr')

r = round(length(testT)/arL)-1;
idx = 1;

for i = 1:arL
  for index = idx:(r+idx)
    if I(index) == i
        stem(index,I(index),':g')
    else
        stem(index,I(index),':xr')
    end
    if i ~= arL
    hold on
    end
  end
  idx = (r*i);
end

%view(net)

% Finally, this line will save the trained model which we can use for
%further testing process
save('trained_model.mat','net');
```


8.3.4. Live Streaming Function

```
function [a,b] = LiveStreaming(r,posAr,boxP,boxA,boxR,handles)


if boxR == 1
    load('last_arr.mat');
    posAr = lastPosAr;
else
    lastPosAr = posAr;
    save('last_arr.mat','lastPosAr');
end


clear ar s;
ar = arduino('COM8', 'Leonardo', 'Libraries', 'Servo');
w = {'D5','D6','D9','D10','D11'};
cmd = {0,0,0,0,0};
lastCmd = {1,1,1,1,1};
 for i = 1:length(w)
  if lastCmd{i} ~= cmd{i}
    y = string(w(i));
    z = cell2mat(cmd(i));
    s = servo(ar, y, 'MinPulseDuration', 700*10^-6, 'MaxPulseDuration', 2400*10^-6);
    writePosition(s, z);
    pause(0.315)
    lastCmd{i} = cmd{i};
  end
 end
clear ar s;


%Load the pretrained model for testing in live time for new inputs
load('trained_model.mat'); %#ok<LOAD>
%% Myo Start
install_myo_mex;
sdk_path = 'C:\myo-sdk-win-0.9.0'; % root path to Myo SDK
build_myo_mex(sdk_path); % builds myo_mex
countMyos = 1;


% Instantiate MyoMex
```


```
mm = MyoMex(countMyos)
% Inspect |MyoData|
m1 = mm.myoData(1);

pause(0.1); % wait briefly for the first data frame to come in
% data properties sampled on the IMU time base
m1.timeIMU
m1.quat
m1.rot
m1.gyro
m1.gyro_fixed
m1.accel
m1.accel_fixed
m1.pose
m1.pose_rest
m1.pose_fist
m1.pose_wave_in
m1.pose_wave_out
m1.pose_fingers_spread
m1.pose_double_tap
m1.arm
m1.arm_right
m1.arm_left
m1.arm_unknown
m1.xDir
m1.xDir_wrist
m1.xDir_elbow
m1.xDir_unknown

% data properties sampled on the EMG time base
m1.timeEMG
m1.emg
%%
x=1;
indx = 1;

while x == 1
```

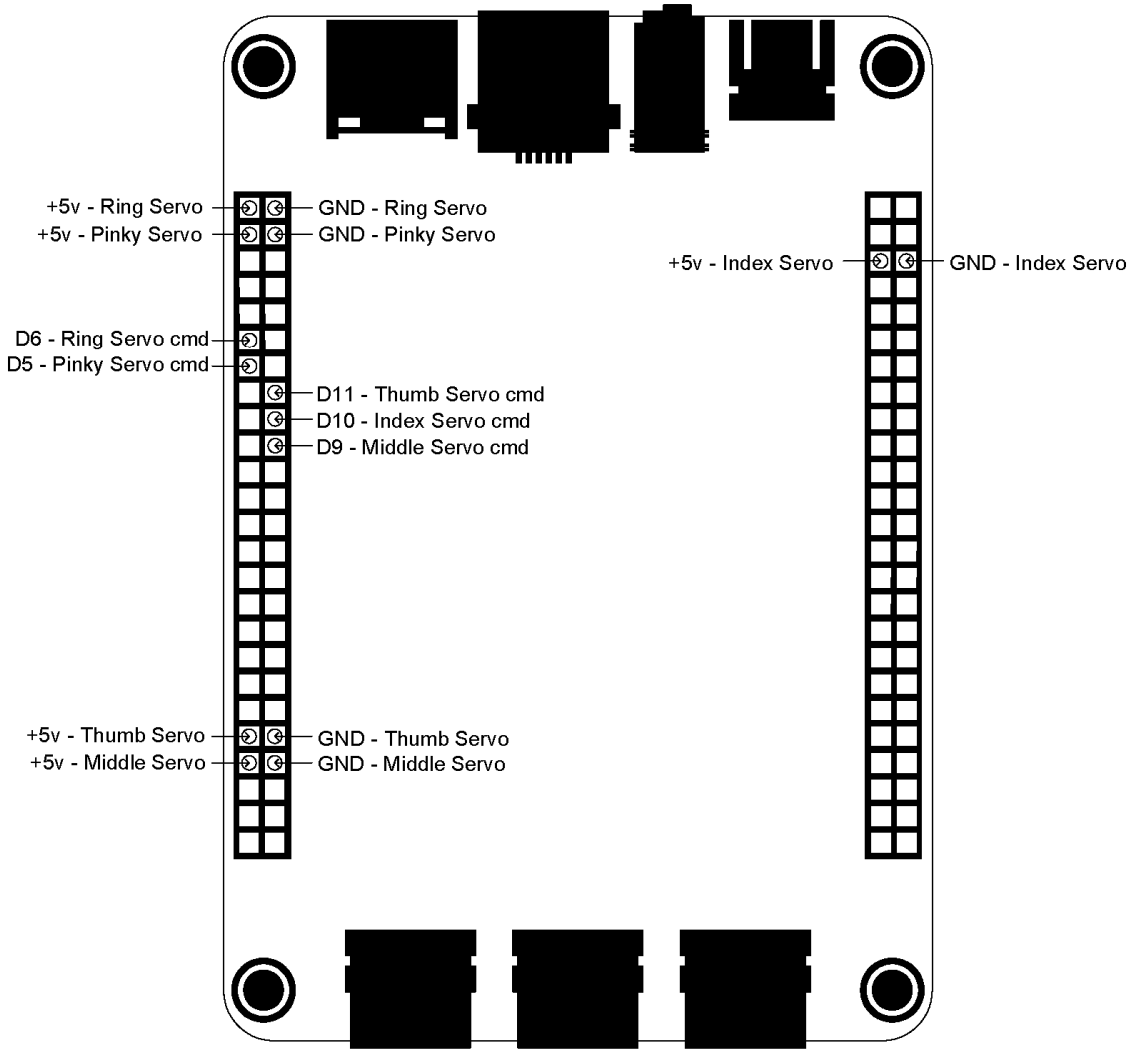

```
m1.startStreaming(); %start data collection
pause(0.35)  % take a 350 mS sample of data
m1.stopStreaming(); %stop collecting data

SensorsData = m1.emg_log'; % transposed emg log

% pass these input values to the net it will give us the prdicted labels as
% we have labeled 1 to n
testY = net(SensorsData);
testIndex = mode(vec2ind(testY));

% after getting the labels for the network we will check the lables and
% assign or show the label accordingly that which class is predicted by our
% network

res = string(posAr{testIndex});
a = res;

 if res == "B"
    cmd = {0,0,0,0,1};
 elseif  res == "C"
    cmd = {0.55,0.55,0.60,0.60,0.55};
 elseif  res == "D"
    cmd = {1,1,1,0,1};
 elseif  res == "F"
    cmd = {0,0,0,1,1};
 elseif  res == "L"
    cmd = {1,1,1,0,0};
 elseif  res == "S"
    cmd = {1,1,1,1,1};
 elseif  res == "V"
    cmd = {1,1,0,0,1};
 elseif  res == "W"
    cmd = {1,0,0,0,1};
 elseif  res == "X"
    cmd = {1,1,1,0.55,1};
```


```
elseif  res == "Y"
    cmd = {0,1,1,1,0};
else
    cmd = {0,0,0,0,0};
end


switch r
case 'All Sensors'
  b = m1.emg_log;
case 'Pod 1'
  b = m1.emg_log(:,1);
case 'Pod 2'
  b = m1.emg_log(:,2);
case 'Pod 3'
  b = m1.emg_log(:,3);
case 'Pod 4'
  b = m1.emg_log(:,4);
case 'Pod 5'
  b = m1.emg_log(:,5);
case 'Pod 6'
  b = m1.emg_log(:,6);
case 'Pod 7'
  b = m1.emg_log(:,7);
case 'Pod 8'
  b = m1.emg_log(:,8);
end

 if ~isempty(handles)
      set(handles.text7,'String',a)
      set(handles.text6,'String','Classified Gesture:')
 end

 if indx == 1
  if boxP == 1
    %plot the emg signals
     cla(handles.axes1)
```


```
    axes(handles.axes1)
    plot(b)
    xlabel('Signal samples');
    ylabel('EMG Signal [mV]');
    set(handles.axes1,'Color','k') %set the background color as black
  end
  indx = 0;
elseif indx == 0
  if boxA == 1
   clear ar s;
   ar = arduino('COM8', 'Leonardo', 'Libraries', 'Servo');
   w = {'D5','D6','D9','D10','D11'};
   for i = 1:length(w)
     if lastCmd{i} ~= cmd{i}
       y = string(w(i));
       z = cell2mat(cmd(i));
       s = servo(ar, y, 'MinPulseDuration', 700*10^-6, 'MaxPulseDuration', 2400*10^-6);
       writePosition(s, z);
       pause(0.315)
       lastCmd{i} = cmd{i};
     end
   end
   clear ar s;
  end
  indx = 1;
end

 st=get(handles.togglebutton1,'Value'); %get if togglebutton is click
 if st==0  %if togglebutton is unclicked
     set(handles.axes1,'Color','w') %set the background color as white
     cla(handles.axes1)  %clear axes
     axes(handles.axes11)
     axis off
     x=0;
    break %break the while loop
 end
```


```
imgStr = strcat('C:\myo-sdk-win-0.9.0\',strcat(res,'.png'));
myImage = imread(imgStr);
set(handles.axes11,'Units','pixels');
resizePos = get(handles.axes11,'Position');
myImage= imresize(myImage, [resizePos(3) resizePos(3)]);
axes(handles.axes11)
axis off
imshow(myImage);
set(handles.axes11,'Units','normalized');

  drawnow(); % make sure that the GUI is refreshed with new content
  pause(0.1) %pause for 0.10 sec


m1.clearLogs() %clear collected data log

end

mm.delete();
clear mm
```


8.4 LattePanda Delta Pinout AutoCad Drawing



9. CONCLUSION

This paper provides the results of research on human prosthetic control via live human body signals (sEMG). With these signals it was more about using muscle memory control than that of signals dictating movement within the brain and throughout the central nervous system. Future work could be to incorporate the use of the inertial measurement unit that has accelerometers and a gyroscope for the sensing components. With this and a more robust robotic hand, a more realistic prosthetic could possibly be created.

There are several facets that can be addressed to improve this current project. The first aspect would be having the LattePanda Alpha rather than the LattePanda Delta. The Alpha has 4 Gb more RAM than the Delta. The Alpha's processor is also an entire 1 GHz faster than that of the Delta. These two improvements alone would improve the system drastically and for an additional cost of $221 it is worth it. This upgrade should be implemented because the system would only become more convoluted as more gestures and data is added into the neural network.

Another area in which this project could be improved is with the 3D printed robotic hand. For the most part the hand is designed well. The few issues are the connecting line used to connect the end of the fingers to the servo motor horns. This line has become worn and has obtained slack causing there to be latency in the finger's response. A better item to use than fishing line would be a type of elastic line that will stretch and stay in position when commanded to close, but when commanded to open it not only opens but assists in opening due to resistance established from stretching it. Currently the servo motor used to control the wrist is not implemented, so connecting that would allow for the motion of an entire new axis. Which in turn would allow for users to add and be able to use more gestures.

REFERENCES

[1] William Morrison, “Electromyography (EMG)”. Internet: https://www.healthline.com/health/electromyography. 2018 [Accessed Apr. 01, 2020].

[2] “What Is a Neural Network?”. Internet: https://www.mathworks.com/discovery/neural-network.html, 2020 [Accessed Apr. 01, 2020].

[3] “patternnet” Internet: https://www.mathworks.com/help/deeplearning/ref/patternnet.html?s_tid=doc_ta. [Accessed Apr. 05, 2020].

[4] “feedforwardnet” Internet: https://www.mathworks.com/help/deeplearning/ref/feedforwardnet.html. [Accessed Apr. 05, 2020].

[5] Mark Tomaszewski, Myo SDK Matlab Mex Wrapper, Matlab File Exchange, https://www.mathworks.com/matlabcentral/fileexchange/55817-myo-sdk-matlab-mex-wrapper, [Accessed Apr. 03, 2020].

[6] LattePanda Delta, Internet: https://www.dfrobot.com/product-1910.html [Accessed Apr. 02, 2020].

[7] InMoov, Internet: http://inmoov.fr/. [Accessed March. 29, 2020].

[8] “crossentropy” Internet: https://www.mathworks.com/help/deeplearning/ref/crossentropy.html, 2020. [Accessed Apr. 10, 2020].