SOFT ROBOTIC EXO-MUSCULAR ARM BRACE

A Major Qualifying Project Report:
submitted to the Faculty of:
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of the requirements for:
Degree of Bachelor of Science by:

Michael Brauckmann	Seth Lipkind
Elliott Calamari	Christopher Molica
Benjamin Leone	Amanda Piscopiello

William Terry

Date: April 25, 2013
Approved:

Professor Gregory S. Fischer, Major Advisor

Professor Edward Clancy, Co-Advisor

Professor Marko Popovic, Co-Advisor

Professor Eduardo Torres-Jara, Co-Advisor

Abstract

The goal of this project is to assist patients with impaired movement and to regain control of their arm. A robotic brace was developed to assist with movement, using signals generated from the user's muscles to drive the arm. This brace was biologically inspired, allowing the user to complete the range of motions of a healthy individual. Different actuators and sensors were evaluated in order to design the best model for home and patient use. Boards were developed to achieve desired values from signals that were read. Classifications were also created to accurately assess the movements the user wanted to perform

Acknowledgements

We would like to thank the robotics engineering, electrical engineering, computer science, and biomedical engineering departments for their support of this project. We are also very grateful towards Advanced Circuitry and Sean Cushman, the Blue Engineering School and David Clark, and Lin Engineering for their generous donations towards this project.

We thank Professor Gregory Fischer for guiding this project and providing us with space in the AIM lab, and we thank all of his helpful grad students. We thank Professor Edward Clancy for his time, his valuable wisdom, and all of his help. We thank Professor Marko Popovic for his insight to the project. Finally, we thank Professor Eduardo Torres-Jara for his help with this project.

We extend our gratitude towards the WPI faculty for their help. Especially Professors Sonya Chernova, and Neil Heffernan for their assistance with the AI models, Professor Steven Bitar for his assistance in diagnosing our electronics, and Professor Taskin Padir for his assistance in developing the controls system.

We also thank Tracy Coetzee for her invaluable help ordering parts and organizing finances, Joe St. Germain for lending us parts from the robotics lab, and Tom and Pat from the ECE shop for helping us find the parts we needed. Additionally, we would like to thank Bob Brown for providing us with server access and Lisa Wall for helping us with our data collection experiments.

We would like to thank Thane Hunt, Sean Weaver, Sean Casey, Adam Jardin, Adam Val-Roth, and Thanacha Choopojcharoen for their aide in completing this project.

We are forever indebted to Michael Delph and Phil Gauthier for their continuous support and friendship throughout the project.

Finally, we extend our tremendous gratitude to the giants upon whose shoulders we stand.

Table of Contents

Authorship

Abstract	Seth
Acknowledgements	Liam
Authorship	Elliott
Executive Summary	Amanda
1) Introduction	Liam
2) Literature Review	
2.1) Current Art	Elliott/Chris
2.2) Anatomy	Liam
2.3) Actuation	
2.3.1) PAMs	Mike
2.3.2) EAPs	Liam
2.3.3) Cabling	Amanda
2.4) Human Interaction	
2.4.1) Joint Analysis	Chris
2.4.2) Electromyography	Liam/Amanda
2.5) Data Interpretation	
2.5.1) Classifiers	Amanda/Ben
2.5.2) Control Systems	Chris
3) Brace Construction	
3.1) Design Alternatives	Elliott
3.2) Methodology	Chris
3.3) Results and Discussion	Chris
4) Actuation	
4.1) Design Alternatives	Mike
4.2) Pneumatic Actuators	
4.2.1) Methodology	Mike
4.2.2) Results	Mike
4.3) Electroactive Polymers	
4.3.1) Methodology	Amanda
4.3.2) Results and Discussion	Liam/Seth
4.4) Cable Driven Actuators	
4.4.1) Methodology	Chris
4.4.2) Results	Chris
4.5) Comparison	Mike
5) Sensors	
5.1) Design Alternatives	
5.1.1) Objectives & Const	Liam/Amanda

5.1.2) Possible Solutions	Liam/Amanda
5.2) Methodology	
5.2.1) EMG Design	Liam/Amanda
5.2.2) EMG Testing	Liam/Amanda
5.2.3) IMU Methodology	Elliott
5.3) Results	Liam/Amanda
6) Software & Systems	
6.1) System Arch & Software	
6.1.1) Design Alternatives	Ben
6.1.2) Methodology	Ben
6.1.3) Results and Discussion	Ben
6.2) Classification	
6.2.1) Design Alternatives	Liam/Amanda
6.2.2) Methodology	Liam/Amanda/Ben
6.2.3) Results and Discuss	Liam/Amanda/Ben
6.3) User Interface	
6.3.1) Design Alternatives	Seth
6.3.2) Methodology	Seth
7) Final Design	
7.1) System Level Design	Elliott
7.2) Brace Construction	Elliott
7.3) Actuation	Elliott
7.4) Sensors	Elliott
7.5) Software & Systems	Elliott
8) Discussion & Conclusion	Seth
9) Appendices	All
10) Bibliography	Amanda

Executive Summary

In the United States there are between 236,000 and 327,000 people with spinal cord injuries related to trauma. These injuries are one of the main causes in decrease movement in the upper and lower arm [1]. Due to the decrease in functionality of the arm, many of these people can no longer preform daily tasks such as eating or brushing their teeth. Home-care helpers and therapist assist the patients to perform these tasks, creating a significant increase of money.

The creation of a device that would help the patient preform these tasks independently would be beneficial for not only the user but the family as well. A device like this would allow the user to regain independence and greatly reduce the cost of living for the family. With this idea in mind an assistive robotic brace was developed. The final design allowed the arm to flex, extend and rotate at the elbow. These motions were provided by one of two actuating forces.

Three designs were considered for the brace: a rigid exoskeleton, a soft system, and a hybrid between the two. It was determined that a hard brace would require a solid hinge, which is misplaced could injure the user. A soft system does not require a solid hinge but relies on the patient's bone structure. This system cannot support actuation because there are strong attachment points, allowing the actuation to slip. Finally, a hybrid of both a hard and soft brace would allow for smaller profile devices and many options regarding safety.

The goals for brace design included limited slippage, comfort, easy mounting, and no limit for range of motion. Neoprene and nylon-neoprene were both tested for limit slippage. The nylon-neoprene was more capable of conforming to the arm and shifted much less than the other prototype. A kinematic model and dynamic physiological model were developed to ensure the designated range of motion and torque and safety factors respectively.

The final brace design consisted of actuation points on the upper and lower arm and cable guides. These connections, printed in ABS plastic, gave the cables needed to actuate the braces hard points to attach to and were attached to the arm through bucked straps, which could be tightened to fit any user. By only using actuation points, the brace was kept to a soft shell, minimizing weight and bulk in the final design.

The design of a powered elbow brace requires actuators that can mimic the function of the skeletal muscles that naturally powered this joint. Electric motors and Bowden cable systems, McKibben Pneumatic actuators, and dielectric polymer spring roles were all considered to perform these motions. The Bowden cables system is made up of electric motors, semi-rigid cables sheath, and a flexible cable. With the right motor these actuators could produce more than adequate force and precise positioning. A McKibben actuator is a woven shell surrounding a bladder where both attached to rigged end caps. The muscle's shape, contraction, and tension could be determined and computed by using information about the bladder and the braided sleeve. The spring rolls uses the elasticity of the polymer to compress a spring. The polymer is stretched and attached to either end of a spring such that the spring is encompassed by the polymer. When a voltage is applied across the electrodes of the polymer the polymer experiences a force compressing it perpendicular to the circumference of the spring which allows the spring to expand laterally.

Three Different actuators were considered for use on this project: Electro-Active Polymers, Pneumatic Artificial Muscles and a Bowden Cable system. Each type of actuator was constructed and tested. The Electro Active Polymer (EAP) is an actuator that contract when charged by an electric current. Several versions were created and tested. They are very light weight and do not require additional systems to control them however, we were unable to achieve desired contractions and most designs failed in testing. Pneumatic Artificial Muscles were also tested using the McKibbens design. These actuators contract when pressurized with compressed air. They also have compliance property which varies with compression, and can be combined in series and parallel to create an actuator with controllable compliance. These actuators were constructed via 3D printing as well as from off the shelf part and were tested both as

standalone actuators and in series and parallel networks. The Mckibbens are also very light weight and naturally limit the range of motion; however they would require the addition of a compressed air system. The series and parallel configuration allows controlled compliance however it also complicates the controls scheme. Finally a Bowden Cable system was tested. This system allows cable on the arm to be actuated be motors in a backpack removing most of the weight from the arm. Additionally it can be combined with series elastic force sensors and encodes on the motors for easy force and position sensing. Due to its simplicity, easy combination with force and position sensing, and light weight on the arm this system was implemented on the final device.

It was determined that a non-commercial, house-made EMG acquisition system would best satisfy the requirements for this project. The system is broken up into two parts and electrode to capture the data and a board to filter and amplify the signal. Prior revisions of the electrode circuit was based on used a gain of 20, however gains of 50 and 100 were being examined for this function. The design of the custom filtering board was split into four sub designs. These were the first stage of amplification, the second stage of amplification and signal filtering the digitization of the signal, and the mechanical components, which contacted the user's skin.

The first stage of testing validated that the filter design of the Sallen-Key filters and amplifiers functioned properly. The second stage of the circuit verified the function of the physical circuitry and the circuit's ability to acquire an actual EMG signal. The last stage of testing ensured that the hardware had been designed and assembled properly.

The EMG acquisition system was successfully implemented as a modular printed circuit board. Any number of the boards can be connected to a backplane and collect EMG data through an embedded system. The system was composed of stainless steel electrodes which rested on the skin, an instrumentation amplifier which amplified the signal by a gain of 100 and performed high pass filtering on the circuit. The signal was then processed by a 26 Hz to 1500 Hz bandpass filter as well as two variable gain stages totaling a maximum gain of 30. The signal was then digitized by a 14 bit analog to

digital converter. The system successfully acquired EMG data from a test subject's biceps muscle.

A singular computer model and a distributed computer model were looked at to control the system. The overall computer structure was one main computer directly running the controls, reading sensors, and driving motors or several smaller computers for the individual tasks. The single central computer offered simplicity, less space, and less power draw. A distributed computer model provided a reasonable contrast to the single computer model. The distributed computer model incorporated several processors and micro controllers to handle the system's needs. On the down side power requirements could go up due to replication of hardware over multiple boards.

Distributing the computer systems required for this system was very beneficial to the project. It allowed for the simultaneous development of the EMG acquisition software, high level software, and motor controller software. Additionally, the computational power was never strained with the distributed system and was well worth the slight increase in power draw.

Three EMG electrodes were used to gathering signals from the biceps and triceps. This information was used to train and test the classifier, which returned results with over 85% accuracy. Both the EMG signals collected and the classifier results are represented on the GUI. A soft, exo-musculature brace capable of lifting the user's arm was constructed. The brace is driven by two antagonistic pairs of cables, and force and position feedback is provided from integrated sensors. The motion of the arm is determined by the EMG classifier running on the Raspberry Pi, which passes this information to the motor control circuits.

1) Introduction

In the United States alone, there is currently over one million spinal cord injuries related to trauma. These injuries are one of the main causes in decrease movement in the upper and lower arm [1]. Due to the decrease in functionality of the arm, many of these people can no longer preform daily tasks such as eating or brushing their teeth. Home-care helpers and therapist assist the patients to perform these tasks, creating a significant increase of money.

The creation of a device that would help the patient preform these tasks independently would be beneficial for not only the user but the family as well. A device like this would allow the user to regain independence and greatly reduce the cost of living for the family. With this idea in mind, the team developed an assistive robotic brace.

To assist the patient regain their mobility, the team evaluated several designs of the brace. The final design allowed the arm to flex, extend and rotate at the elbow. These motions were provided by one of two actuating forces.

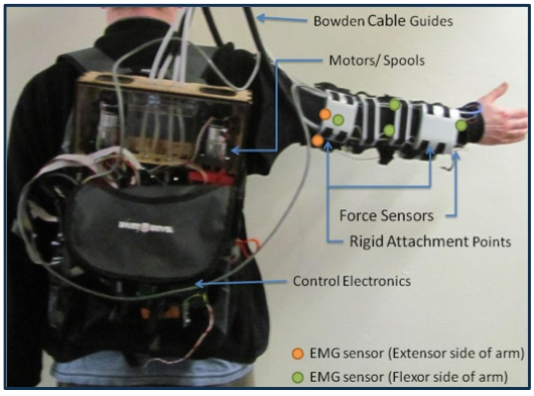


Figure 1: Powered orthotic system

The first actuator on the system is a set of cables. Some of these cables were pulled to contract the lower arm to the upper and released to allow straighten the arm again. The other cables were pulled to rotate the lower arm in a clockwise or counter-clockwise motion. Motors, located in the backpack, drove all of the cables.

McKibbon actuators were also used to move the arm. When filled with air the McKibbon actuator would compress lengthwise. These actuators are placed in series and parallel and each compress a specific amount.

This makes it so the actuators can only compress up to a specific amount, which prohibits the quantity of available force.

The actuator's movements were determined by the signals produced from muscles. These signals were captures in six positions on the arm the processed to determine the movement.

All of these components are interlocked and fit on the arm with connections to the soft robotic brace. This brace has two main connects points: one on the upper arm and the other connection on the lower arm. Some of the electrodes are pictured by the green sections in Figure 3.

This prototype gives the patient the ability to move their arm to complete daily tasks. This allows the patient to be more independent and it puts a smaller burden on the family.

2) Literature Review

A significant amount of research has been done on powered orthotics. There are a few commercially available devices, and many more systems residing in labs. Many smaller subsystems, like biological signal acquisition, biological signal interpretation, and experimental actuator research, have been evaluated independently by researchers.

2.1) Current Art

Many powered orthotics exist today, ranging in size and scope. Two examples of powered orthotics, the Myomo mPower 1000 (Myomo) and the "OrthoJacket" system (Eck, Ute, Matthies, Richard, Pylatiuk, Christian, et al., 2011), have similar application and design to the soft orthotic brace. The mPower 1000 was based on technology developed at MIT, and the OrthoJacket was developed by students at Duke University [2] . There is also a wide range of unpowered braces which provide structural support, and are good models to base powered exoskeleton orthotics off of.

The Myomo mPower 1000, shown in Figure 2, uses bicep and tricep mounted surface electromyography (EMG) sensors to measure signals to actuate the elbow joint. The range of motion is from 3 to 130 degrees, and the total weight is just under 850 g, or about 2 lbs [2]. It offers a variety of modes of operation which allows it to be adapted from use as a daily assistive device to rehabilitative therapy. The device can be used with data from both the triceps and the biceps or only one muscle. The mPower has a built in interface which allows the therapist or user to change settings to suit their current needs. A suite of mobile software is under development which will collect biometrics from the devices as it runs and track the progress of the user.

Figure 2: Myomo mPower 1000 orthotic

The 'OrthoJacket' is larger and actuates about the shoulder, wrist, and elbow joints. The Orthojacket also uses EMG to measure muscle signals, and, based on the readings, drives both motors and fluid actuators. The shoulder joint is primarily driven by stepper motors, utilizing a chain of two motors which push on the back of the user's upper arm rotating the whole arm up or down. The elbow joint is driven by a set of fluid actuators around the back of the joint which expand and contract to control flexion and extension.

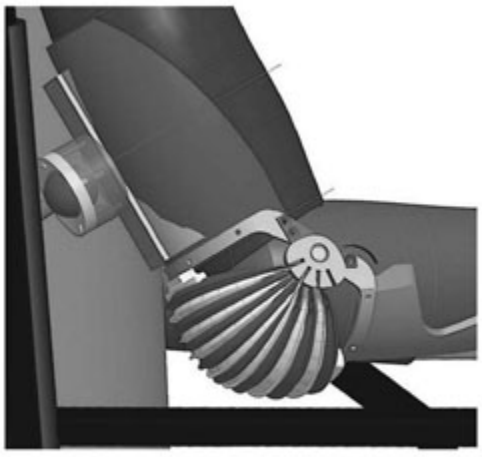


Figure 3: CAD drawing of orthojacket elbow

A significant observation made by Duke University team was that extension of the elbow needed considerably less torque to actuate because this motion was aided by

gravity. The design review discussed the use of fluid actuators, pneumatics, and cable drives to actuate movement. Fluidic actuators were selected to power the elbow because of the high power-to-weight density [3].

In addition to these two examples of powered orthotics, unpowered orthotics were also examined. Unpowered elbow orthotics are produced in a variety of styles for a variety of purposes. Additionally, the knee joint is very similar to the elbow joint, so braces for the knee can also be examined for insight. While fully cloth braces are available, the overwhelming majority of rigid braces use a hinge connected to rigid bars to support the elbow or knee. Both partial shell orthotics, which have a 'half-pipe' molded section of plastic to support the limb, and metal frame orthotics, which have a metal frame encompassing half of the arm with thick padded straps to secure the device, are common. A popular joint mechanism is the double gear joint. This design is essentially two pin joints whose motions are linked by two gears which rotate about the pin joint. Some orthotics also use simple pin joints, although these are frequently lockable and are intended to immobilize the arm. Orthotics typically come in two to three sizes, and have a number of adjustable points and straps.

2.2) Anatomy

In order to develop the mechanical device for the elbow, appropriate research and understanding of the anatomical structure of the human arm is necessary. The elbow is a 1-DOF joint which connects the humerus and ulna. About this passive synovial joint, the arm is capable of flexion, raising the lower arm about the elbow, and extension, lowering the lower arm about the elbow [4]. This path of motion resembles a mechanical revolute joint. An antagonistic set of muscles is necessary to develop motion for the elbow since a single muscle can only retract and is not capable of extension. The muscles used in elbow flexion are the biceps brachii, the brachialis, and the triceps brachii. These muscles are located on the top and bottom of the humerus if the arm is held vertical palm up. The muscles for wrist rotation are located just below the wrist and in front of the elbow on top of the forearm bones . Two muscles are necessary for the elbow to flex and

extend. The biceps are responsible for flexion and the triceps are responsible for extension.

Anatomical Mechanical properties

The elbow has several limitations that should not be exceeded in order to prevent damage to the joint. Additionally the plant model of the elbow depends on a number of anatomical factors. The elbow is capable of a range of motion of 0º- 150º. The shoulder is capable of 0-90º during flexion, 0-50º during extension, 0-90º during abduction, 90º-0º during adduction, and 0-90º during lateral and medial rotation [5].

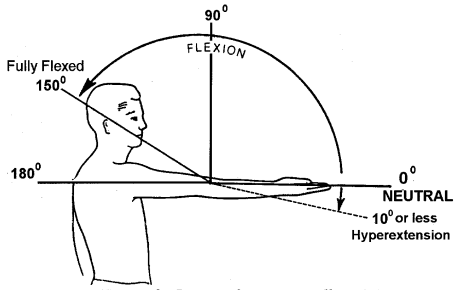


Figure 4: Elbow motion. Figure taken from muralitharan.org

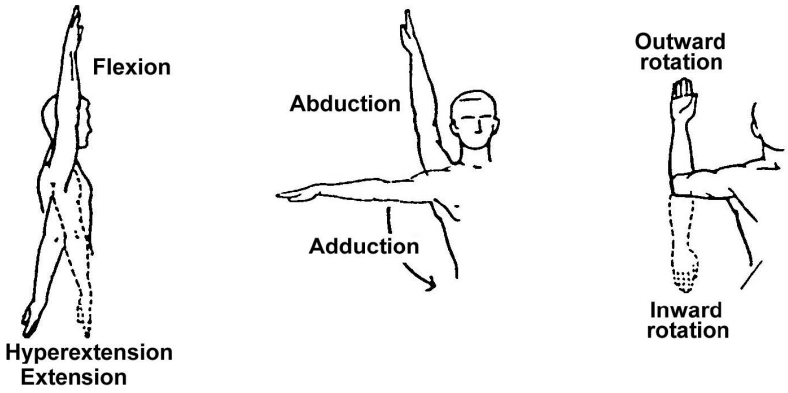


Figure 5: Shoulder motion. Figure taken from nursing411.org

The minimum required force can be determined by analyzing the weight of the lower arm. An assistive device must be at least capable of lifting the forearm and hand. Using body segment data, forearm and upper arm weights can be calculated based on percentages of the patient's body weight. Based on this data the forearm and upper arm are equivalent to just over 2.5% of the user's body weight.

Using this data in conjunction with a Hill-type muscle model the necessary tension for each motion can be determined .

$$(v + b)(F + a) = b(F_0 + a)$$

$$b = \frac{a \cdot v_0}{F_0}$$

In this equation *v* represents the velocity of the contraction, F_0 is the maximum isometric tension generated in the muscle, and *a* is the coefficient of shortening heat. [6]. This equation demonstrates that the relationship between force and velocity is hyperbolic. This indicates that when there is a higher load applied to the muscle the velocity will be lower than it would be if the load was applied to a lower section of the muscle.

Using Hill's elastic model, the total force in the muscle can be broken into three elements: the contractile element, the parallel element, and the series element. These elements represent the passive and active components of muscle. The parallel and series elements are mechanically equivalent to a non-linear spring, while the contractile element is equivalent to a damper.

2.3) Actuation

A variety of actuators have been used in previous powered orthotics. Electric motors have been used to drive the joint directly, stepper motors were used by the OrthoJacket to push and pull on the user's arm to drive the shoulder, and Bowden cables and pneumatic actuators have been used to in a similar manner to the way human skeletal muscles work. Additionally, new actuators called electro-active polymers are under development which posses a number of desirable qualities in biomimetic applications.

2.3.1) Pneumatic Artificial Muscles

Pneumatic Artificial Muscles (PAM's) are a type of gas operated actuator that provides liner and unidirectional motion. These actuators are constructed from flexible closed membranes restrained at both ends. When inflated, a pressure differential is

created between the inside of the membrane and the outside air. This results in a radial expansion coupled with an axial contraction. The latter axial contraction produces a pulling force between the two end fittings. Another way of viewing this contraction is as an inverse bellows creating a contractile force when inflated instead of an expansion as is the case with bellows [7]. Unlike pneumatic cylinders, the force developed by this contraction is dependent on both pressure and load. For any combination of these there is a single equilibrium length of the actuator. This means under constant pressure the muscle will extend as load increases, and under constant load the muscle will contract as pressure is increased. This allows for position control of the actuator, contrary to pneumatic pistons which generate a constant force regardless of the state of contraction. The force generated by a PAM is unidirectional occurring only under contraction and relying on load for expansion. As such antagonistic pairs are required for controlled bidirectional actuation [7].

PAM actuators have several advantages over other types of actuation. One advantage is the inherent compliance of the system. All pneumatic actuators possess a degree of compliance due to the compressibility of gas, however, the PAM's dropping force to contraction curve and spring like behavior of the membrane increase its compliance over other pneumatic actuators [8]. Furthermore, the compliance of a PAM is dependent on the pressure used for actuation and the properties of the membrane used. This means that the degree of compliance can be controlled both during the design phase by choice of membrane and actuator length as well as during operation by altering the load and pressure balance. Another advantage of PAM's is their light weight while maintaining strength. Because the primary construction element is a thin membrane, PAM's are usually very light, however a PAM operating at the same volume and pressure as a pneumatic piston would produce an equivalent force. This high power to weight ratio make PAM's ideal for use in small mobile systems where weight is a significant constraint. There are two major designs commonly used to construct Pneumatic Artificial Muscles: the McKibben Artificial Muscle and the Pleated Artificial Muscle [8].

McKibben artificial Muscles are the most frequently used PAM to date. They are comprised of an internal air bladder surrounded by a braided sleeve. The braid is composed of strands wound helically about the long axis in opposite directions. The angle of the weave, which is designated as ±θ, is known as pitch or weave angle. When pressurized the internal bladder presses against the sleeve and is balanced by tension in the braided fibers. The fiber tension is translated to the end caps to create the desired contracting force. The bladder and the braided sleeve determine the behavior of the muscle in terms of shape, contraction, and tension. Tension and volume can be computed from these characteristics and the operating pressure [9].

$$Force = \frac{\pi D_{\max}^2 P}{4} * 3\cos^2(\theta - 1)$$

$$or$$

$$\varepsilon = 1 - \frac{l}{l_0}$$

$$Force = \frac{\pi D_0^2 P}{4} * \left(\frac{3}{\tan^2(\theta_0)}(1-\varepsilon)^2 - \frac{1}{\sin^2(\theta_0)}\right)$$

$$Volume = \frac{I_s^3}{4\pi N^2}\cos(\theta)\sin^2(\theta)$$

- l is the measured length of the muscle
- l_0 is the nominal length of the muscle
- D_0 is the nominal diameter
- D_{max} is the maximum Diameter of the muscle (weave angle of 90 degrees)
- θ is the weave angle and
- P is the Pressure.

A typical McKibben Muscle operates on pressures from 100-500 kPa and contracts by 40% of its length. Although weights and power densities vary widely by material usage, an average of 6g for 10cm of muscle at 1cm diameter can be expected

producing a power to weight ratio ranging from 3-5 kW/kg. The McKibben is the most common type of PAM due to its low cost, light weight and easy assembly. The major concern with the McKibben muscle is the amount of dry friction between the bladder and sleeve [9].

Pleated Muscles were developed as an improvement to the McKibben design to increase contraction percentage and reduce energy loss due to dry friction and deformation of the bladder. The muscle membrane has a series of pleats running axial along it which unfolds when pressurized. This process does not rely on friction with a sleeve and little energy is needed to expand the membrane. Thus pleated muscles are considerably more efficient than the original McKibben design. The pleated muscle bulges spherically as pressure is increased requiring more space for radial expansion. The force produced by a pleated muscle is a function of the materials' stress properties, contraction rate, and operating pressure. A theoretical maximum contraction is 54% of length, however material thickness and properties makes a more practical value 45%. Most commercial actuators of this type only achieve 41.5% contraction at reasonable operating pressures. In published tests these muscles were used to control a 1DOF arm with .1 degree accuracy while maintaining compliance control. McKibben actuators placed under the same test resulted in accuracy less than 1 degree and larger response time primarily due to inconsistent friction and hysteresis in the McKibben actuators [8] [9].

2.3.2) Electro Active Polymers

Electro Active Polymers (EAP's) are polymers which react to electrical phenomena by altering shape. EAP's can be used as actuators because of their behavior in the presence of electricity. There are many different types of EAP's, each having a unique composition and activation mechanism. EAP based actuators have several characteristics which make them advantageous in biomimetic applications. EAP actuators are excellent parallels of skeletal muscle, they actuate linearly and they are compliant meaning that they reach an equilibrium between force and length for a given stiffness. EAP's possess a distinct advantage over conventional actuators because EAP's are the

electro-mechanical transducers, they do not require additional hardware to transform the electrical energy or transmit the physical motion [10]. The three EAP materials which were considered are dielectric polymers, conductive polymers, and ionic polymer metal composites.

Dielectric EAP's (dEAP's) take advantage of the polarizability and elasticity of certain polymers. dEAP's consist of a thin sheet of dielectric polymer with a compliant electrode deposited on either side of the sheet. When the electrodes are subjected to a voltage difference the electric field that results compresses the polymer sheet vertically which in turn results in horizontal expansion [11]. Figure 6 shows an diagram of this phenomenon. As with capacitors, a thinner dielectric and higher dielectric material can increase capacitance and consequently the strength of the electric field. DEAP's can have strains anywhere from 20-300% depending on the quality of manufacturing, the materials used, and the mechanical coupling [12]. They also typically have stresses on the order of MPa, and respond at a frequency of 10 Hz [12] [13] [14]. A significant advantage that dEAP's possess is that they consume very little energy to maintain a specific stiffness because the stiffness is proportional to the charge, which is intrinsically stored, in the actuator. DEAP's do have working potentials greater than a kilovolt [10]. Often these potentials are close to, or higher than the breakdown voltage of air, which can make producing human friendly dEAP actuators challenging.

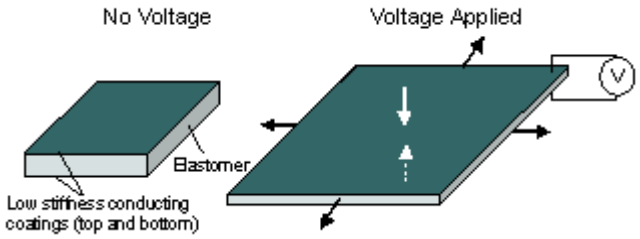


Figure 6: Dielectric EAP mechanism

Conductive polymer EAP's, or intrinsically conducting polymers (ICP's) are polymers capable of conducting electric charge which bend when they are subjected to an electric current. Normally polymers are thought of as good insulators, however, there are many organic polymers which are capable of conductive electricity. ICP's are typically

constructed as long, thin rectangular pieces with a compliant electrode applied to either face of the piece. As current begins to pass through an ICP, ions flow within the polymer as they are rearranged according to the presence of the electric field. The result is a swelling in one side of the material and a contraction in the other side causing the polymer to bend. One Significant advantage of ICP's is that they are operated primarily by current, and require electric potentials of only a few volts [15]. Also, they have very fast response times, and can be used to recycle energy. Conductive polymers typically exhibit very high stresses, up to 30 MPa in some cases [16]. However, they have lower strains, around 12%, and with low mechanical coupling they can be difficult to use effectively [17] [10].

Ionic polymer metal composites (IPMC's) have a similar construction and mechanism to ICP's. IPMC's differ in their physical construction. They are a sponge like polymer which has been impregnated with an electrolyte and is encased between two thin metal layers [18]. Applying a voltage to an IPMC causes ions in the electrolyte to redistribute within the polymer. As the ions redistribute themselves within the material, they are followed by water which causes one side of the ICMP to swell and bend the actuator. IPMC's typically have lower strains than ICP's, around 2%, and lower stresses, around 0.23 MPa [19] [20]. However, they typically have better mechanical coupling. Unfortunately, deformation in IPMC's is difficult to maintain with direct current because of the mobile ions and water molecules . This makes IPMC's difficult to operate in a controlled manner.

2.3.3) Cable Driven Actuation

Bowden cables are flexible two layer cables used to move forces or energy from one end to another [21]. These cables usually include a small sturdy inner layer and a larger hollow outside layer. One end of this system is connected to a control member, commonly DC motors [22]. These motors are able to manually control movement with cables. These devices are being used for many reasons, including vehicle hatchback's, car seat adjustments, and bicycles. Recently, cable driven actuators have been used in

robotic systems and new ways to use these systems are constantly being discovered [22] [21] [23] .

The outer level of these cables is used as housing for the inside steel wire. The housing is commonly made out of tightly wound helix or joined steel wire. The inside wires are usually used to preform a pulling movement, however it can sometimes be used as a pushing force. Wires that are solely used for pulling will be more flexible than ones used for pushing. Both the inside wires are usually incased in some type of plastic to prevent corrosion in the system [21].

One of the current robotic systems that use Bowden cables is the Wearable Soft Robotic Device for Post-Stroke Shoulder. This device is designed to be used at home by stroke victims. For this system the cables are used to help control from linear and angular displacements [24].

2.4) Human Interaction

Developing a device which can integrate with a human arm in a natural, fluid way requires an understanding of the biological signals and linkages involved. Previous experiments have established a strong correlation between elbow and shoulder movement for specific motions. Additionally, weak residual EMG signals have been used to interpret the intended motion of spinal injury patients.

2.4.1) Interconnected Joint Analysis

Studying three-dimensional measurements from motion capture studies provides an assessment of multiple joint movements. These studies set up three cameras about the transverse, sagittal, and frontal planes and track the motion of the arm performing daily activities using a series of spherical reflective markers. A study conducted at the University of Göteborg developed a three-dimensional kinematic motion analysis of drinking from a glass [25] . The user was asked to first reach for a glass, drink from the glass, and then return the glass to the table.

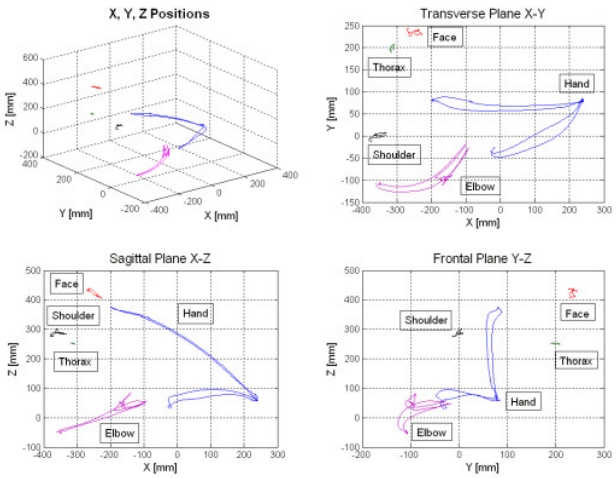


Figure 7: Three dimensional analysis of drinking from a cup

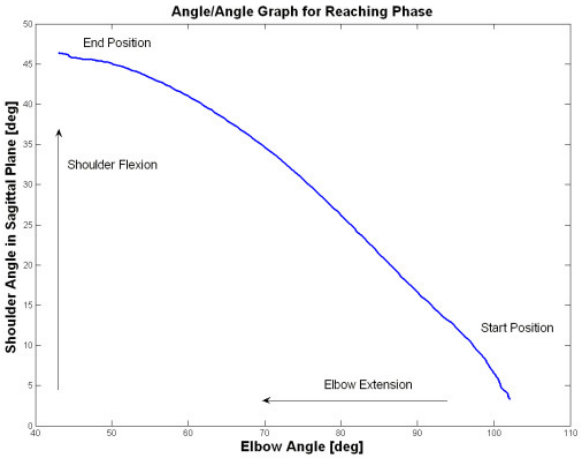


Figure 8: Interjoint correlation

Findings from this study strongly suggested a correlation in joint angles between the elbow and shoulder in both the frontal and sagittal planes. The interconnection between the shoulder and elbow joint for this exercise suggested a high correlation between movements. Based on a graph comparing the joint angles of the elbow and shoulder about the sagittal plane, as shown in Figure 8, there was an almost linear correlation between these joint angles with correction coefficient ranging from 0.92 to 0.98 [25].

2.4.2) Electromyography

Electromyography or EMG is a field that studies electrical activity in muscles. The amount and size of this activity is due to the level of activity and travels by an electrical current through the arm .

Paralysis and Electromyogrphy

Traumatic injury to the C5 and C6 vertebrae typically leads to paralysis in the lower arm. A deterministic approach has yet to be found which can predict the amount of control retained by a victim baring observation. The control retained varies greatly from person to person and injury to injury. A study done in 2001 on the hand function of C6 or C7 (the two vertebrae below the C5 vertebrae) tetraplegics reports that the subjects were able to complete 90% of the tasks in the study, but object manipulation was poor as well as the lateral grasp [26]. The national spinal cord injury statistics center repots that incomplete tetraplegia is the most commonly reported type of injury of all spinal cord injuries in 2012 at 40.8% (NSCISC 2012). Incomplete tetraplegia meaning that the patient has recovered at least partial functionality of the nerves originating at or below the site of the injury. Often muscle groups become very weak due to the loss of nerve connections which severely limits the functionality of the affected muscles. However, because the muscles are weakened and not completely paralyzed, the muscles can still produce weak electric fields. Multiple efforts to utilize these weak signals have been made to map functional electrical stimulation to neural input [27]. Neck injuries are further complicated by the potential for spasticity, or involuntary contractions, in paralyzed muscles [28].

Different muscle velocities and forces help determine arm motion. Multiple muscle groups can be used to determine the motions, which becomes useful when some signals are lost after being paralyzed. Using this knowledge, the movement flexion can be determined by electrode placements on the biceps and triceps or multiple on the shoulder. Electrode placement can affect the quality of an EMG signal. This is demonstrated in Figure 9. EMG electrodes are designed to sense the faint electrical potential below the pickups. Accordingly, placing the sensors along different spots of the

muscle can affect how much of the desired muscle signal is picked up, as well as how much cross-talk from other muscles is sensed. In some ways reproducible electrode placement can be more important than location on the arm because all the controller is really concerned with is mapping a repeatable signal with a single motion [27]. In other respects, the placement of electrodes can affect the strength and timing of the intercepted signal. Electrodes located closer to the muscle-nerve junction see larger signals and less delay in signal reception [29].

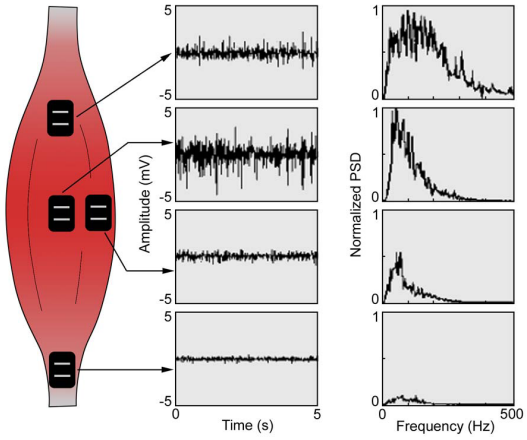


Figure 9: Effect of sensor placement on signal acquisition Invalid source specified.

Two examples of EMG electrode placement are shown in Figure 10. Additionally, by widening the gap between electrodes, the electrode can sense deeper into the muscle, however this also brings the electrode further from the muscle nerve junction and closer to the end of the muscle where signal delay is greater and there can be cross-talk from other muscles [30,31]

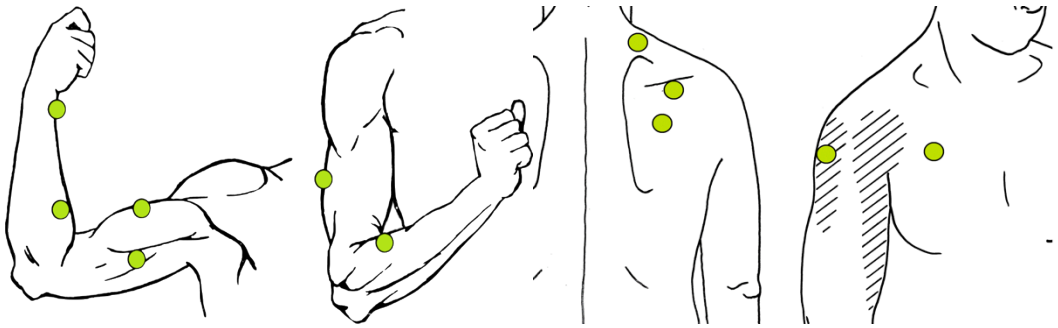


Figure 10: The left picture shows EMG electrode placement for six different muscle. This set up is used to determine velocities to determine movements such as flexion, extension, supination, and pronation. On the right the electrode are all placed around the shoulder. This set-up can determine movements from forces.

Electromyographic Analysis

EMG signals are data heavy, but they are not information heavy meaning that they contain a lot of data about what the biological processes that give rise to the signals, but not as much information about what the muscle is doing. This is because of the way the body sends nerve signals to the muscle. The signals reach muscle fibers at asynchronous times and in a random pattern. This random recruitment causes the signal seen by a surface electrode, which is the result of constructive interference between many muscle fibers, to tend towards a Gaussian curve.

This property in conjunction with the small signal amplitudes, on the order of tens of microvolts, means that acquiring a clean signal is key to analyzing the data. There are four sources of noise present in EMG systems. The first is inherent noise due to imperfections in the electronics which cannot be prevented [32]. Ambient noise is due to the electromagnetic radiation present in the environment, the most common of which is 60 Hz noise from AC power [33]. The third source of noise is motion artifacts, which occur when the electrode shifts on the surface of the skin. Finally, inherit instability of the signal is noise that is important to remove. The effect of amplitudes fired within the 0Hz to 20Hz range will cause noise in the EMG signal. Noise will greatly impact the recorded signal which is why filtering is very important to EMG analysis. [32].

Simpler control schemes utilize muscles like buttons, by flexing a specific muscle the user can “turn on” a specific motion or “turn off”. By performing a co-contraction, or

clenching several muscles at once, a user can achieve a wide array of functions with just a few muscles [31].

One example of EMG analysis is based on thresholds. The single-threshold method compares the amplitude of an EMG signal with a fixed threshold. A single-threshold has a higher rate of false positives. The double-threshold method has a higher accuracy detection than the single-threshold method. Using the double-threshold method allows the user to determine a link between false positives and accurate results, which will result in more successful trials [32].

Wavelet analysis is another type of EMG evaluation. This is a mathematical tool for analysis of non-stationary and fast transient signals. Wavelet analysis can be implemented in two ways. The first way is by the mean of a discrete time filter bank, and the second is by specific Fourier Transforms called wavelet filters [34].

The time-frequency approach is another method of EMG analysis. Time-frequency approach has many sub-approaches. The most popular are Wigner-Ville distribution and Choi-Williams distribution. The Wigner-Ville distribution is a probability distribution function, whose graph uses the probability density where the signal is compared to distorted version of itself for all possible shifts and lags [35]. The function is a semicircle of radius centered at the origin. The Choi-Williams distribution is used to filter out the cross-term results from the components in both time and frequency function that crosses out differences in both time and frequency domain [35].

The next type of analysis is autoregressive model. The autoregressive model is when the electrode picks up a signal with minimal cross talk from nearby muscles. This model uses linear prediction formulas. A linear prediction formula is used to estimate a future value of a discrete-time signal based upon previous samples [32].

Finally, the last type of analysis is high-order statistics. High-order statistics are used to analyze and interpret the characteristics and nature of a random process. These detections and characteristics exist in nonlinearities of EMG signals [36].

2.5) Data Interpretation

Biological systems are very complex and often operate very differently from mechanical system. As such, it is necessary to develop methods with which a non-biological system can interpret complex data, like the signals from a half dozen muscles generating a specific motion, and output the correct intention. Artificial intelligence models called classifiers are commonly used for this task. It also is then necessary to translate this intention into a movement achievable by a mechanical system. This is done through the development of control systems.

2.5.1) Classifiers

With advancements in computing power, speed, and size, classifiers have become popular tools to interpret EMG signals. Classifiers are excellent tools for analyzing EMG signals with regard to the user's intended motion because classifiers can take data which may not be perfectly consistent and map the input data to output commands.

Artificial neural networks are common classifiers. Devices called functional electrical stimulators (FES) have been successfully developed which utilize neural networks to help restore motion in partially paralyzed limbs. A neural network takes EMG as input and then electrically stimulates muscles which no longer receive nervous stimulation using implanted electrodes [37,38].

Fuzzy logic algorithms have also been used in FES systems. Fuzzy logic algorithms have the advantage of being able to interpret imprecise, or 'fuzzy', information [39]. Additionally, attempts have been made to characterize EMG signals by looking at the activity of several muscles, and then determine the desired activity [40].

2.5.2) Control Systems

A control system consists of a series of subsystems and processes, which are assembled to determine a desired output, from a given input. When developing a actuating system that interacts with humans, a precise controller with zero impedance

would be a desired controller for the system [41]. Depending on the mode of control, the brace must be able to either actively compensate for friction and motor inertia, or provide an assistive torque. To control a series elastic actuator, two controllers were proposed.

One proposed system came from the University of Berkeley, which uses rotational series elastic actuators [42]. The key advantage of this system is that series elastic actuators have low impedance and are capable of generating the necessary torque output to interact with humans. Additionally, series elastic actuators are back drivable and has a shock tolerance, improved with the spring. The motor's required force fidelity will be reduced, allowing for cheaper motors, because the control system relies specifically on position rather than torque output of the motor.

Another proposed control system uses a force feedback loop based upon a series elastic element. Veneman discussed the importance springs played for series elastic actuators. A spring that is too stiff will reduce the maximum controller gain, which cases a less-smooth output [43]. Low spring stiffness will decrease the system performance, as the series elastic actuator will saturate quicker.

3) Brace Construction

A custom brace was constructed in order to provide for the desired functionalities for the system. The system needed to satisfy all of the requirements set forth such as no heavier being lighter than five pounds, having limited motion, and having limited force. Since no brace was found to have all of these requirements, one was constructed.

3.1) Design Alternatives

For the physical design of the arm, three basic designs were considered; a rigid exoskeleton, a soft system, and a hybrid between the two. Within each of these categories, alternatives opened up in design.

Hard systems were what we first looked at. Rigid exoskeletons are the 'classic' image of an exoskeleton. They provide excellent support for the patients arm as well as solid mounting points for actuators. We found that the disadvantages of hard systems far outweighed the advantages though. First, the system may not be as safe as possible. With a hard system, a solid hinge is necessary, and if misplaced, could tear the elbow out of alignment when actuated upon. Also, hard systems tend to be bulky, while we were looking at something low profile, so that it could be worn under the clothing.

The next design we looked at was an entirely soft exoskeleton. Soft systems are very low profile, essentially an extra sleeve under the patients clothing. Also, soft exoskeletons remove the hinge problem that arose with hard exoskeletons. Without a rigid structure, there is not a solid hinge to actuate upon; rather the system relies upon the patient's bone structure to serve as the hinge. Drawbacks of entirely soft systems come from actuation. Slippage in actuation points is a real problem, as the elasticity of skin undermines rigidity. Without strong attachment points, there is slippage, resulting in failed actuation and pain for the patient.

The third design was a hybrid between soft and rigid. Using rigid braces in a soft system allow for smaller profile devices and many options regarding safety, attachment points, and mobility.

3.2) Methodology

The primary goals for the brace were to:

- prevent physical harm
- limit slippage
- be comfortable to wear
- be easy to put on
- not limit the users range of motion

The material for the sleeve of the brace needed to limit slippage from the system. A series of tests were conducted with three brace sleeve designs to compare the amount of slippage of the sleeve. The first system, Figure 11 used a modified knee brace made from 1mm thick neoprene. The brace was cut into half, connecting the posterior of the arm. Four sets of Velcro were used to attach the brace to the arm. The cable guides were manufactured on a 3D printer from ABS plastic. The early guides were designed to enable testing of the Bowden cable system and the McKibben actuators.

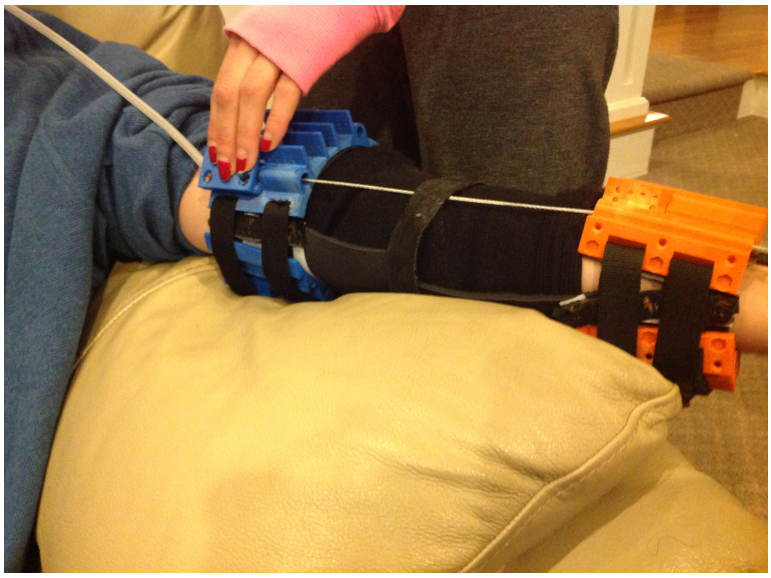


Figure 11: Slippage test for the first prototype of the brace.

The second system, Figure 11, uses a 3 mm wetsuit arm sleeve, made from a nylon-neoprene hybrid. The thicker neoprene is capable of conforming well to the arm and shifted much less than the previous prototype. Two additional cable guides were added above the lateral epicondyle and extensor tendons to help guide the triceps during extension. Like the previous prototype, the mounting points are manufactured from ABS plastic.

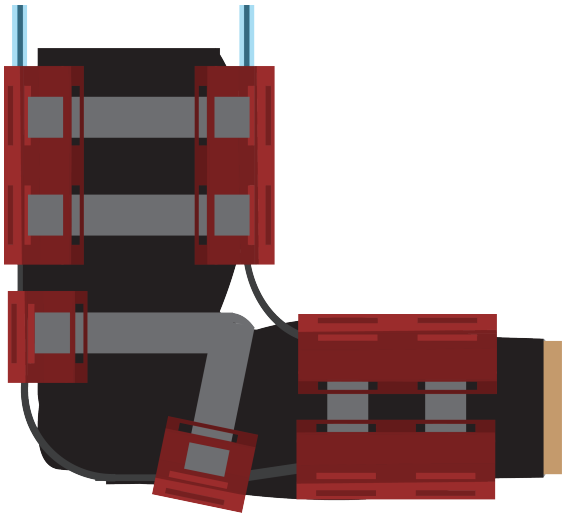


Figure 12: Final Prototype for Brace Design

To test slippage, cabling was attached to the brace at the appropriate mounting points and pulled to maximum flexion and extension. To preserve the arm when developing the brace, a model of the arm was fabricated from silicon and Dragon Skin. The model is shown in Figure 13.

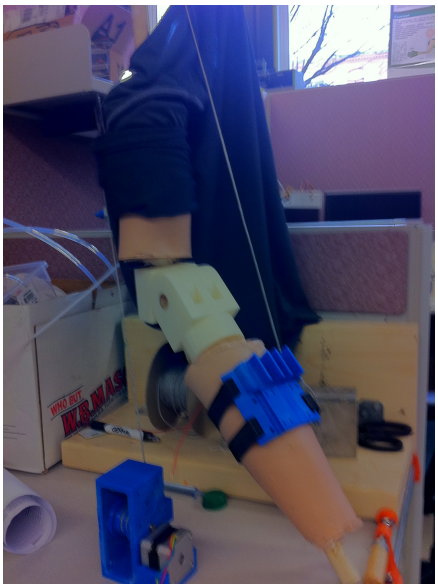


Figure 13: First revision of silicon model of a human arm

To ensure the brace is capable of actuating about the arm for the designated range of motion, a kinematic model was developed. In order to determine the necessary torques and safety factors necessary to actuate the elbow, a dynamic physiological model was developed using the *Investigation of Inertial Properties of the Human Body* [44]. Development of the dynamic model requires determining the body weights of the lower arm, which comprises the forearm and hand. Using [44], the masses of the limbs were determined to be:

Table 1: Approximate arm weights

Arm Segment	Mean	Standard Deviation
Upper Arm	4.061 lbs	0.481 lb
Forearm	2.399 lbs	0.408 lb
Hand	0.882 lb	0.201 lb

3.3) Results and Discussion

The first brace design was simply a proof of concept to verify that at the given attachment points, the brace would be able to assist a user in both flexion and extension. The brace guides were placed at the bicep, triceps, the forearm flexor muscles, and wrist flexor muscles. The aim of this model was to verify the attachment points selected were capable of actuating the arm, and helped determine the proper configuration of the bottom layer to reduce slippage, when the brace was being tensed. Hypothetically, when each of the cables was pulled the brace would flex and extend.

The major problems with the initial prototype focused on determining a proper attachment system to the arm to reduce slippage, reducing torque necessary for extension, measuring force in the system, and modifying attachment points to better suit the user and sensors used with this system.

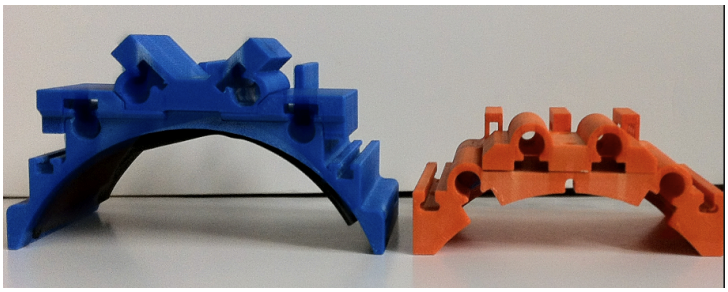


Figure 14: First (blue) and second(orange) revisions of lower arm segments

The updated system, as seen in Figure 14, solved the number issues resulting from the previous iteration. First, a 1mm neoprene sports brace was added to the posterior side of the arm, connecting the attachment points from the forearm flexor muscle to the triceps. In addition, an additional 1mm neoprene adhesive was attached to the top attachment points. With this configuration slippage was minimized, but not completely reduced. The next iteration of the overall size of the brace was reduced by 65% to have improve how well the brace conforms brace's conform around the users arm.

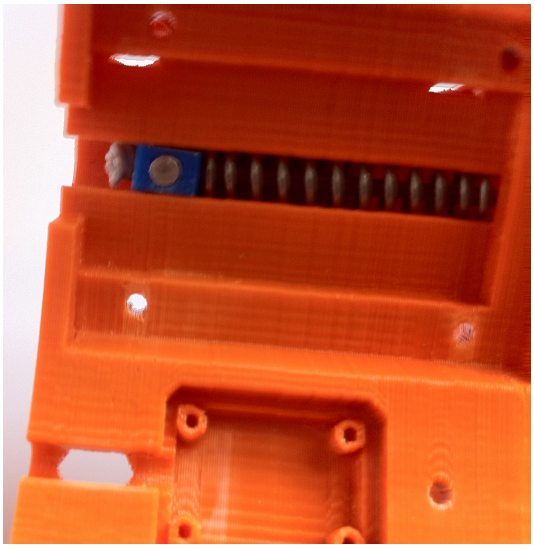


Figure 15: View of SEA mechanism and magnet mount

Third, the modification to the brace was the addition of a series-elastic actuator (SEA). As depicted in Figure 15, the shows the location of the SEA, which uses a 16.81 lb/in. compression spring attached to a slider with an embedded neodinium magnet. Using a magnetic linear potentiometer, the tension be determined using Hooke's law.

Figure 16: Final revision of brace

Based on the modifications from second iteration of the brace, a final design was developed, which had minimal to no slippage, was much simpler to attach, and further reduced the torque necessary for actuating extension. Using the arm of 3mm wetsuit, the brace and including rivited strap guides, seen above in Figure 16, slipage was reduced. When the brace is being pulled the guides prevent the brace from shifting by increasing the surface area, which the brace is applying a force to the entire sleave. Next, the addition of the two attachment points for extension, distributed the cable around the subjects elbow, instead of rubbing against the elbow. This improvement greatly reduced friction in the system and decreased the necessary force by 40%.

4) Actuation

Actuation is used as the driving force of the arm. These actuators are used to move the arm to desired locations and positions. There are many different types of actuators able to control the forces needed for the system; three are examined in depth below.

4.1) Design Alternatives

The design of a powered elbow brace requires actuators that can mimic the function of the skeletal muscles that naturally power this joint. Skeletal muscles are linear, contractile actuators with an extremely high power to weight ratio. Excessive weight on the arm could cause discomfort and injury. Similarly, applying forces or displacements which are significantly greater than the body's natural limits could also cause damage to the arm. Thus it was necessary to find an actuator that mimics the low weight profile of skeletal muscles, generates similar forces, and can be constrained to the force and position limits of the body. Three such actuators, shown in Figure 17, were considered for this project:

- Bowden Cables
- Pneumatic Artificial Muscles
- Electro Active Polymers.

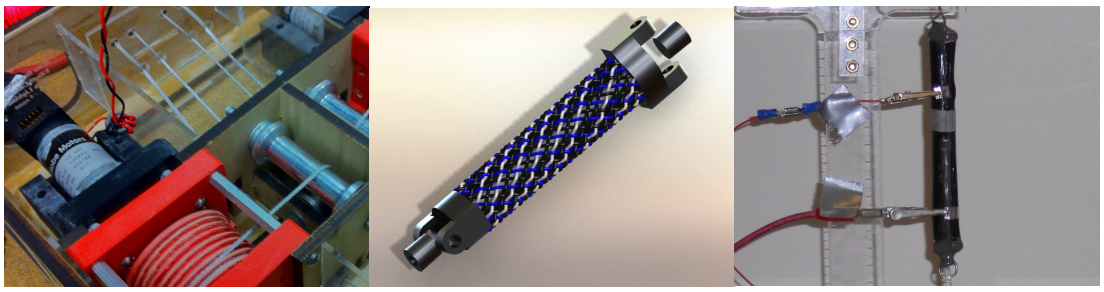


Figure 17: (From left to right) Bowden cables, pneumatic artificial muscle, electroactive polymers

The first type of actuator that could be used to move the device was an electric motor and Bowden cable system. This system is composed of an electric motor, a semi-rigid cable sheath, and a flexible cable. The motor and both ends of the cable sheath were fixed to a ground surface, and one end of the cable is attached to the motor shaft while the other end is attached to the surface which was to be actuated. By reeling in the cable with the motor the two surfaces which were attached to the cable sheath and the cable could be pulled together. Electric motors were widely used in industry; consequently they come in a wide variety of capabilities. For the purpose of actuating a human arm the right motors could produce a more than adequate force as well as precise positioning. Also, motors have no limitation to the amount of displacement they could achieve since they rotate continuously, allowing for gearing to achieve higher force. Motors are typically powered by a DC source and can be driven by a variety of voltage levels including 12v. The size of a motor depends on the power output and the transmission. For reference a 1kw motor weighs 3.8kg. Motors require special electronics to control the power provided to them, these controllers are commercially available. A common, simple controller was the H-bridge controller which controls the speed of a motor by modulating the duty cycle of a digital signal which turns the current on and off rapidly.

Another design alterative for actuators was a type of Pneumatic Artificial Muscle (PAM) known as a McKibben Muscle. These muscles consist of a woven shell surrounding a bladder, both of which are attached to rigged end caps. Inflating the bladder with compressed air causes it to bulge outward generating tension in the threads of the weave. This tension pulls the end caps together causing the actuator to contract along its axis. The behavior of the muscle in terms of shape, contraction, and tension was determined by the bladder and the braided sleeve. Tension and volume could be computed from these characteristics and the operating pressure. These actuators were inherently compliant because contraction depends both on the applied force and pressure. Typical commercial McKibben actuators attain 40% contractions under maximum pressure ratings and have a power to weight ratio of 3-5 kw/kg. These actuators were lightweight and closely parallel the function of skeletal muscles making them ideal for use as artificial muscles. However, there required a source of compressed air to operate

which meant adding significant amounts of air storage and a compressor to the design of the device. Additionally the stress-strain relationship was linear and the controls needed for series and parallel configurations were more complicated and less common than those used for electric motors.

The final actuator that was considered was a dielectric polymer spring roll. A dielectric polymer is a material which has a high dielectric constant, but also enough malleability that it compresses when placed between two electrically charged plates. The spring roll construction is a method of increasing the strain of the dielectric polymer. The device uses the elasticity of the polymer to compress a spring. The polymer is stretched and attached to either end of a spring such that the spring is encompassed by the polymer. When a voltage is applied across the electrodes of the polymer the polymer experiences a force compressing it perpendicular to the circumference of the spring which allows the spring to expand laterally. Using this method, dielectric polymers could achieve strains of up to 300% and stresses of up to several tens of MPa. The size can vary widely, but these actuators were typically constructed on a centimeter to meter scale, and were typically fairly lightweight since the majority of the device is the hollow inside of the spring. Spring roll actuators could require voltages of over 1kv, but these potentials could be reduced by using thinner materials with higher dielectrics. Controllers exist for spring rolls, and they are essentially adjustable voltage supplies. However, these devices were neither widespread nor inexpensive.

Each of these actuators had the potential to mimic the behavior of human muscles which was a vital part of this project. A test procedure was designed to evaluate the essential properties of all three actuators in order to determine the best approach for the development of this project. Table 2 breaks down the properties of each actuator and the tradeoffs that must be considered before one could be selected.

Table 2: Actuator selection matrix

Actuator	Advantages	Disadvantages	Estimated ability to meet actuator specifications
Electric Motor/ Cabling	• Extremely advanced and well researched comparatively • Frequently available as 'off the shelf' systems • Can drive a joint antagonistically with a single motor	• Requires complex mechanical coupling to achieve compliance • Motors do not have a nice form factor, and may weigh a significant amount • Not self-damped	• Good; Very common actuator with few bugs which have not already been explored and solved
Mckibben Muscle	• Simple and inexpensive to manufacture • Lightweight actuators with excellent form factors • Self limited contractions	• May be difficult to control • Requires a pressure source which can be noisy and heavy	• Good; Very close parallel to human muscle in terms of performance and form factor
Dielectric Polymers	• Very efficient because it consumes very little energy comparatively • Silent operation • Does not require intermediate transducer	• Requires high voltages (>1kV) • Prohibitively expensive • Self manufactured spring rolls may not be able to achieve very high strains	• Moderate; high voltages poses a potential safety threat

4.2) Pneumatic Actuators

Pneumatic Artificial Muscles are gas operated and provides liner, and unidirectional motion. As the air is supplied to the component, the bladder expands, forcing the whole system to contact. When placed in series and parallel with each other they can provide multiple contraction lengths and stiffness equilibriums.

4.2.1) Methodology

This section will cover the process taken to develop the McKibben model of a Pneumatic Artificial Muscle (PAM) as well as the experiments done to determine its feasibility for use on the robotic arm brace.

Construction

Two different approaches were taken to construct the PAM's. 3D printing allows for exact dimensions which allow both more control over the parameters of the muscles and for the muscles to be made very small for testing in series and parallel configurations. However it was not clear that the current 3D printing technology would be sufficient to produce working muscles due to the properties of the available printing materials. Another possibility is to construct them from off the shelf components while this is simpler it restricts us to the properties and dimensions of available components. Both methods are covered in the following sections.

3D printing

One way to construct the McKibben design is to print the whole assembly on a rapid prototyping. Some 3D printers such as the Objet are capable of printing a range of materials which or even creating "digital materials" which obtain different material properties by mixing two base materials together while printing. This process allows the flexible bladder to be printed in the same step as the more rigid weave and end caps. Before the McKibben muscle can be printed, a CAD model must be produced and appropriate materials must be assigned to each part. Additional steps must be taken to

make sure the CAD model is within the tolerances of the prototyping machine, and that room is left to remove the support material from inside the bladder.

Constructing a CAD Model of the cylindrical bladder and end caps is not difficult with Solidworks. However, the helical structure of the weave is difficult to define in Solidworks and care must be taken to ensure the threads do not intersect as this would limit the contraction of the artificial muscles. In order to create the model of the weave the shape of the thread was defined mathematically and imported into solid works as a 3D curve. The cross sectional shape is then swept along the path defined by this curve to create each thread. A helix can be defined mathematically by equation 1.

Equation 1: $$r(t) = A * \cos(t)\mathbf{i} + A * \sin(t)\mathbf{j} + B * t\mathbf{k}$$

Here A represents the radius of the helix and B represents the helical pitch which relates to change in height of the helix in one rotation. This however will not make a weave in order for the threads to be interwoven they must change from a helix with one diameter (outside diameter in Figure 18) to a helix with a smaller diameter (inside diameter in Figure 18).

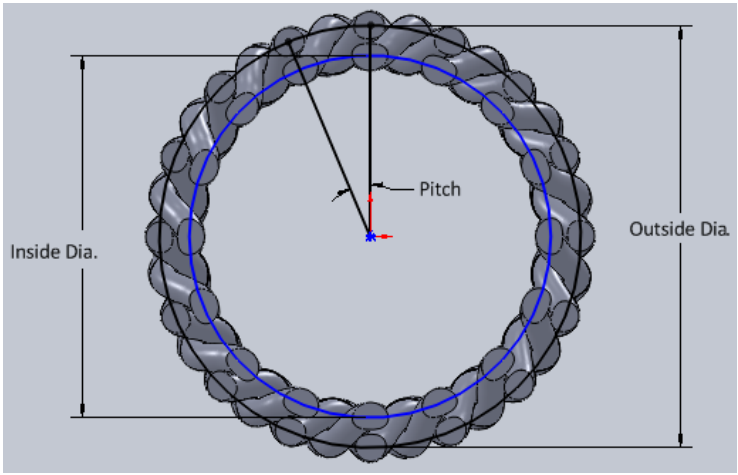


Figure 18: Cross section of McKibben weave CAD model

Excel was used to generate a list of Cartesian coordinates for consecutive point on these two helixes at each half pitch increment. The same process is then repeated starting on the opposite radius and proceeding in the opposite direction around the circle. This

creates two counter rotating threads which are at opposite radii at any given z level. Solidworks 3D curve through points tool can be used to generate the geometry from this data. Using the circular pattern tool to replicate that geometry space around the circle by the pitch angle produces the full helical weave as seen in Figure 19.

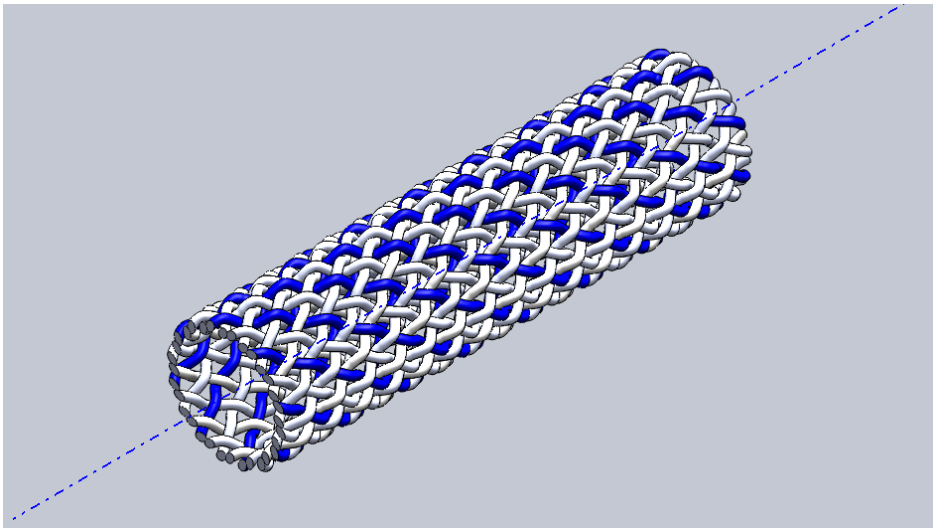


Figure 19: Orthographic view of McKibben weave CAD model

The Objet machine used to print the McKibbens muscles is capable of printing in two materials or combining them into a digital material which results in material proprieties somewhere between those of the two bases. In this case we are mostly interested in the stiffness of the material. The Bladder was assigned the "Tango Plus White" the softest material available while the end caps where designed to be rigid and Used Tango Black the stiffest material available. The weave needed to be flexible enough not to break when the bladder was in flatted but rigid enough to transfer the desired forces to the end caps. It was not clear what the appropriate balances where so several muscles were printed with the weave material with a mix ranging from 50% of the rigid material to 90% of the rigid material.

The Objet prints support material wherever there are gaps in a part so that the part will remain rigid and structurally sound throughout the printing process. This process means that upon completion of the printing process the inside of the bladder will be filled with support material which must be removed before the muscle will function. To allow

for removal then end caps were made with large holes in both ends and tapered plugs were designed to fit into the holes with one plug attaching to the air hose and the other a solid plug. Figure 20 shows the final assembly.

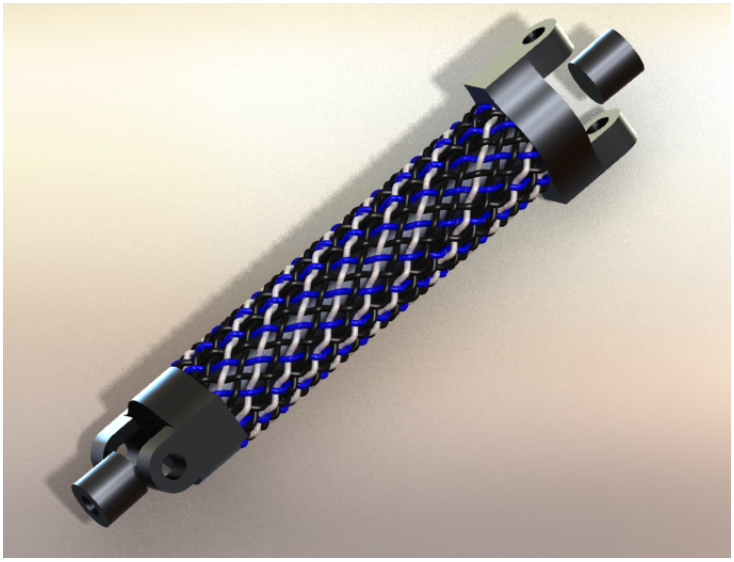


Figure 20: Orthographic view of full McKibben actuator CAD model

Off the shelf parts

Another approach to constructing the PAMs is to buy off the shelf components that meet the need of the bladder and weave and fasten them to end caps via an air tight seal. This approach is simpler then 3D printing however you are limited by the size and material properties of the available components. A nylon weave meant for bundling cables was used as the weave and silicon tubing was used as the bladder because pneumatic tubing is to stiff to expand as needed. End caps were machined with plugs that the end of the silicon tubing could be stretched over. The weave was then placed over the silicon tubing and a house clamp was tightened over the portion of the weave and silicon tubing that was backed by the plug on the end cap. One end was drilled and taped to receive a pneumatic fitting and to allow airflow into the bladder. With the exception of the machined end caps all other parts were ordered from mcmastercarr Bill of materials for McKibben actuator is the bill of materials (Quantities of fittings for 5 muscles) and Figure 21 show the final assembly.

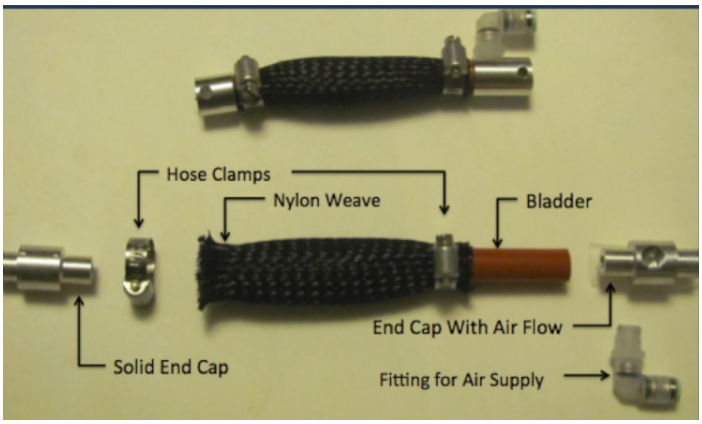


Figure 21: Exploded view of McKibben actuator

Table 3: Bill of materials for McKibben actuator

McMaster Part	Description	Quantity
9142K34	Heavy Duty Polyester Expandable Mesh Sleeving, 1-1/2" ID, 3/4" to 2" Bundle Diameter, 3' L, Black	1
5236K15	High-Temperature Silicone Rubber Tubing, Soft, 3/8" ID, 1/2" OD, 1/16" Wall, Red	3 feet
5526K21	Metric Clean-Room Polypropylene Push-to-Connect, 90 Degree Elbow for 4MM Tube OD X 1/8" BSPT Male Pipe	5
5388K14	Worm-Drive Hose Clamp with Zinc Plated Steel Screw, 7/32" to 5/8" Clamp Diameter Range, 5/16" Band Width, Packs of 10	1

Experiments

Several Experiments were conducted to determine the properties of the muscles we constructed, if pneumatic muscles were feasible for use on this project, and how they would best be used. The first experiment we conducted involved measuring the contraction length while pressure was varied. This experiment as conducted one McKibbens muscles of three different lengths 2, 4 and 6 inches and under different loads No load, 500g and 1000g. The length was recorded as the pressure varied from 0 to 5 bar. This experiment is intended to capture the important qualities of the muscles we constructed including the maximum percentage of contraction and the relationship between force, pressure and length.

The Second Experiment was designed to test a method of controlling the muscles. Generally the pressure is varied to change the contraction length but it is also possible to control the length with a fixed pressure by pulsing the solenoid controlling the air flow at different duty cycles. This experiment repeated the first experiment but varying duty cycle on the valve controlling the muscles. The results are compared to the first experiment to determine the relationship to PWM signal and the effective pressure in the bladder.

Another experiment was designed to test the compliance of the muscles in series and parallel configurations. Because the McKibbens are inherently compliant and that compliance varies with pressure placing them in series with each other can result in a variable compliance actuator. Two muscles were tested in a series relationship and four in a series parallel relationship.

Finally the McKibbens were attached to a series elastic force sensor to experiment with force sensing techniques and force based controllers. McKibbens can present a unique issue for position and force measurement because the length of the muscle (position) is dependent on both force and pressure. Assuming the driving pressure is known either force or position must be measured in order to determine the other. The force sensor is comprised of an additional spring in series with the McKibbens muscles attached to a potentiometer so its displacement

4.2.2) Results

Two sets of Muscles were printed on the Objet 3D printer using the CADD models the team constructed. The Initial set was an inch and a half in length and a quarter inch in diameter. The weave on these muscles was printed in a range of stiffness available on the Objet. All three weaves were broken during post processing to remove the support material. Although the weaves were damaged the bladders were tested under pressure and broke at 2 bar. A second trial was made after scaling up the model to make the weave more durable the second muscles were 3 inches in length and half inch in diameter the bladder wall thickness was also increased. These muscles survived post processing and the functionality of the weave and bladder was tested by hand, however

the solvent used for removing support material dried the plastic out and it cracked before it could be tested under pressure the bladders were again tested and proved stable to 3 bar with the increased wall thickness. At this point Muscles were constructed from off the shelve materials and used for further testing of the concepts. The 3D printing approach resulted in useable CADD models but the materials and post processing procedures need to be refined in order to preserve the muscles.

The first experiment conducted on the McKibbens Muscles tested the contraction of different muscles length and under different loads. Figure 22 shows the percentage of contraction under different loads of each of the muscles. Regardless of length a pressure of 1.5 bar is required in order to produce considerable displacements this is known as the activation pressure of the muscles. Another observation is that the maximum percentage of contraction is reduced when the size of the muscle is reduced. The result is the smaller muscles loss actuation length both from being shorter and from producing less contraction, a factor that will limit the minimum size of muscles when trying to create series and parallel combinations.

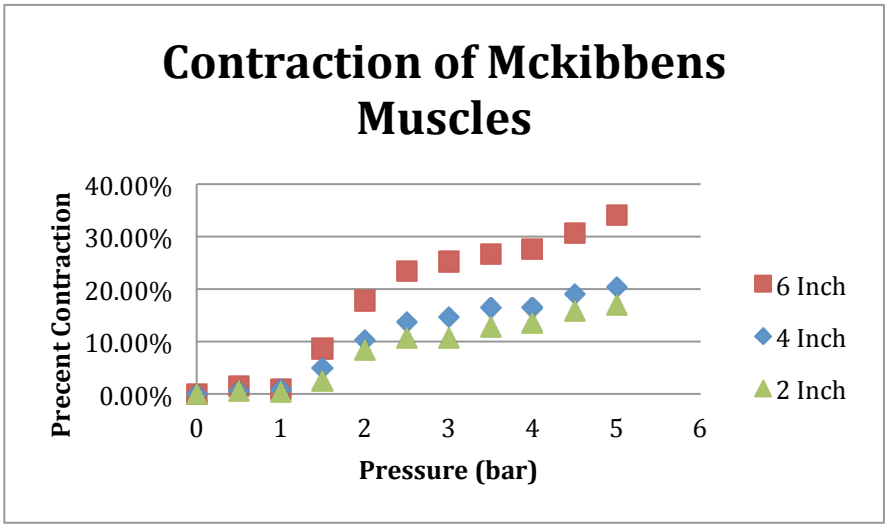


Figure 22: Graph comparing contraction efficiencies of different length McKibbens

Additionally the length of the muscle was recorded under different weights at each pressure this allows the compliance to be calculated as the Change in Length over the applied in Force Figure 23 shows the compliance measurements of the 4 inch muscle

as the pressure is varied from 0 to 5 bar. The compliance tends to decrees as the pressure increases resulting in a variable compliance actuator. With just one muscle this compliance is linked to the amount of contraction. However, when placed in series and parallel configurations the variable compliance of each muscle can be used to create a network in which compliance and position can be controlled independently.

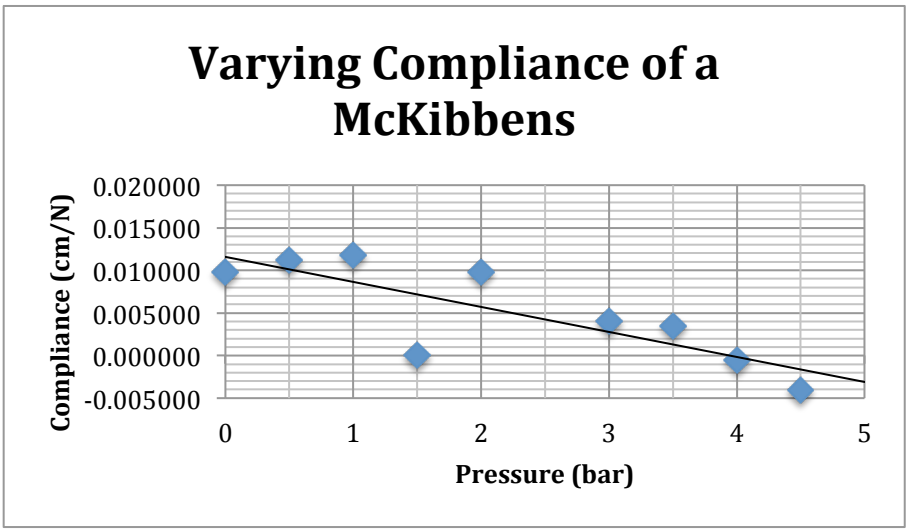


Figure 23: Compliance of McKibben actuator as a function of pressure

A network of four muscles was created with two pairs of series muscles were constructed and then attached in parallel with each other. The pressure and length was measured for each combination of a discreet set of three pressures on the four muscles. Table 4 shows three of these combinations that produced comparable lengths with different compliance values. The compliance can be approximated by treating each muscle as a spring with compliance dependent on it pressure. Compliance add in Series and the inverse compliance or stiffness adds in parallel configurations.

Table 4: Selected data set from series/parallel tests

Pressure 1	Pressure 2	Pressure 3	Pressure 4	Length	Compliance
0	0	2	2	8.1965	.0104

2	2	2	2	8.195	.0098
2	0	4	2	8.1985	.0070

Finally we tested driving the McKibbens by pulsing a constant pressure instead of controlling pressure directly. The controlling solenoids were operated by a square wave with varying duty cycle know as Pulse Width Modulation (PWM) signal. We found that a duty cycle of at least 22% was needed in order to activate the Muscles corresponding to a pressure of 1.5 bar the muscle reached full compression at 78% corresponding to the constant pressure applied which was 4 bar for this test. Over this range the conversion from duty cycle to pressure is closely approximated by a linear function. This allows easy control of the McKibbens using readily available microcontrollers and solenoid valves and eliminating the need for expensive digital pressure regulators.

4.3) Electroactive Polymers

EAP's are polymers, which react to electrical phenomena by altering shape. They can be used as actuators because of their behavior to such in the presence of electricity. There are many different types of EAP's, each having a unique composition and activation mechanism. Only dielectric EAP's were explored in this project.

4.3.1) Methodology

Four experiments were performed to test the viability of EAP's as an actuator for an orthotic device. Each experiment tested an EAP actuator which was constructed slightly differently than the others. All four experiments followed the same basic procedure.

For each experiment, the EAP under test was hung from a test stand and stepped through a number of voltages. The test bench consisted of an inch thick two foot by four foot sheet of acrylic. A high power voltage supply, handheld multimeter, and the test stand were placed on the sheet. The voltage supply was grounded to a nearby water pipe and the indicator voltage was connected to the multimeter which was used to display the high level voltage. The test stand was a solid piece of acrylic which had a six by six inch base with a foot tall trunk and a six inch long plank with two slots at the top. The EAP under test was hung from a slot, and a basket with a displacement indicator and weight was hung from the end of the EAP. A variety of weights were used depending on the size of the EAP under test; they ranged from 500g to 700g. Once connected to the high voltage output, each EAP was subject to increasing levels of voltage until the maximum output of the voltage supply was reached, or the EAP failed, whichever occurred first. The displacement for each voltage level was observed and recorded.

The first EAP consisted of a single strip of 3M approximately 1.5 inches in length. This piece of tape was stretched width-wise for approximately 12 hours. The tape was removed from the stretching vise and two acrylic vices were affixed to either side. A conductive aluminum tab was affixed to either side of the tape, and a piece of paper with conductive grease was placed on either face of the EAP to serve as the electrode.

The second EAP was constructed with two pieces of tape, each approximately one inch in length. These were not stretched beforehand. The two pieces were clamped with the acrylic vices at either end such that the two pieces were side by side. The EAP was then painted directly with conductive grease using a fine tipped paintbrush.

The third EAP was constructed similar to the first, but with improvements in the manufacturing process, and the tape pieces used were approximately 2.5 inches in length. First, the tape pieces were stretched slightly before being placed side by side and were then pressed together. Second, the aluminum tabs were placed opposite one another on the EAP. Third, pieces of parchment paper were used as grease mask to keep the grease electrode uniform and well removed from the edge.

The fourth EAP was markedly different from all of the previous versions. In an attempt to increase capacitance, the fourth EAP was rolled in a fashion similar to dielectric capacitors. It was also significantly larger than any of the previous versions using an approximately 36 square inches of VHB tape. First, a sheet of tape was made by taking six strips of tape, each was six inches in length, and placing them slightly overlapping with one another on a sheet of parchment paper. Another piece of parchment paper was placed on top of the square of VHB material, and this was then compressed by a mechanical press in an attempt to fuse the tape strips together. Next, half of the VHB sheet was painted with conductive grease using masking to constrain it, and the sheet was folded back on itself enveloping the grease. The now exposed back half of the VHB sheet was painted with conductive grease using masking, and the sheet was then rolled up and capped with a semi flexible epoxy.

4.3.2) Results and Discussion

Several experiments were performed using the EAPs. The first experiment used a single piece of pre-strained tape. When testing with this tape, we found that at around 2000 volts, electricity began to arc between the two sides of the tape. At approximately 2500 volts, the tape began to tear near source probe. The second test performed was with two pre-strained strips of tape. The two strands began to arc at the seam of the two pieces of tape at approximately 4000 volts once again. It then began to tear at approximately 5500 volts. The third test was similar to the second one, but parchment paper was placed on the sides of the tape while the grease was applied. When 6000 volts was applied to the tape, it began to stretch. The tape stretched approximately 2mm. The tape then shrank when the voltage was turned down. At 6500 volts, the tape arced and tore again. The last test performed on tape that was laid close to each other, and then pressed together. The tape was then greased and rolled up.

During testing the EAPs, several flaws during the set-up were discovered. The first flaw was when the grease was applied to the single strand of the tape. The grease was not carefully applied, and as a result, connected the ground side and power side of the tape together. This caused the arc that occurred on the sides of the tape, and

eventually resulted in the tearing. The second test produced a similar error in the center of the two strands.

Another flaw later discovered was testing the tape using weights attached to the bottom of the roll. This design was used to try to mimic placing a spring in the center of the roll. This resulted in a failure, since the weight attached to the bottom would result in different expansion and compression forces than the spring centered in the roll would produce.

4.4) Cable Driven Actuators

Cables were attached to different locations on the arm, found on both the anterior_ and ventral sides. Motors, controlling these cables, will release and pull certain lines to direct the arm to a desired location.

4.4.1) Methodology

When reviewing the necessary design alternatives for the design of the cabling mechanism, analysis of three designs provided the necessary feedback to select the correct spool system that works with the Bowden cable system. The needed functionality of the cabling system is to provide accurate actuation with respect to the desired location given by the input. Next, the cabling mechanism must regulate the motor based upon force. Finally, the cabling system needed to provide an appropriate safety features to prevent the application of high forces on to the user.

Spool System

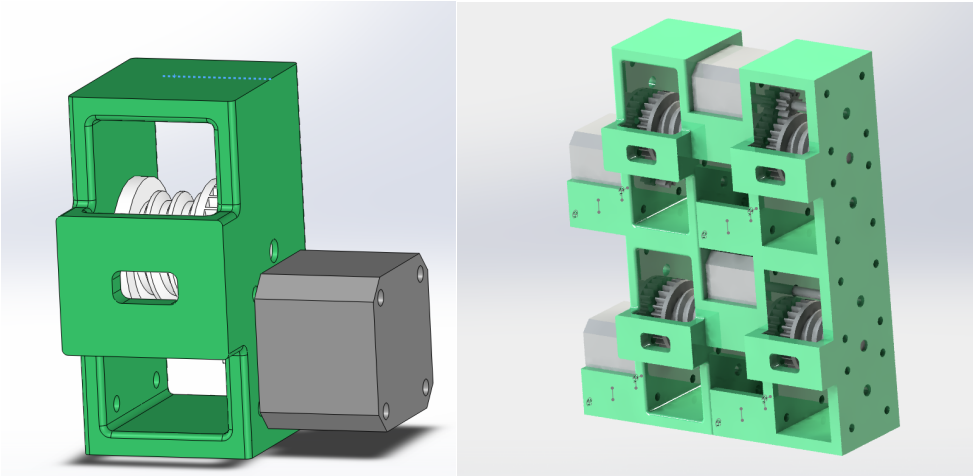


Figure 24: First design on the spool system

The first proposed system was a self-advancing cabling mechanism using a threaded spool. Using a threaded screw with a pitch of 1.5 degrees and spacing of the 10mm, the rotation of the spool would ensure continuous advancement about the spool, as the selected rope would align with the thread of the spool. A major advantage of this system was that a threaded spool prevents rope overlapping. Another advantage of this spool system was that this design was able to be stacked into a grid to save space to accommodate additional motor drivers for actuation.

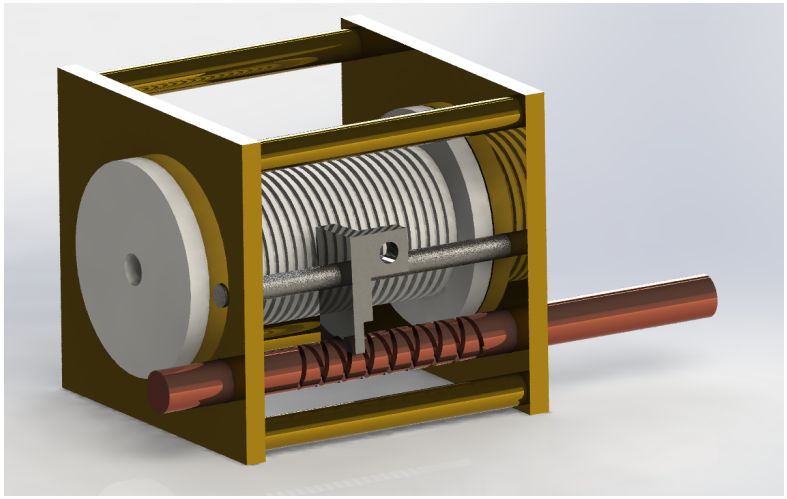


Figure 25: Second design of the spool system

The next proposed system was designed to auto-advance the cable collection system. The approach for this design was to use a driven screw to advance the spool based upon the spool's placement. By using a diamond screw the mechanism could actuate in both directions. This was a result of the diamond screw having a continuous thread. The spool also featured a 1.5 degree threaded screw to collect the cable driven by a stepper motor. An advantage of this cable collection system is that, if implemented correctly, it would prevent tangling and overlapping of the spool. A major disadvantage of this design would be implementing the complex timing to accurately ensure the spool would rotate at the same speed that the collection spool actuates.

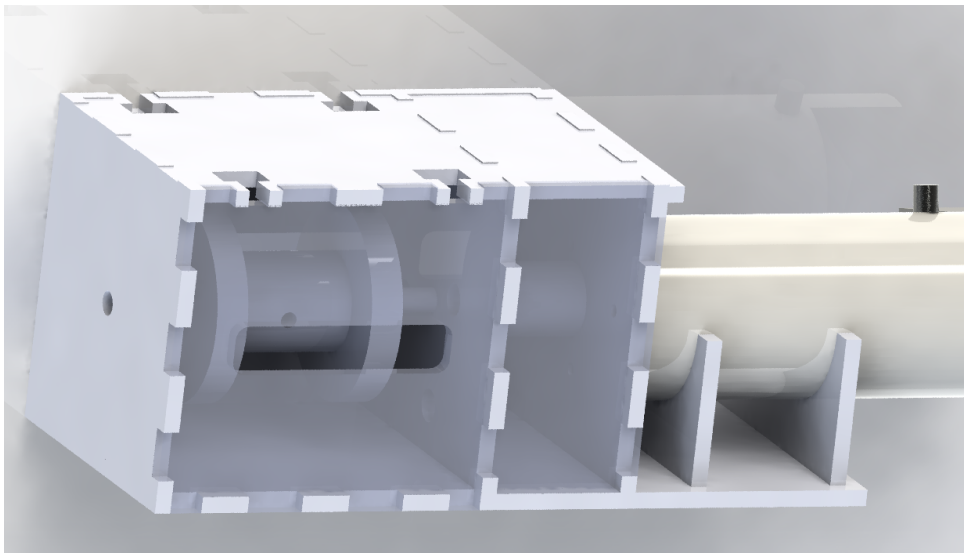


Figure 26: Third design of the spool system

The final system proposed used a smooth spool directly driven by a motor. The approach for this design was to develop a very simple and inexpensive module that could be customized to accommodate different types of motors.

Motor Selection

Actuators were selected to meet the necessary properties for this design. The motors for this system were required to have the necessary torque to actuate an arm for both flexion and extension of the elbow and supination and pronation for the wrist. To

ensure the safety of the user, the motors needed to be able to accurately actuate according to the given input from the user.

Using a physiological model the weight of the arm was determined to range between 8-12 lbs. After developing a force diagram, which analyzed the forces working upon the arm, three force components were generated: $F_{Bicep}, F_{tricep}, and\ F_{arm}$.

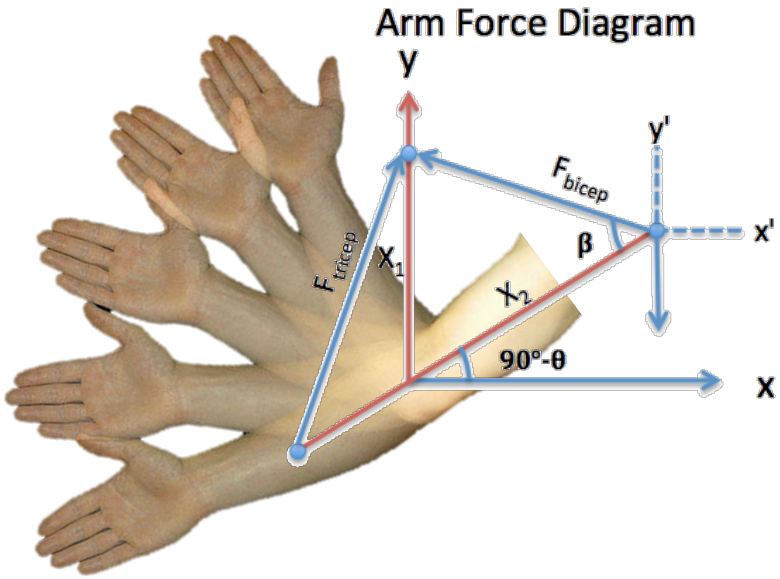


Figure 27: Free Body Diagram of forces applied to the arm

The force from the bicep could be arranged into a 2x2 rotation matrix, which generated the angular displacement of the force from the bicep with respect to the origin (located at the elbow).

$$R_1^0 = \begin{bmatrix} sin(\theta) & -cos(\theta) \\ cos(\theta) & sin(\theta) \end{bmatrix} \text{ (Eqn. 1)}$$

$$R_2^1 = \begin{bmatrix} sin(\beta) & -cos(\beta) \\ cos(\beta) & sin(\beta) \end{bmatrix} \text{ (Eqn. 2)}$$

$$\begin{bmatrix} F_{(x)Bicep} \\ F_{(y)Bicep} \end{bmatrix} = R_1^0 R_2^1 = \begin{bmatrix} -c(\theta+\beta) & -s(\theta+\beta) \\ s(\theta+\beta) & -c(\theta+\beta) \end{bmatrix} \begin{bmatrix} 0 \\ F_{Bicep} \end{bmatrix} \text{ (Eqn. 3)}$$

$$\sum F_y = 0 = -F_{arm} + F_{Bicep} \text{(Eqn. 4)}$$

$$F_{Bicep} = \frac{F_{arm}}{-c(\theta+\beta)} \text{(Eqn. 5)}$$

Equation 5 was graphed as a force and torque analysis. The torque was determined by multiplying the force obtained by the radius of the spool. The radius of the spool was set to five inches.

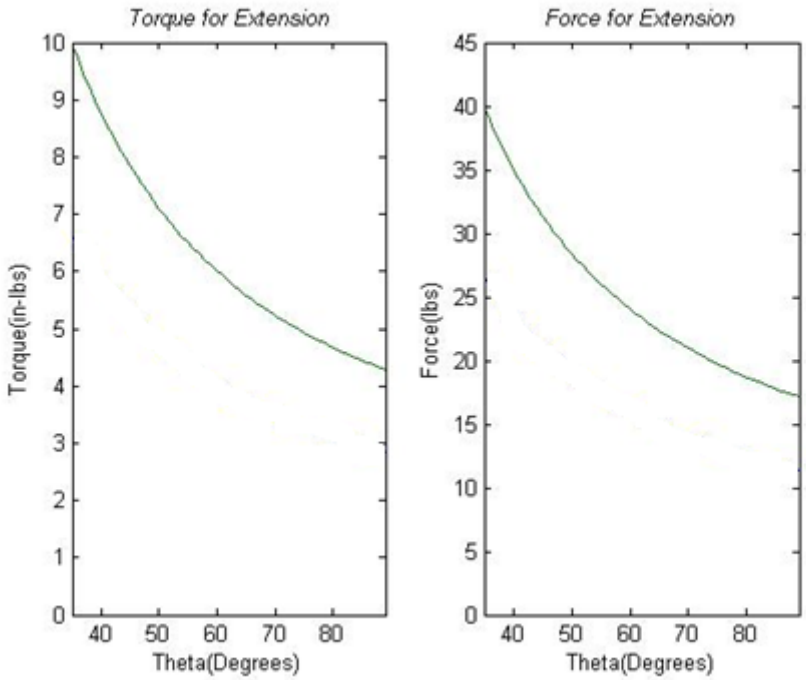


Figure 28: Torque and Force Analyses.

As seen in Figure 28, the torque necessary to actuate the elbow during flexion was determined to be 10 in-lbs. As displayed above, as the angular position decreased, the amount of force necessary increased. The maximum force was determined to be when theta was equal to 35 degrees, because this was the minimum angular position according to the physiological model. To allow the user to manipulate small items while wearing the brace, an additional 5 in-lbs were factored into the total torque calculation. The determined torque needed was determined to be 15 in-lbs.

With the calculated torque necessary for the motor, three types of motors were selected as design alternatives: stepper motors, brushed dc motors, and brushless dc motors.

4.4.2) Results

Testing was conducted to determine if the selected motor would drive the cables effectively. The goal of this experiment was to determine the motor that had the best torque, and smallest size and weight. To perform this experiment, three motors were selected and tested: a 32 oz-in Stepper Motor from Sparkfun, a 100 oz-in DC Globe Motor, and a 120 oz-in Stepper Motor from Lin Engineering. . During the experiment, the brace was mounted on the mock arm with three different weights, 4 5, and 6 lb attached to the center of the forearm. The weights were selected to simulate the necessary force a single motor would need to reach during this experiment.

Table 5: Comparison of motors tested

Weight	Lin Engineering 120 oz-in Stepper	Sparkfun 32 oz-in Stepper	DC Globe Motor 100 oz-in
4 lb	Yes	No	Yes
5 lb	Yes	No	No
6 lb	Yes	No	No

Based on the first experiment the Lin Engineering Stepper motor was capable of lifting the most torque at stall.

Since these motors need to fit within a small profile, the overall dimensions of the motor were assessed. The volume of each motor was calculated and compared:

Table 6: Volume and weight comparisons of motors tested

Dimensions	Lin Engineering 120 oz-in Stepper	Sparkfun 32 oz-in Stepper	DC Globe Motor 100 oz-in
Volume (mm^3)	106,942 mm^3	55,219 mm^3	153,231 mm^3
Weight (kg)	0.41 kg	0.20 kg	0.32 kg

From this evaluation, both the Lin Engineering Stepper and Globe Motor had tradeoffs; the Stepper motor was heavier, while the Globe Motor had a larger volume.

The stepper motor was selected for the final project, but a major flaw with this device was controlling the motor. Unlike DC motors, steppers need a separate motor controller. The Lin Engineering Stepper Motor required a 36 V, stepper driver which would continuously output 2 A. This required a customized PCB board, which took longer to implement.

4.5) Comparison

After experimenting with all three actuators we could compare our findings and select the most suitable actuator for our project. The Electro-activated polymers were extremely light weight and had a very high power to weight ratio. Additionally they consumed little power compared with other actuators. Unlike McKibbens and the Bowden Cable System the EAPs do not require the addition of intermediate systems such as air compressors or cable spools, all that is needed is a variable voltage regulator. Over 1000 volts is required to generate substantial contractions. These extreme voltages add additional complexity to designing a safe system. However, the current is typically very small making this possible. After several attempt at constructing and testing EAPs it was determined that they were too difficult to manufacture and that homemade actuators did not achieve large enough contractions. Both commercially available actuators of this type and better materials to construct them were prohibitively expensive. The McKibbens actuators were tested both individually and in networks of series and parallel actuators. These had many properties which made them desirable for our project. They are light weight and with the exception of a compressed air system are self-contained on the arm resulting in an excellent form factor. There are natural limits on the amount of compression that can be generated which can easily be used to limit the device to the natural range of motion of the arm. Additionally they are inherently compliant a highly desirable property for applications in soft robotics. The major disadvantage of these actuators is there need for a compressed air sources which would add complexity and weight to the system although it could be stored in a backpack off the arm. When placed in networks an ability to control compliance is gained, however the controls model associated with this is quite complicated requiring the pressure and length of each muscle

to be known and modeled appropriately. The McKibbens were also tested with series elastic force sensors and found to be inaccurate if the compliance of the muscle at its current pressure and the displacement of the muscle were not factored into the calculations. Finally the Bowden cable system was implemented and tested the major advantage of this system is that it allows virtually all the weight to be moved off the arm into a backpack compensating for a low power to weight ratio. Unlike McKibbens and EAPs the Bowden cable system can actuate continually requiring additional mechanical systems to achieve the desired limits on the range of motion. However the cabling system had well known solutions for both force and position sensing which complimented our hybrid force-position controller. In the end this system proved to be the most practical system and Bowden cable were implemented on the final device.

5) Sensors

An inertial measurement unit (IMU), which measures forces and position, and EMGs, velocities from muscle movement, were evaluated for desired actions of the arm.

5.1) Design Alternatives

One of the major objectives in designing the powered orthotic was returning voluntary control of elbow flexion-extension and forearm pronation-supination to the user. This means that there must be some way for the user to provide input to inform the sleeve about their intended movement. There are many sensors which are commonly used to allow a user to give input to a powered orthotic device. These can range from inertial measurement units (IMU's), which gather data about the user's motions, to electromagnetic sensors which can pick up the user's biological signals, to simple pressure sensors which are activated by exhaling into a tube. After considering the constraints on the sensor, three alternatives were considered to allow user input to the powered orthotic.

5.1.1) Objectives and Constraints

The objective of integrating a sensor capable of gathering input from the user is to provide an intuitive way for the user to control the device. Ideally, the sensor would be able to gather this information in such a way that using the device will put a very small mental load on the user. The sensor is further constrained by the fact that the intended user is not a healthy individual, and does not have complete, voluntary control over his or her arms. Additionally, the input device was limited by the time and monetary constraints of this project.

1. the input data must be complete enough to control the two degrees of freedom in the elbow and forearm with enough accuracy to complete simple tasks
2. the input data cannot originate from the movement of the user's elbow

3. the input data should originate from a source which is a natural consequence of the user attempting to move their elbow
4. the sensor must be human safe
5. the sensor must be portable
6. the sensor must be unobtrusive
7. the sensor must be available for under $300
8. the sensor must be able to be developed in under six months

5.1.2) Possible Solutions

It was determined that there were three options which could best meet these constraints. They were an IMU, to gather information about the user's shoulder movement; electromyography (EMG) sensors, to detect residual weak muscle activity; and electroencephalography (EEG) sensors, to detect brain activity.

The first option was the Inertial Measurement Unit. Depending on the site of the injury, the user can retain some voluntary control over his or her shoulder. As discussed in the Interconnected Joint Analysis section, for specific actions, the movement of the shoulder can be very closely correlated to the movement of the elbow. This linked motion can be taken advantage of for individuals who have received damage to the C5-C6 vertebrae and who may retain voluntary control of their shoulder. An IMU can be used to measure the motion of the shoulder, and then this information, in addition to a specific action, can be used to generate the appropriate motion of the elbow. The advantage of this system is that it is fairly intuitive, and unobtrusive for the user. It is also a relatively simple system to implement given the proper tools. However, this system would require that the user have some voluntary control over his or her shoulder. Additionally, without more sensors to differentiate between motion of the user's body and motion of the user's shoulder, the device would only function when the user is stationary.

Another possibility was sensing biological signals with EMG sensors [45] [46]. Depending on the severity of the injury, the user may retain some efferent motor nerve commands to the muscles of their arm (e.g., biceps and triceps muscles). Although the number of active motor units is frequently too inadequate to produce functional movement in a tetraplegic's arm, the resulting EMG signals can still be picked up by EMG sensors placed on the tetraplegic's skin. This system would be very intuitive because it is essentially translating the user's intent to movement. The system is more obtrusive because it would require an array of sensors to be placed over the user's arm. Also, this system requires that there be at least enough residual muscle electrical activity to be reliably detected by the EMG electrodes.

The third option was EEG sensors. Similar to EMG sensors, EEG sensors can be used to gather biometric signals, but these signals would arise from neural activity in the brain instead of muscle contraction. The electromagnetic waves produced by the user's brain when performing some action can be recorded by placing electrodes on the user's scalp. This would allow the sleeve to "read" the user's mind when the user desired to lift his or her arm. This system would be extremely intuitive, and it would not be inhibited by the location or severity of the user's spinal trauma. Unfortunately, an EEG driven system would be extremely difficult and expensive to implement, and have not yet been proven to be reliable in prosthetic or orthotic applications

5.2) Methodology

Due to the many different variations of both how sensors are designed and how they may be applied it is imperative to understand both the design and testing of these units.

5.2.1) EMG Design

It was determined that a non-commercial, custom-made EMG acquisition system would best satisfy the requirements for this project. Although commercially available acquisition systems would ensure a level of quality and robustness, and would eliminate

the need to design a new system, the drawbacks of a commercial system were not worth the investment. A commercially available system would be substantially more expensive than a custom-made system. Delsys offers an eight channel system for $7,900 and Motion Labs offers a 12 channel system for $12,950.00. We estimate that we can make a system that will fit our needs for under $500. Additionally, the majority of commercial EMG acquisition systems are desktop solutions and are not intended for portable applications. By designing a custom system, the hardware could be tailored to the specific requirements and constraints of the project while cost could be minimized. Furthermore, the expertise and equipment necessary to design and construct an acquisition system were readily available, so the task was not an excessive burden on the project.

The design of the custom EMG acquisition system was split into four sub designs. These were the mechanical components which contacted the user's skin (electrode), the first stage of amplification (pre-amp), the second stage of amplification and signal filtering (signal conditioning), and the digitization of the signal. Figure 29 shows a block diagram of the full acquisition system.

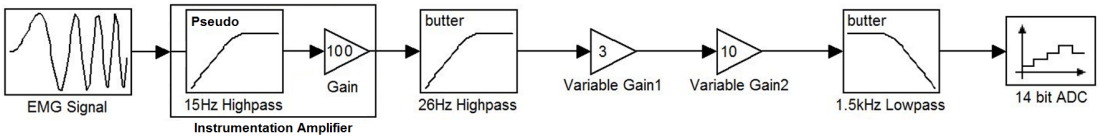


Figure 29: Block diagram of EMG signal processing

The purpose of the pre-amp was to amplify the muscle signal as much as possible before any significant noise could enter the system. Five instrumentation amplifiers were considered for the pre-amp. Instrumentation amplifiers were selected for the pre-amp because they have a very high input impedance which minimally disturbs the charge on the skin, and because instrumentation amplifiers maintain a high signal to noise ratio because of their common mode rejection. The five amplifiers considered were the INA116, INA826, and INA101 from Texas Instruments, Dallas, TX, and AD620, and AD8221 from Analog Devices, Norwood, MA. These five chips were considered because they were capable of producing a unique frequency response. This feature

allowed the team to consider higher gain settings for the pre-amp. The first three chips had relatively high noise ratios, over $13\frac{\boldsymbol{nV}}{\sqrt{\boldsymbol{Hz}}}$, and the last two were both under $10\frac{\boldsymbol{nV}}{\sqrt{\boldsymbol{Hz}}}$. The AD8221 had slightly better performance characteristics and was selected over the AD620.

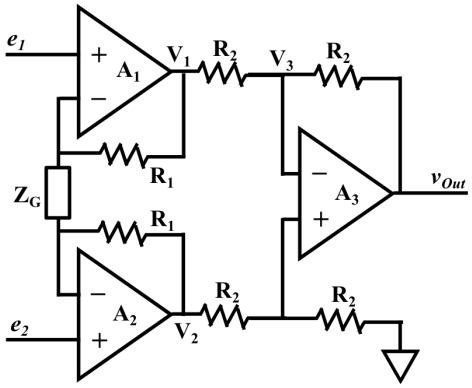


Figure 30: Analogous circuit diagram to AD8221 amplifier

A previous design that this system was based on, and was developed by a team under the direction of Edward Clancy, used a gain of 20 [31]. However, higher gains of 50 or 100 are possible with appropriate circuit design. By using a complex impedance for the gain selection, the chips would amplify a signal less at lower frequencies than at higher frequencies. This effect is described by equations 5.1 through 5.3. A higher gain in this section of the circuitry would result in increased signal to noise ratio. This was achieved by implementing a pseudo high-pass filter with the AD8221 using a 250 ohm resistor in series with a 33uF capacitor and a 4.7uF capacitor in parallel to select the gain. Otherwise the DC offset of the muscle signal would saturate the amplifier and completely obscure the signal.

$$\frac{V_{out}}{e_1 - e_2} = 1 + 2\left(\frac{R_1}{R_G + \frac{1}{jwC_G}}\right) \tag{5.1}$$

$$R_G = \left(\frac{2R_1}{G-1}\right) \tag{5.2}$$

$$C_G = \left(\frac{0.707G^2 - 1.707G + 1}{0.586GR_1 * (2\pi) * f_b}\right) \tag{5.3}$$

The signal conditioning served to filter noise outside of the EMG bandwidth, and amplify the signal further so that it utilized the full voltage range of the analog-to-digital converter (ADC). The amplification was implemented in two cascaded stages using two non-inverting amplifiers and two potentiometers. The first stage was for fine gain control with a maximum gain of 3, and the second stage was for rough gain control with a maximum gain of 10. The filter was a band-pass filter implemented as a second order high-pass cascaded with a second order low-pass. The cutoff frequencies of the filters were 26 Hz and 1500 Hz, respectively. The 26 Hz cutoff was chosen to eliminate the DC offset and inherent instability present in the muscle signal and the motion artifacts that may result from slight shifts in the electrode position. The 1500 Hz cutoff was chosen to prevent aliasing of the signal when the signal is digitized because EMG signals contain very little power beyond 1500 Hz. Both high-pass and low-pass filters featured Sallen-Key topography, and utilized the TL084 op-amp. The resistor and capacitor values were selected using a matlab toolbox, which was written for Sallen-Key filter design, and is described in [31]. The signal could be further conditioned by a 60 Hz notch filter, however, these are typically implemented in software for EMG acquisition systems', and they are not necessary for this project. Consequently, a notch filter was not implemented in hardware and was left as an optional function of the microprocessor.

The last electrical component to be selected for the EMG acquisition system was the ADC. The ADC had to have a 9v bipolar range to accommodate the full range of the amplifiers and had to be capable of a minimum of 3ksps to satisfy the Nyquist frequency. Additionally, a 14 bit range was desirable to achieve a good resolution for the data. A serial peripheral interface capable device would be desirable because SPI is a simple, fast communication protocol. However SPI protocol was not completely necessary. The AD-7367 produced by Analog Devices was chosen to satisfy these requirements. It was relatively inexpensive compared to other high resolution ADC's on the market at $12.32, and it met or exceeded all the requirements. The chip was not conformant to SPI, but it did communicate over serial, which was acceptable. Also, the chip had two multiplexed channels, which was beneficial because it allowed up to four channels to be placed on a single board.

The last electrical component to be selected for the EMG acquisition system was the ADC. The ADC had to have a 9v bipolar range to accommodate the full range of the amplifiers and had to be capable of a minimum of 3ksps to satisfy the Nyquist frequency. Additionally, a 14 bit range was desirable to achieve a good resolution for the data. A serial peripheral interface capable device would be desirable because SPI is a simple, fast communication protocol. However SPI protocol was not completely necessary. The AD-7367 produced by Analog Devices was chosen to satisfy these requirements. It was relatively inexpensive compared to other high resolution ADC's on the market at $12.32, and it met or exceeded all the requirements. The chip was not conformant to SPI, but it did communicate over serial, which was acceptable. Also, the chip had two multiplexed channels, which was beneficial because it allowed up to four channels to be placed on a single board.

The electrode was designed with functionality and ergonomics in mind. Previous revisions of these electrodes were rectangular prisms with 8mm stainless steel screw heads for contacts. This shape permitted the metal screw heads to stick far enough out of the electrode package to contact the skin. However, the electrode cases that were produced for this project were made to accommodate long shafts of the screws and their securing nut. These contacts were sized and spaced for optimal bandwidth detection. The bottom of the case, the face which contacts the skin, had two parallel peaks. This shape allows for maximum surface contact with the skin and good mechanical stability and is shown in Figure 31.

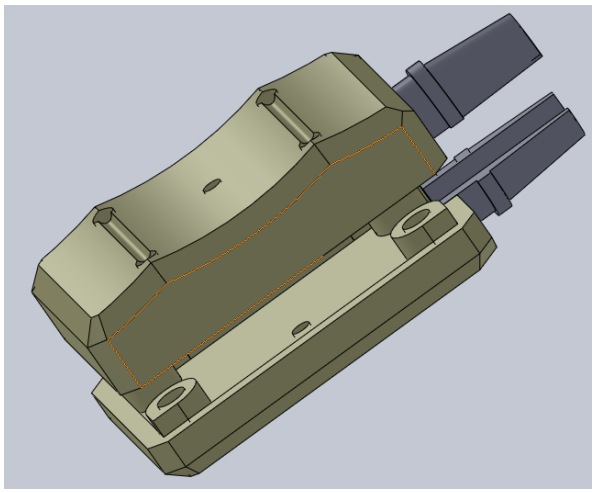


Figure 31: Electrode case design with 'wave' style surface

A significant design consideration for the EMG electrode was the type of metal which contacts the wearer's skin and collects the EMG signal. Three metals are commonly used as EMG electrodes: copper, silver, and stainless steel. Both copper and silver form good electrical contact with lead solder (on the pre-amp circuit board) which makes these metals ideal for connecting to printed circuit boards (PCB). Copper, however, is less frequently used in electrodes because allergic reactions, although rare, can occur. Copper can also stain skin a green color after prolonged periods of contact. Silver has similar issues with both allergic reactions and skin discoloration, but to a lesser extent. Stainless steel is difficult to solder well because it forms a hard oxidized shell very quickly which can inhibit electrical connectivity. However, because of this oxidized shell, stainless is a fairly inert metal so allergies are very mild and slow to appear.

5.2.2) EMG Testing

The testing of the EMG acquisition system happened in three stages throughout the development process. First, the design of the filtering and amplification was tested in simulation. Next, a physical prototyped circuit was tested as to its ability to acquire and filter a signal. Finally, the full circuit was tested to ensure the functionality of the components and printed circuit board (PCB).

The first stage of testing validated that the filter design of the Sallen-Key filters and amplifiers functioned properly. Multisim was used to model the circuit and test it. The filters were tested using a sine wave frequency sweep from 1 Hz to 4k Hz with an amplitude of 1 V pk-pk. The amplifiers were tested with a virtual oscilloscope, and the output voltage for a 500 Hz input was observed as the potentiometers in the amplifiers were varied.

The second stage of testing verified the function of the physical circuitry and the circuit's ability to acquire an actual EMG signal. Each stage was tested independently on the breadboard. The pre-amp was tested by connecting the output to an oscilloscope and attaching the inputs of the instrumentation amplifier to Vermed resting electrodes which were placed on a subject's arm. The high pass and low pass filters were tested by sweeping a function generator through a range of frequencies and observing the output amplitude of the filter compared to the input amplitude. The high pass filter was tested at frequencies of 14 Hz through 36 Hz in steps of 2 Hz, the low pass filter was tested at frequencies of 800 Hz through 3k Hz in steps of 200 Hz, and a wave of amplitude 200 mV pk-pk was used for both tests. The amplifiers were tested by observing the output of the two stages for a 1k Hz, 200 mV input as the potentiometers were varied. Lastly, the pre-amp and conditioning circuitry were used together to acquire an EMG signal from a subject's biceps muscles. An analysis of the bandwidth and of the power spectrum of this signal was then performed using the fft and pwelch functions in matlab.

The last stage of testing ensured that the hardware had been designed and assembled properly. The full circuitry and electrode contacts, with the exception of the digitization portion, were used to acquire an EMG signal from a subject's biceps muscles. A Tectronix oscilloscope was used to acquire the signal at a sample frequency of 4k Hz. An analysis of the bandwidth and of the power spectrum of this signal was then performed using the fft and pwelch functions in matlab.

An attempt was made to copper plate the tips of stainless steel staples in order to increase conductivity. A copper electrode was immersed in a bath of copper sulfate solution, and a stainless steel staple was suspended with the tips immersed in the bath as

well. The tips of the staple were polish with an abrasive before immersion in the plating solution. Figure 32 shows an image of the plating setup. A voltage supply was used to drive current through the plating set up.

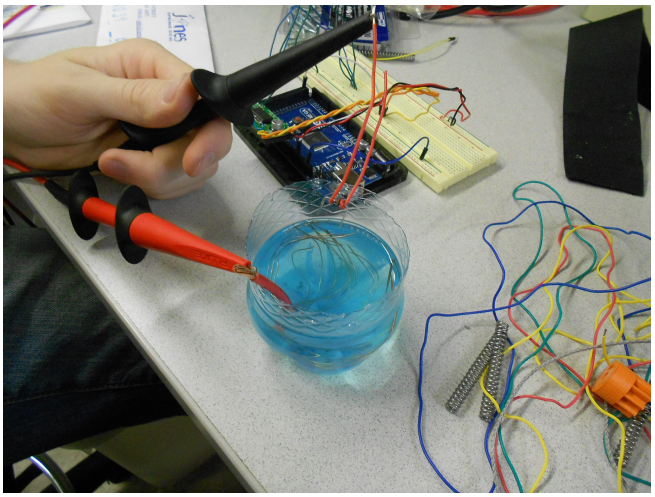


Figure 32: Copper plating set up

5.2.3) Inertial Measurement Unit Methodology

The methodology behind the use of the Inertial Measurement Unit (IMU) relied on the linking between shoulder motion and elbow flexion and extension. As seen in the Interconnected Joint Analysis section of the Literature Review, three dimensional camera mapping was able to find high correlations between movements in the front and sagittal planes, almost to the point of being linear.

Early testing was carried out with a 3D camera system and mimicked the studies. The model was outfitted with a retro-reflective ball array on his shoulder and lower arm and asked to drink from a cup. The binary motion capture camera system tracked three markers arrayed in a circle on the shoulder and 4 arranged in a "Y." This setup would allow the camera to record shoulder motion cleanly between the three points and give a comparison to the motion of the lower arm.

Moving forward from camera testing, an IMU was obtained. This allowed for easy data collection without a large amount of setup, as well as less constrained motion as the user would not be confined to the camera system's field of view. Preliminary tests

were conducted using just the IMU, in order to collect data on shoulder movement without interfering. Angles were marked out on a wall and a subject was made to stand in front and keep his arm constrained to an angle while moving his shoulder. This would allow the IMU to collect data in several distinct and measurable orientations and provide a base point for future investigation.

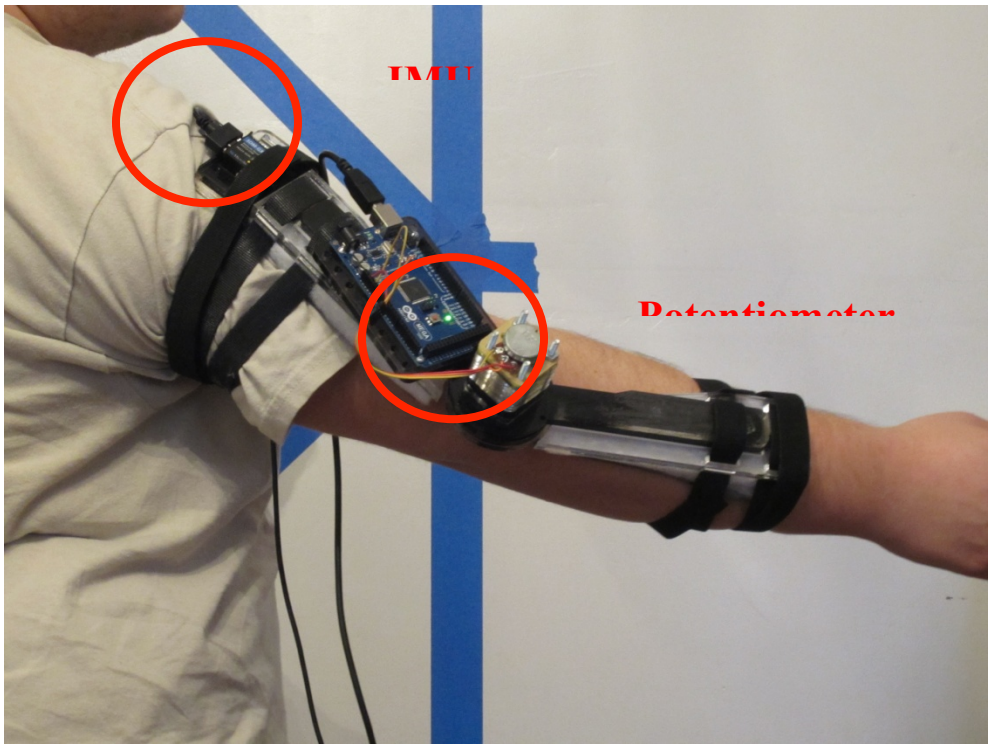


Figure 33: IMU and Potentiometer Mounting

The final tests for the IMU were carried out using a combination of the shoulder mounted IMU with an elbow mounted potentiometer. Potentiometer values were streamed through either an mBed or Arduino microprocessor then back to a computer while the IMU streamed Acceleration, Angular Rates, and an Orientation Matrix. From the IMU data, the exact position of the shoulder is recorded as well as any changes in movements. Coupling this with the data streamed from the potentiometer allowed for accurate motion capture.

5.3) Results

The first test to acquire an EMG signal using just an instrumentation amplifier and electrode pads yielded satisfactory results. Figure 34 shows a side by side comparison of oscilloscope captures. The image o the left shows the output of the operational amplifier while the test subject's arm was at rest. The image on the right shows the output while the test subject was flexing their bicep. In the first there is clearly no signal being acquired, while in the second there is a discernible signal being output by the oscilloscope.

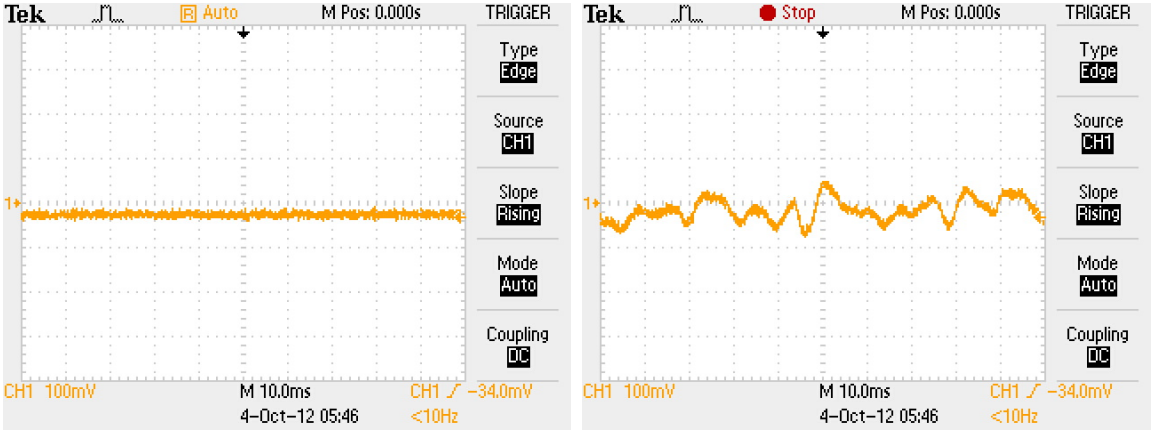


Figure 34: The left image is an oscope capture with no activity, the right is from contraction of the biceps

The second test to confirm the proper characteristics of the filtering was also met with satisfactory results. Table 7 displays the numerical data recorded from filter testing. Although there was a fair amount of discrepancy between calculated, simulated, and actual break frequencies, all results were satisfactory for our purposes. Figure 35 shows an EMG signal in the frequency domain acquired through a breadboard implementation of the full circuit.

Table 7; Table of frequency response data for the high and low pass filters

high pass results						low pass results				
f (Hz)	Vin (mV)	Vout (mV)	G	G (dB)		f (Hz)	Vin (mV)	Vout (mV)	G	G (dB)
14	98	44	0.449	-6.955		3000	88	30	0.341	-9.347
16	98	50	0.51	-5.845		2800	88	32	0.364	-8.787
18	98	56	0.571	-4.861		2600	88	36	0.409	-7.764
20	98	60	0.612	-4.261		2400	88	38	0.432	-7.294
22	98	64	0.653	-3.701		2200	88	44	0.5	-6.021
24	98	66	0.673	-3.434		2000	88	48	0.545	-5.265
26	98	70	0.714	-2.923		1800	90	54	0.6	-4.437
28	98	74	0.755	-2.44		1600	90	60	0.667	-3.522
30	96	78	0.813	-1.804		1400	90	66	0.733	-2.694
32	96	78	0.813	-1.804		1200	90	74	0.822	-1.7
34	96	82	0.854	-1.369		1000	90	82	0.911	-0.809
36	96	84	0.875	-1.16		800	92	86	0.935	-0.586
10k	115	120	1.043	0.3697		10	100	102	1.02	0.172
	break frequency						break frequency			
	simulation		15				simulation		1.2k	
	calculated		20				calculated		1.7k	
	actual		26				actual		1.5k	

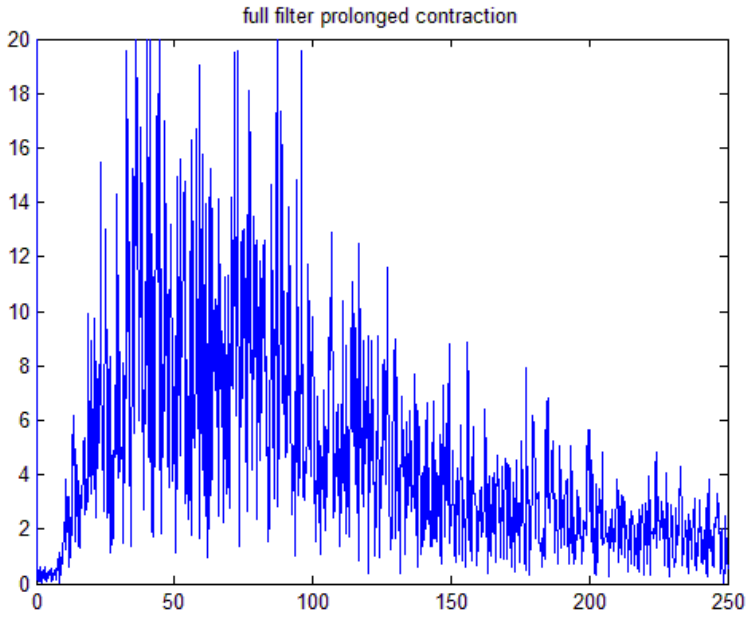


Figure 35: Frequency analysis of EMG signal acquired from full breadboarded circuit

The last test to confirm the functionality of the full PCB circuit was very successful. Figure 36 shows a comprehensive representation of an EMG signal that was acquired through the PCB circuitry. The first graph shows the signal in the time domain, the second shows the signal in the frequency domain, and the third shows the power density of the signal. The power density shows the characteristic lump from approximately 10 Hz to approximately 400 Hz that slowly wanes as it approaches 2000 Hz which is typical of EMG signals.

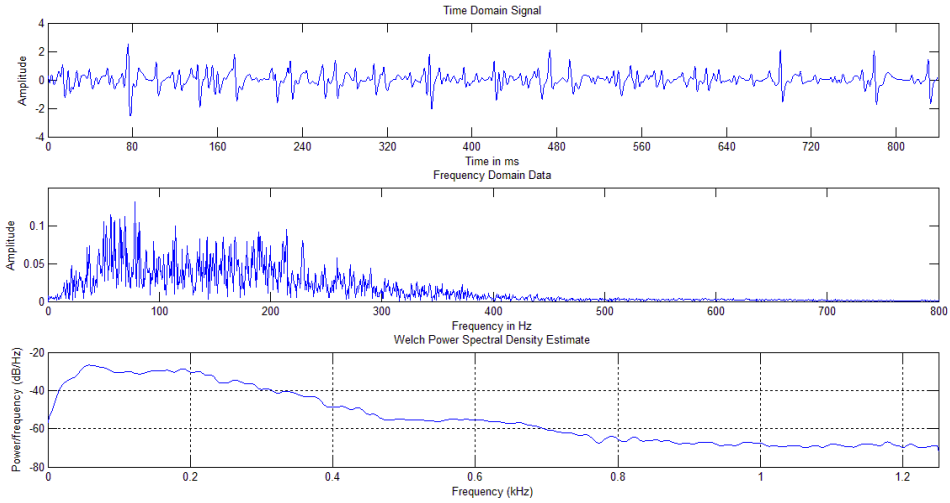


Figure 36: Graph of EMG signal in time domain, frequency domain, and power distribution respectively

Copper plating the electrode tips was met with moderate success. The first attempt produced a blackened, brittle plating on the electrode. However, by turning the voltage down and thereby reducing the current, a shiny, smooth copper plating was achieved on the electrode tips. This plating turned out to be too thin to withstand the mechanical wear on the electrode because the plating wore off after being rubbed slightly. Further plating was not performed because soldering the electrodes directly to PCB copper yielded satisfactory results in terms of conductivity.

The IMU testing did not turn up worthwhile results. Because the IMU was borrowed from another project group, extensive testing could not be carried out with it. The tests that were performed began to show a correlation, but there was simply not enough data to make a meaningful analysis and draw up results.

6) Software & Systems

The entire system sends and receives information to and from a central computer that makes sure the entire system operates correctly.

6.1) System Architecture and Software

The main pieces in the software architecture included a Raspberry Pi as the central communication tool and two mbeds. One mbed controlled EMG information and the other controlled the actuators. Figure 37 shows a diagram of the system architecture.

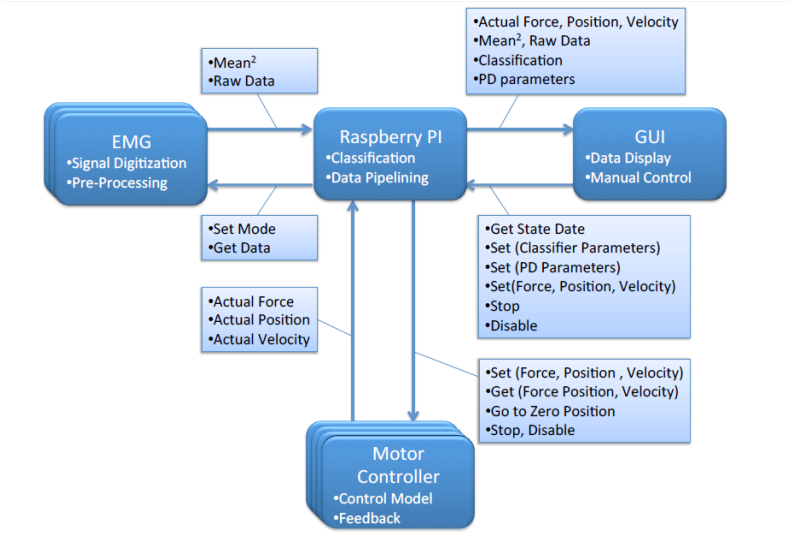


Figure 37: System communication pipelining

6.1.1) Design Alternatives

For our computer systems we needed to pick a system capable of handling and managing the complexity of the other subsystems (Classification, Sensors, Motors, and User Interfaces). Two models were possible, either a single central computer or several smaller computers. The metrics for judging these systems were modularity, ease of development, and system integration. Modularity is any systems ability to rapidly adapt to new subsystems which could be different sensors or motors as an example. Ease of

development is based on the existing libraries for a system, the cost of development, and the community around any given product. Systems integration is based on how well a system sill tie in with other system.

Single computers were considered but none were found that could support the amount of low level I/O needed. Also with weight and power as a requirement larger system such as a laptop were judged to be too much for what was needed; So all efforts were focused to smaller modular systems utilizing multiple micro controllers to achieve similar results. The systems considered were based around a micro controller that could support an OS, Ethernet, USB, and other communications standards to support our main software and multiple micro controllers to run the subsystems.

The main micro controller was considered from multiple ARM based SBC's such as the Raspberry Pi, rascal, and other similar system that support limited Linux kernels with both high and low level I/O. The Raspberry Pi was selected for it low price and similar performance to other systems. It had a large active community behind it and reasonable libraries for various operations on the SBC. The systems integration for this device was simple since it only needed to use Ethernet and USB to talk to its subsystems.

Selection of the low level micro controls was between the PIC32 start kit board and the mbed a NXT ARM32 bit arm with similar features. Both boards offered modularity in that they could be plugged into a back plane to interface with any needed devices. For ease of development the mbed comes ahead of the PIC in that it has better abstracted libraries, a free development environment, a fast programmer that did not require any special set up, and an online compiler that allowed development anywhere from any team member. The PIC suffers from a more difficult programming environment or one that has a very high price tag. Systems integration further favored the mbed since it used pins that could mount to bread boards or PCB's, while the PIC required a special socket that could only go on a PCB preventing any systems testing prior to board printing. Also the mbed's pins are much more durable than the pic's socket allowing for faster debugging and testing.

The software for controlling the systems level will be a mix of C++ and java, with C on the lower microcontrollers.

6.1.2) Methodology

The computer systems will be mounted in a back pack near their respective systems. The Raspberry Pi will be mounted on an edge so its Ethernet socket can be used to connect to the user application. USB cables will be used to connect the lower systems and an abstracted communication protocol will allow for swapping of the lower systems while maintaining system integrity.

Backplanes will mount the mbeds and connect them to the devices that they need to interact with. The back planes will be modular and the mbed will need to be told by the Raspberry Pi what they are connected to. The mbeds will either support multiple execution files locally and load a different one on command or have a single larger program with multiple states based on received information.

6.1.3) Results and Discussion

An API has been developed for the Ethernet side of the Raspbery Pi. The API handles the current set of needed commands for reading sensors, driving motors, reading motor sensor values, and several other commands.

Table 8: Examples of Commands and Replys through Raspberry Pi

Command	Reply
GET_RAW_EMG\n	EMG1(int):EMG2(int):EMG3(int):EMG4(int):EMG5(int)\n
SET_MOTOT_VELOCITY:MOTOR1_VEL(double):MOTOR2_VEL(double):MOTOR3_VEL(double):MOTOR4(double)\n	NO REPLY

The above are examples of commands from the API. These are sent as string via serial or Ethernet. While these commands are specific to the constructed system extra data could be added to make them expandable, but could reduce performance. These commands are parsed by the Raspberry pi and executed by sending commands to the lower boards through USB.

Mbed code has been produced to interact with the API and to drive McKibbons, Stepper, and DC motors. Mbed code has also been written to communicate with and use the EMG that was built for this project. Both sets of code work, but in the absence of finished hardware full systems integration testing is still pending. The base of all mbed code is interrupt driven with single byte instructions from the raspberry pi. In the EMG code a main timer interrupt handles reading and basic single processing while a main loop handles board communication. This code would handle squared sums of the data for the RMS signal processing, but rectifying and normalizing the single for more advanced processing would be handled by the raspberry pi. The motor driver code runs a single main loop with sensors and motor outputs being read processed and issued to satisfy arm force or velocity control

6.2) Classification

Two classifiers were evaluated and implemented in the system; a linear discrimination classifier and a neural network.

6.2.1) Design Alternatives

Many types of classifiers can be used to classify the EMG readings into motions at the elbow. The output of a classifier is not limited by the data input to the system; a classifier can output information in any number of ways. Input EMG data could be mapped to intentions like contraction or extension, a specific toque about a joint, or even variable torques about a joint. As the number of outputs becomes larger compared to the number of inputs the ability of a classifier to accurately predict the correct output is

reduced. There are a number of classifier models which have open source software support including artificial neural networks (ANN), support vector machines (SVM), and linear discriminant analysis (LDA).

Classifiers can attempt fixed speed, fixed torque, or variable torque output. Fixed torque output has four states raise, lower, hold, and rest. Variable torque has multiple states, each denoting a discrete torque, but this makes the states difficult to classify. A variable torque model would provide a more natural movement for the user, but it also has a higher likelihood of confusion. Fixed torque or fixed speed classifiers would provide less natural movement, but would likely be more reliable.

The initial selection of models was based on previous work rather than by features since machine learning algorithms are unpredictable. Three models were selected for evaluation. LDA's have been used with good results to classify fixed speed and fixed torque EMG systems. ANN's and SVM's have been used in attempts to classify variable force with reasonable results.

6.2.2) Methodology

Classifiers are a type of machine learning algorithm which require data to learn from. Consequently, a large amount of training and testing data must be collected to train the classifier. The following details how data was collected, and then how that data was used to train the classifiers.

Velocity Based Classifier

The linear classifier was based off of activity originating in the muscles involved in extension flexion, pronation, and supination. Electrodes were placed along the upper arm and lower arm on the skin above key muscles involved in each movement as shown in Figure 38. Data for five 'motions' was collected. The motions were rest, flexion, extension, pronation, and supination. Data was collected from a number of volunteers using a BioPac EMG100 system. Volunteers were asked to perform two rounds of each motion, an aide provided resistance during the four active motions, for 30 seconds per motion. This data was compiled and imported to matlab.

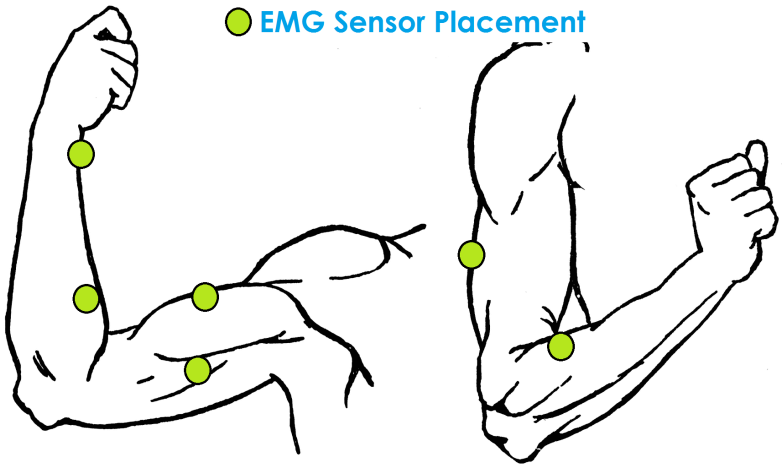


Figure 38: Sensor placement for velocity based controller

The data was then organized into testing and training sets and several signal processing algorithms were applied to the sets to prepare them for the linear classifier. Both testing and training sets consisted of data for six muscles during each of the five motions. The sets were up scaled by a factor of two so that they would be comparable to data collected from the circuits built in the course of this project. They were also windowed into 250, and the RMS was calculated for each window. The sets then consisted of rows of RMS values, a column designating a muscle, with a designator to identify which motion was performed. These data sets were then used to train the linear discriminant classifier on a person by person basis. The classifier took six inputs (muscles) and would create rules for five outputs (movements).

Once the classifiers had been trained, they were tested against the test set data. The test sets were the same format as the training data, but the classifier had not previously been exposed to the test sets. The confusion matrices were produced for the test sets, as well as the accuracy.

Torque Based Classifier

Data from a single subject was collected on the TI EMG sensors. The subject provided various torques at the elbow with sensors on the pictorial, deltoids, and trapezius muscles. The toques was a matter of force on the lower arm with forces of the following: arm at rest, arm held idle at 90 degrees flexion, 1, 3, and 4 pounds in both flexion and extension.

The data was then processed by windows where data in the windows was rectified, normalized, and then placed into a histogram before being given to the classifiers (ANN, DT, SVM, BNET). These classifiers were able to produce results of up to 96% accuracy. These results are found in Appendix 9.4. The classification was all handled by the open source project WEKA.

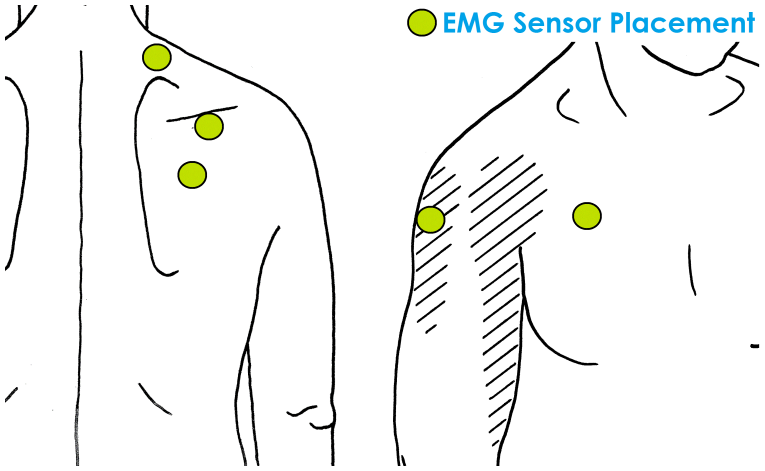


Figure 39: EMG sensro placement for torque based classifier

6.2.3) Results and Discussion

Linear and Quadratic Discrimination

The classifier inputs could also be split. One training and sampling matrix could be used for the upper arm motion, and one for the lower arm motion. The matrices for upper arm motion would include information for the movements rest, flexion, and extension from the biceps and triceps. The lower arm matrices would focus on rest, supination, and pronation, from the supinator, pronator, and pronator quadratics. Splitting the information into two sections allowed for an average accuracy of 96%. Like the combination classifier, errors would be small however accuracy would increase.

One major pro for this system is that a lot of crosstalk noise between muscles not needed for certain motions would be eliminated before the classifier creating more accuracy. However, being able to move the arm in two motions may require a more sophisticated control system or more power to the arm.

Both of the versions of a linear discrimination classifier would be suitable for this device. The one change that would need to be made would to remove it from MATLAB completely. MATLAB need licenses to run and a computer to run on both of which could cause potential issues for this device. They could increase the cost and could require more parts, which would interfere with weight and size constraints.

Table 9: Average results for linear and quadratic discrimination

Linear Discrimination							Quadratic Discrimination						
74.2	2.8	1.5	0.3	0.2	Rest	movement	75.7	4.5	0.8	0.2	0.7	Rest	movement
4.2	66.3	2.3	4.5	3.0	Flexion		1.0	70.0	9.0	0.7	0.7	Flexion	
1.0	3.5	70.3	6.2	1.5	Extension		0.0	2.8	66.5	9.5	0.0	Extension	
1.7	2.2	2.7	66.8	8.8	Supination		0.0	7.0	5.8	60.0	5.7	Supination	
8.0	2.0	0.5	3.3	62.2	Pronation		0.3	1.0	0.7	4.8	72.7	Pronation	
R	F	E	S	P			R	F	E	S	P		
classified as							classified as						
accuracy: 85.0%							accuracy: 86.2%						

The WEKA Classifiers for torque produced accuracies of 96%, which can be found in Appendix 9.4. The classified out puts were three forces in flexion and extension, rest, and idle. These results are significant since WEKA is open source and can be more easily deployed than MATLAB classifiers.

6.3) User Interface

The user interface was created so a user could easily see what was currently happening on the arm and in for the brace. Along with an easy to see explanation the user could also use the interface to start or stop the system.

6.3.1) Design Alternatives

Choosing the proper Graphical User Interface (or GUI) was one of the many challenges in this project. The GUI had to be easy for both patients and doctors to use. The data in the presented charts also needed to be relevant information for evaluating the

effectiveness of the arm. The first part of designing the GUIs was to develop several design options. The first design is shown in Figure 40:

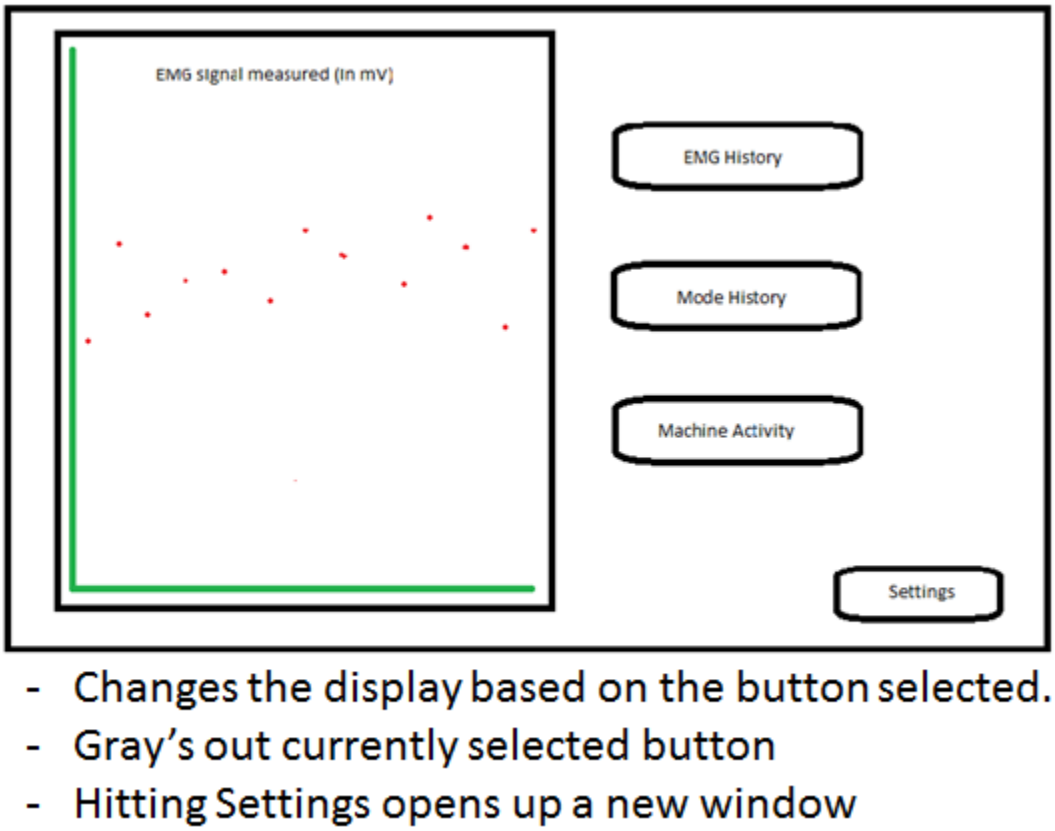


Figure 40: First interface concept

This design was the first one we came up with. In this design, the user selected what type of data they wanted to view, and the data would be shown on the left hand side of the screen. Selecting the settings button opened a popup window that would allow the user to adjust settings such as the current speed of the arm, or what mode the arm was operating. This design contained a single chart on one screen, which made it easy to read. It also allowed for a password to be attached to the settings button, so a patient could not set the speed too high. The biggest issues with this design were that it was difficult to implement changing the graph on the one screen and that if the user wasn't paying close attention, they could select the wrong data.

The second design is shown in Figure 41.

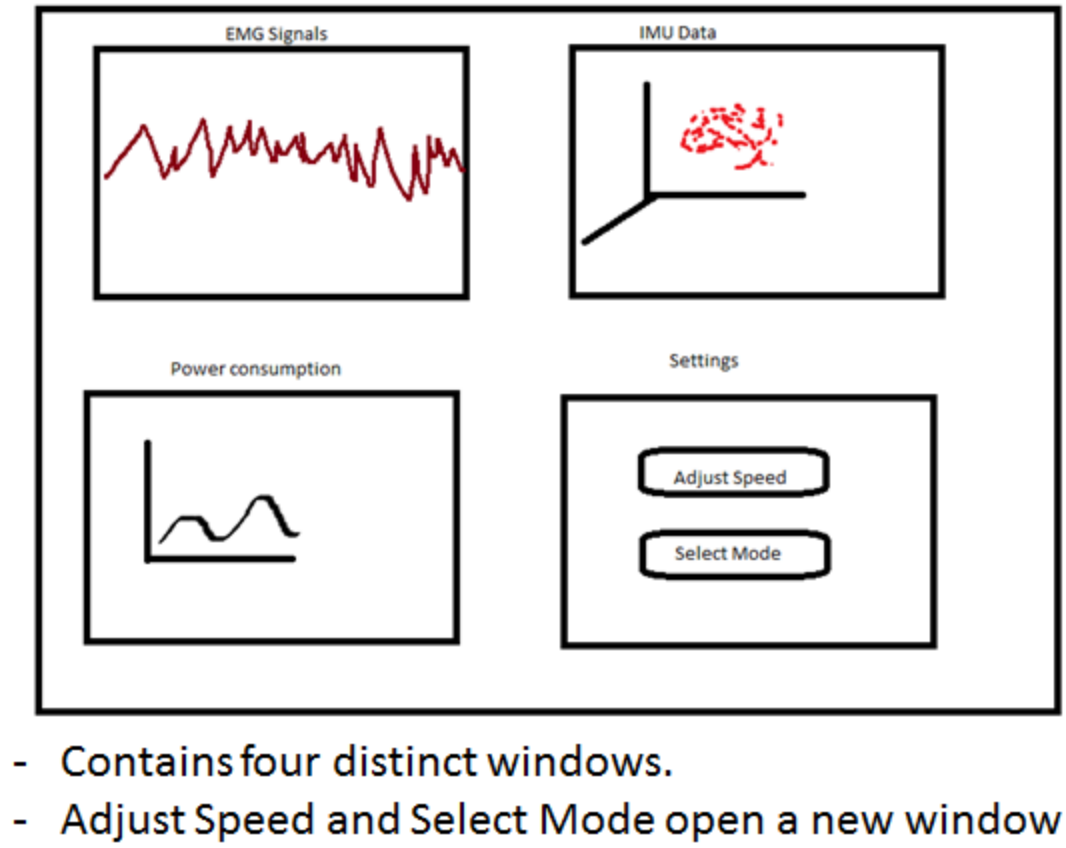


Figure 41: Second interface design concept

This design featured multiple charts on the same window. Similarly to the first chart, the user could select to adjust the speed or the mode with the buttons. This would open a popup window, which could be easily password protected. By containing all the charts on one screen, everything would be presented at once. This allowed the data to be accessed easily. The largest disadvantage with this design was the size of the charts. Since there were many charts on one screen, they were very small, and could be hard to read on smaller screens.

The last design featured a window with buttons to pop open a new window with the selected chart in it. The speed control was a slider bar that was found on the home screen. Figure 42 below shows this design.

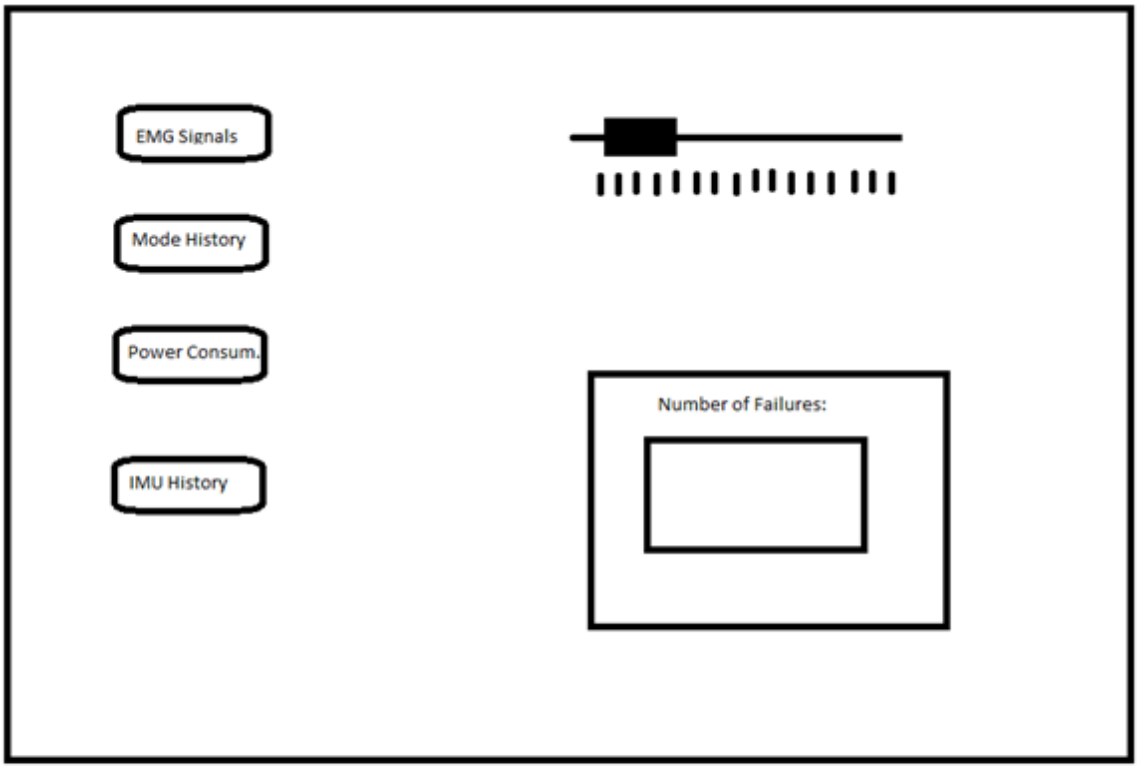


Figure 42: Third interface design concept

This design allowed the user to select the data that they wanted to view, and opened the chart in a new window. The advantage to this was that the chart would be in its own window, which would allow the chart to be larger than the other two designs. Unfortunately, this chart is reliant on popup windows, which could clutter the screen. This device also changed the speed of the arm with a slider bar, which would be hard to password protect.

6.3.2) Methodology

Java was selected to design the user interface, as it had built in tools that made designing windows easier. One of the most important tools that were used to develop the window was the Google WindowBuilder. When designing the window, the interface

needed to be able to read data from all six electrodes and display them on the screen. The user also needed to be able to see what forces and positions were being read in from the sensors attached to the motors.

When developing the interface, the decision was made to separate the signals read from the electrodes from that of the motors. This meant that separate windows were needed to show the EMG data and to show the motor data. In order to select what data would be viewed, a home screen would also be needed. From the home screen, the user had the ability to raise and lower the arm in addition to reading the signals. This would give the clinician direct control over the orthotic for purposes of testing, training, and evaluation.

7) Final Design

From experimentation with the different actuators, sensors, control schemes discussed, a final prototype was constructed. This design included a soft brace with rigid points, EMG sensors, cable driven actuators and distributed system architecture.

7.1) System Level Design

The overall system for the final orthotic prototype split the product into four sections, a soft brace, the motors driving it, the sensors reading user input, and the software and systems driving it all. Figure 43 below shows the final design and placement on a user's body.

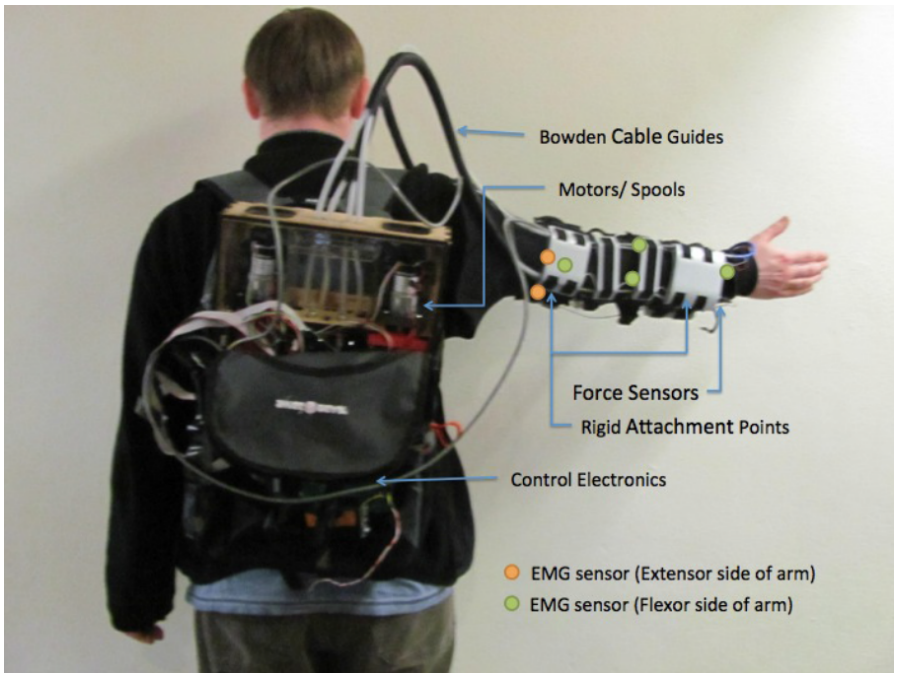


Figure 43: The final design and placement of the orthotic on a user

7.2) Brace Construction

The brace was constructed of a series of rigid actuation points attached to a soft sleeve. A user could easily put their arm into the sleeve and tighten the straps around the actuation points to prevent slippage. To aid actuation around the flexed elbow, a pair of cable guides was added, as seen in Figure 44.

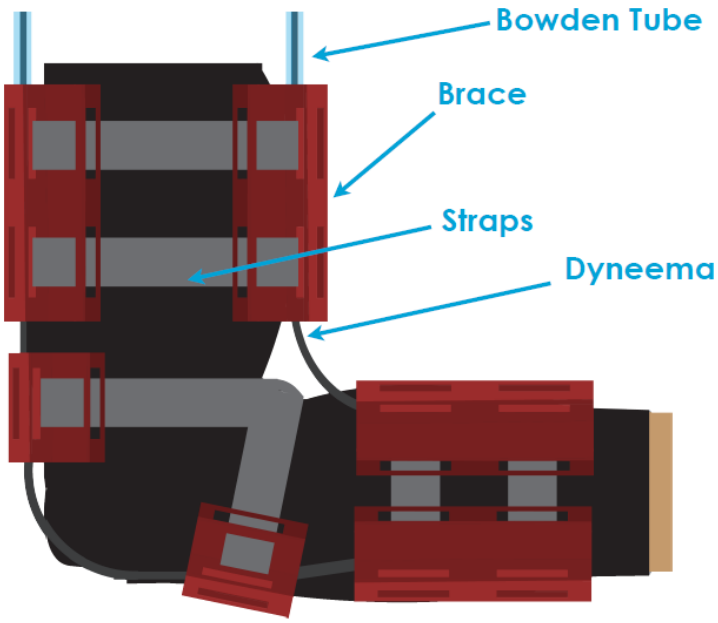


Figure 44: Schematic for final brace design

The actuator mounting points on this brace were neoprene shells, which contained two types of sensors; force gauges and EMG electrodes. The mounting points were attached to the brace using buckled straps, which allowed resizing to fit any user's arm. The cabling that ran through the brace was Dyneema® fiber cabling.

The force sensor was a spring mount potentiometer assembly. When force was applied to the spring, it changed the potentiometer reading, allowing for force feedback control of the users arm.

Figure 45: Fully realized brace system

7.3) Actuation

From our actuator testing, we settled on motor driven Bowden cable actuation. From a motor mounting box in the backpack, shown in Figure 46, two antagonistic pairs of cables ran to the mounting points on the brace. Actuating the cables allows for flexion and extension of the elbow joint.

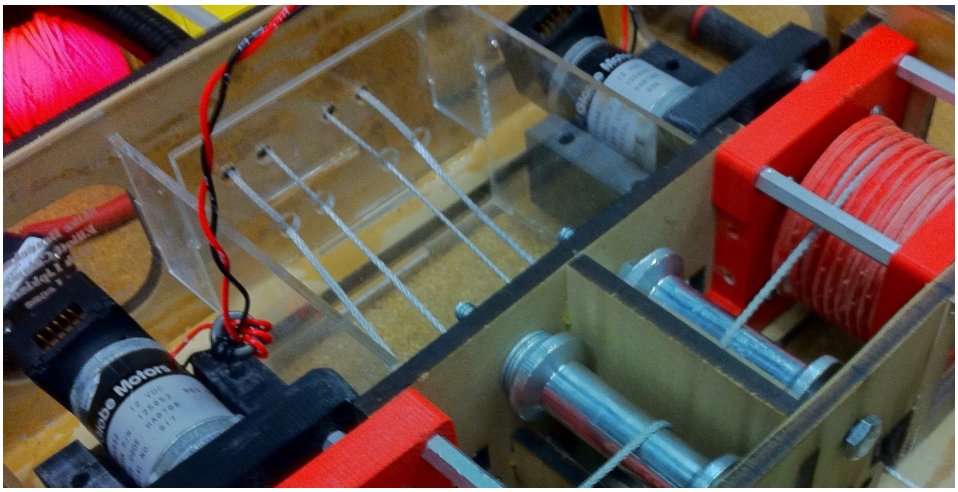


Figure 46: Close up of cable spool box

The motors selected were Lin Engineering "Super Torque" stepper motors, and were selected because they were able to provide over 8 oz-inch of torque. These motors were controlled from a dedicated microprocessor and motor control boards, which allowed the force sensors in the arm to apply feedback control and keep the patient safe from excessive force as well as hyperextension and hyperflexion. The PCB that mounted all the circuitry to drive the stepper motors is shown in Figure 47. The final board that was built is in Figure 48.

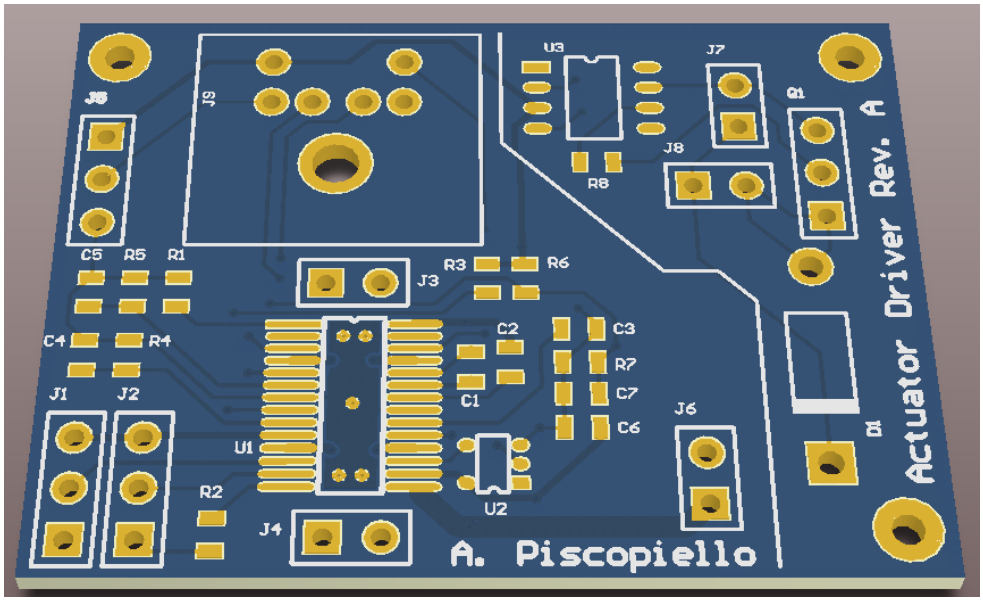


Figure 47: Actuator driver boards. These were capable of driving both McKibbens and stepper motors from the same inputs.

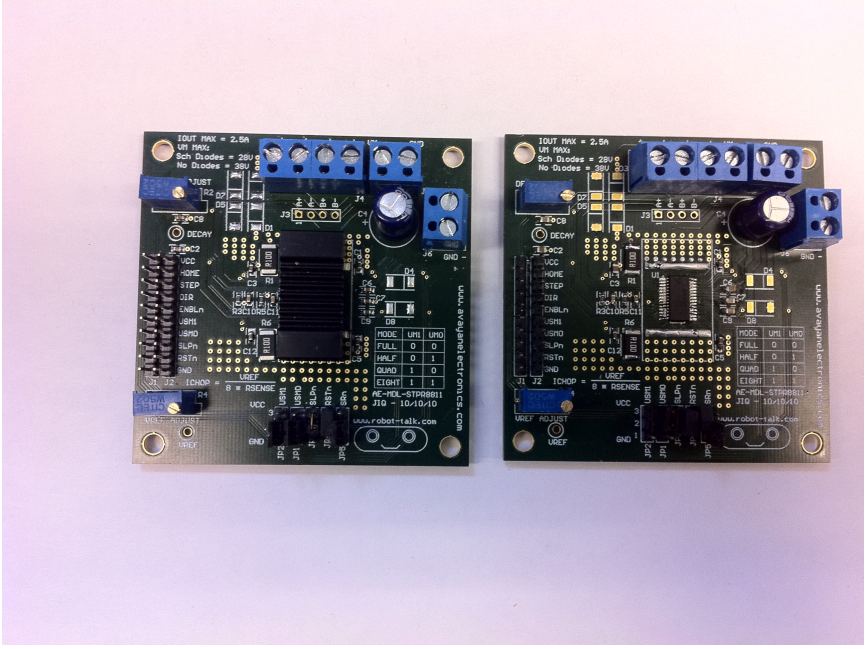


Figure 48: Stepper motor drivers that were built into the final design. These are not capable of driving the McKibbens since Bowden cables were used in the final design

7.4) Sensors

Of the sensors we tested, only one set made it onto the final design. The Inertial Measurement Unit (IMU) was left out simply due to cost and availability, as discussed above. The EMG acquisition system was implemented on a printed circuit board, and is shown in Figure 49. Three of these boards mounted to a backplane which contained all of the circuitry to regulate power to the system, as well as the microcontroller which was responsible for collecting the digitized signals from the ADC's on the EMG acquisition boards. The backplane is shown in Figure 50.

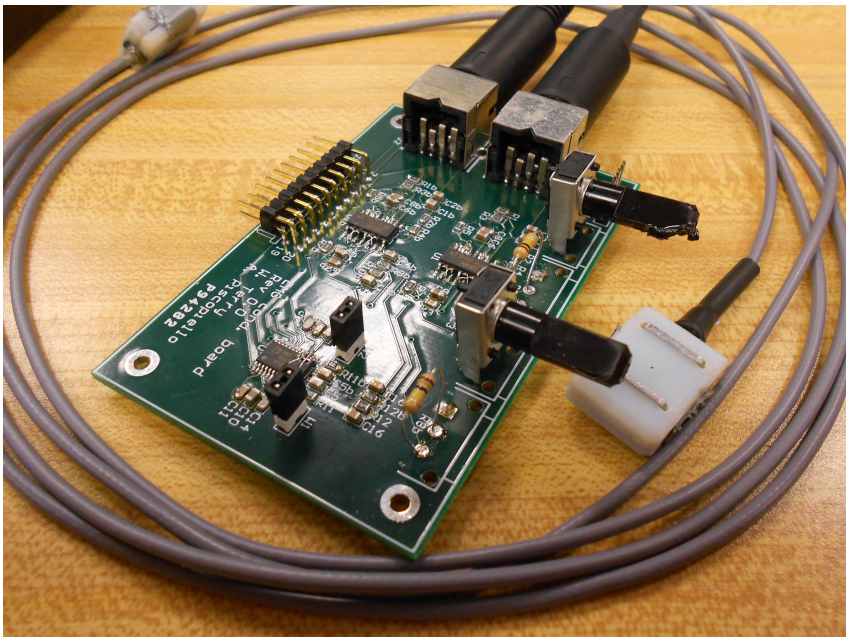


Figure 49: Fully realized, two channel, modular EMG sensor board

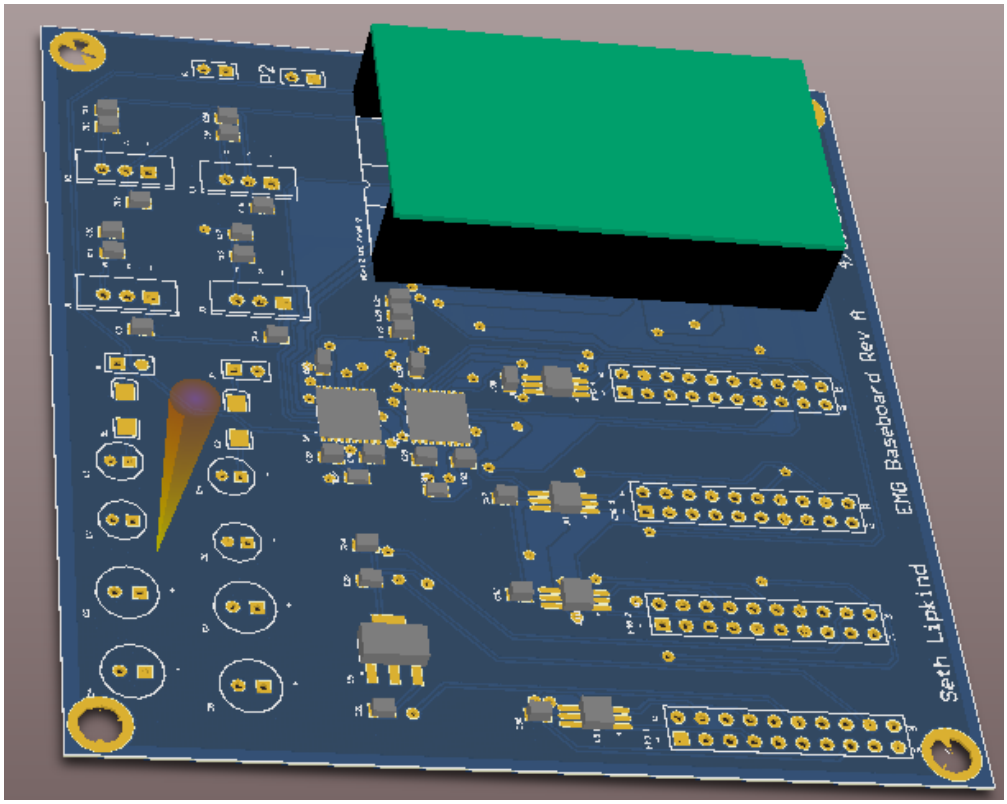


Figure 50: EMG backplane capable of mounting up to eight EMG channels

The six positions of electromyographic (EMG) sensors, shown in Figure 51, were able to give accurate predictions of intent, with the linear classifier model giving outputs between 80% and 96% correct, depending on the electrode separation used.

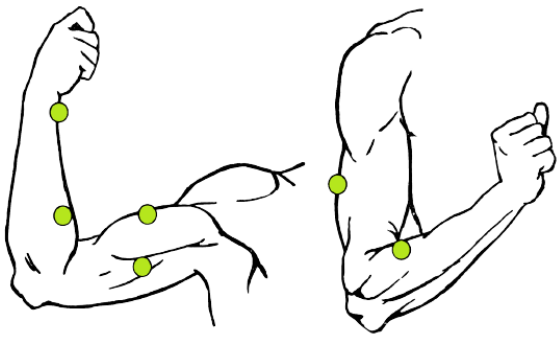


Figure 51: EMG mounting points

7.5) Software & Systems

The software in the final prototype was fit to the overall design. The overall design can be seen as a central hub of a Raspberry Pi microcontroller with spokes out to handle sensor input, motor control, and user interface, as pictured in Figure 52.

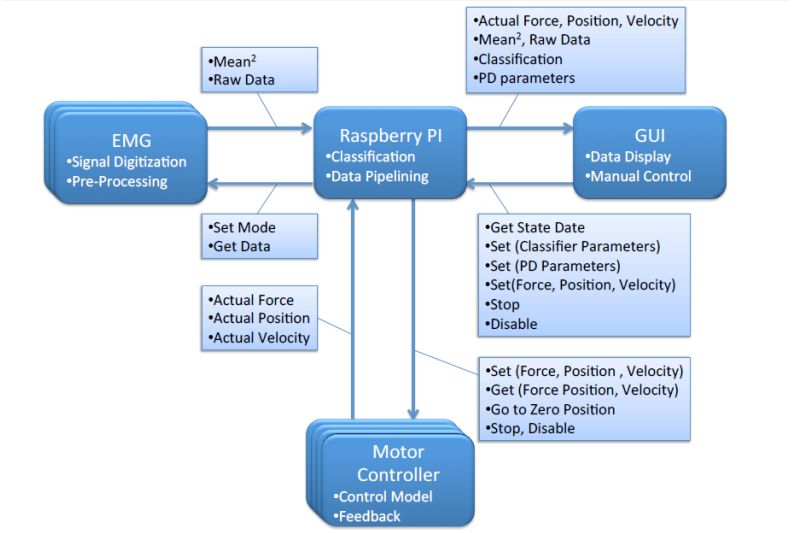


Figure 52: The system architecture

The sensor side collected and pre-processed input signals coming from the EMG sensors, handling noise removal and simple calculations such as root mean square. From there, the signals were sent, in either processed or unprocessed form, to the mbed and then to the main Raspberry Pi, where the signals were run through a classifier. From there, the signals were sent to the computer to be shown in the user interface.

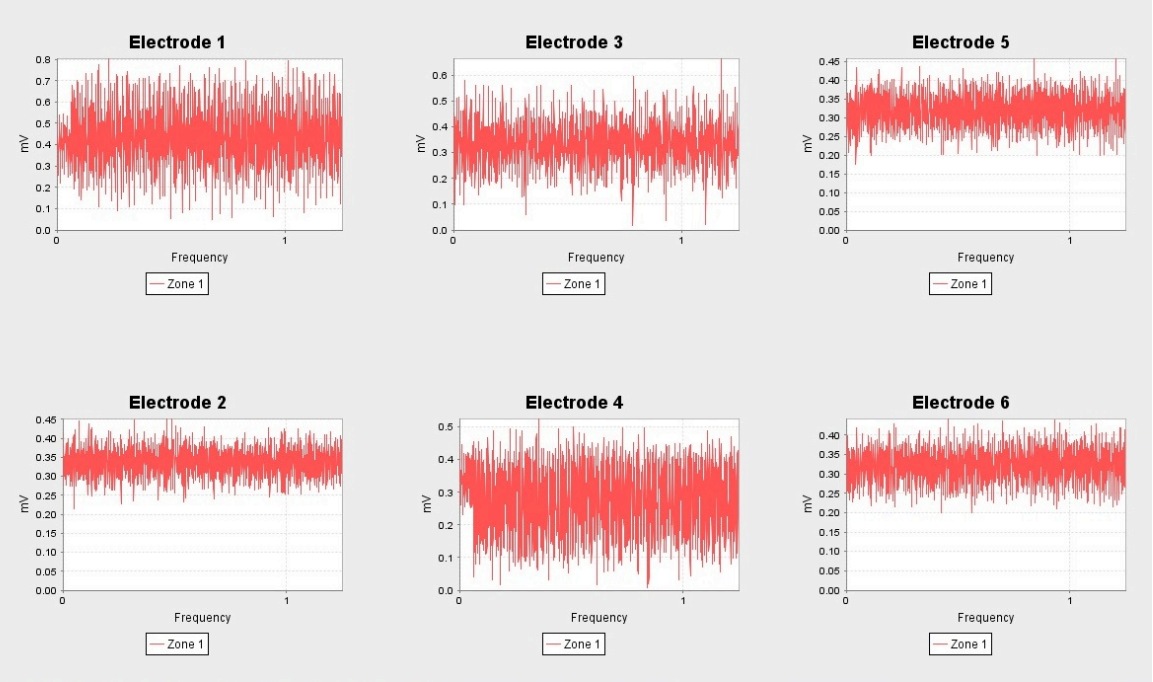


Figure 53: Example of EMG data on user interface.

8) Discussion & Conclusion

The final system was not fully functional, but contained many portions that worked. The electrodes and processing boards that the team designed were able to obtain the desired data. They received the noisy signal from the user's muscles, and filtered out the noise. The boards, however, had many issues before working correctly. On nearly all of the boards, the rough gain adjustment potentiometers burnt out, and needed to be replaced by resistors. The pots burning out caused another issue however, where the board was producing a square wave that alternated between the rails instead of the desired signal. Wires from the boards also became unsoldered, and caused the signal to be distorted.

Another aspect of the system that was the Bowden cables being driven with DC motors. The motors did not have enough torque to fully drive the arm, however, it was able to raise and lower the arm. This was done by the two antagonistic pairs of cables on the upper and lower sides of the arm. The DC motors drove the two cables attached to points of the brace near the bicep, and raised the arm. The motors then drove two cables attached to points on the brace near the triceps, and lowered the arm.

Unfortunately the system did not integrate together. Although the motors were able to move, and the signals were able to be correctly read from the person, the motors did not move based on the signals. The motors ended up moving based on a switch. The board that ended up driving the motors was the same board used in the RBE 3001 class at WPI. This was due to the lack of time to test the boards that the team developed to drive the stepper motors.

Things that should be changed in the next iteration of this project would be to make sure parts were ordered far earlier in the project. This would allow us to have adequate time to test and debug the any issues that came up while experimenting with the different parts. I would also encourage the entire team to communicate effectively on portions of the project they are working on. This would allow the entire team to know if

a new direction was taken, so there was no wasted time working on portions of the design that was not going to be used on the final product.

The next steps for our project would be system integration. Currently, parts of the system are working, but the system in its entirety has not been tested. The first step to test the system would be to attach the electrodes to the processing boards and then the Mbed. Once this works, the data would need to be sent to the Raspberry Pi. Finally after that is working, the data would need to be sent into the GUIs. The motor side would be a fairly similar process, where the data from the encoders and magnet potentiometers would be sent to the Mbed and up the rest of the chain.

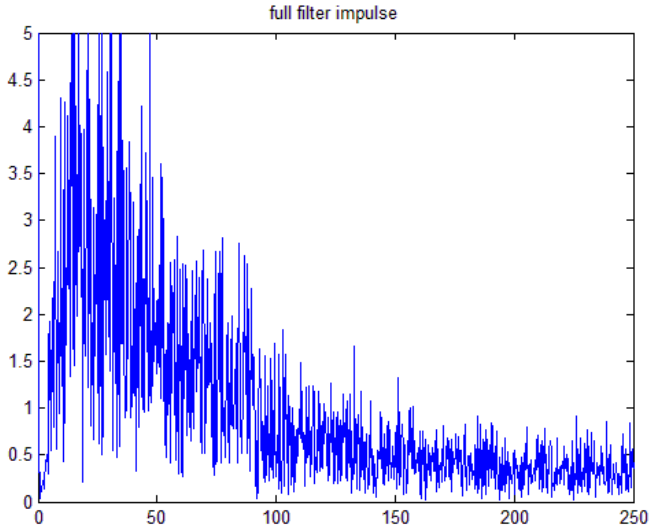
In order for the system to be integrated, a few parts would need to be fixed. First, the interface needs to have the serial commands added to interface with the Raspberry Pi. This will allow for the streaming of data between the two components. The next step would also be to become familiar with JFreechart through the demos available online. Currently the only way to get documentation is to pay for it, but there are many examples to look through online.

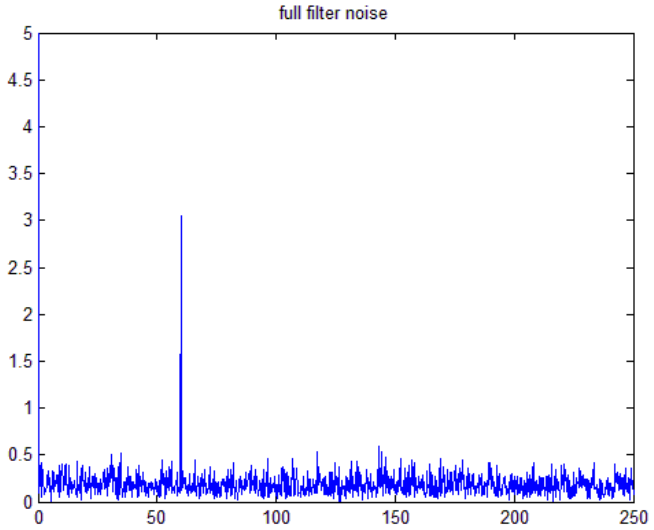
One main suggestion for the EMGs would be to change the ADC used on the filtering boards. The AD7367 used this year was both expensive and not equivalent to some standard thinking ways. For example both chip select and convert start go low when active, and are technically named not chip select and not convert start. From experimenting, it was found that busy took longer than the maximum time specified on the data sheet. If this part is changed, the requirements are dual, or two channeled with a 14-Bit resolution.

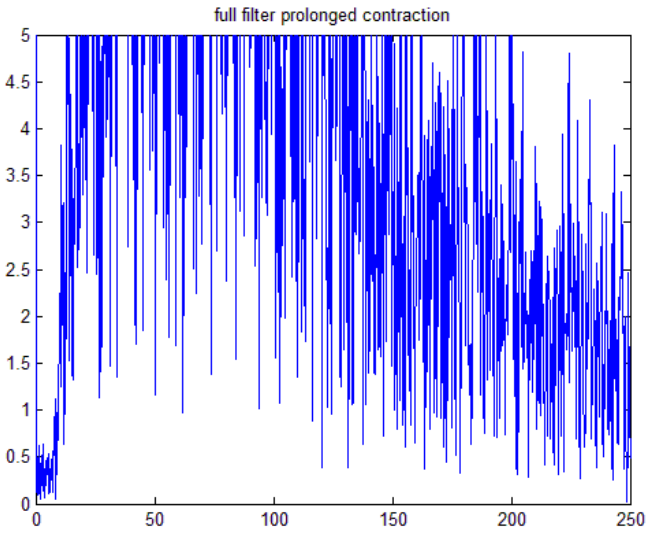
For motors, the biggest step forward will be to get the boards to work with the stepper motors. The team both designed boards and ordered boards to try to work with the stepper motor. Unfortunately, due to timing, the last motor board the team designed was not completely finished. However, other motor boards that should properly interface with the stepper motors were ordered, but were not able to be debugged. For view the code for any of these parts, please see Appendix 9.4.

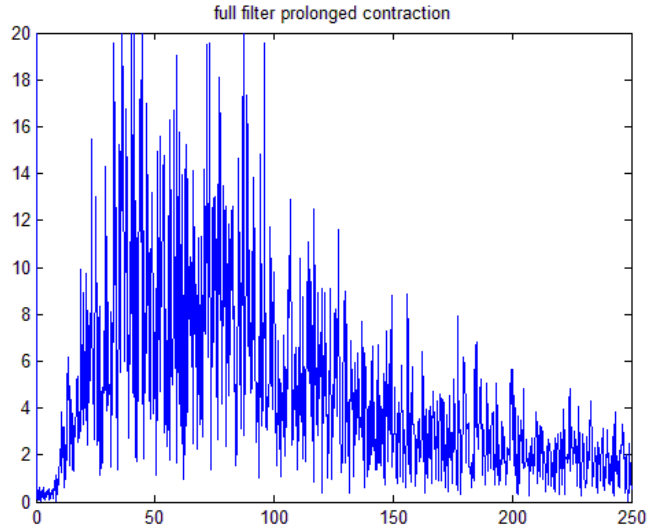
9) Appendices

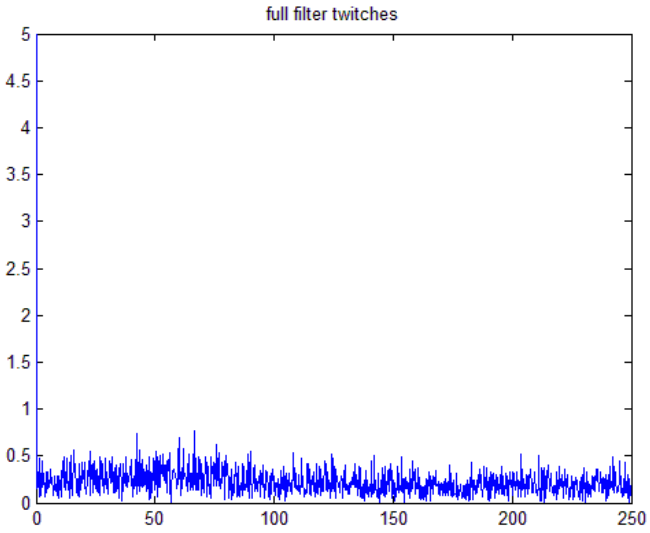
9.1) Sensor Data Collection

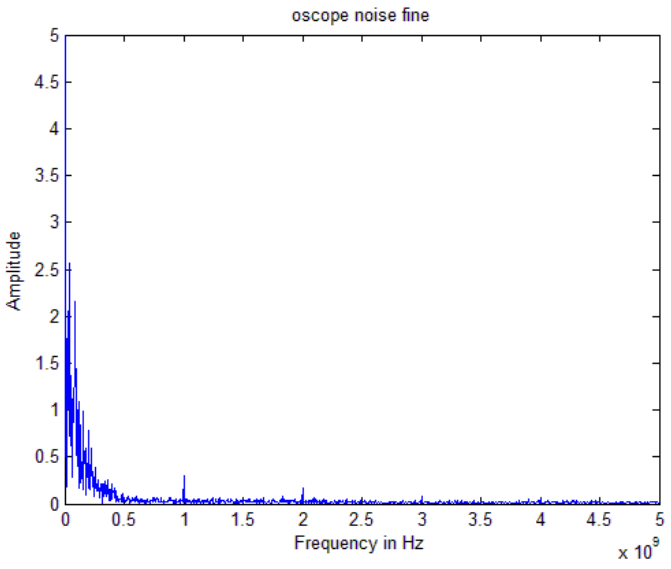


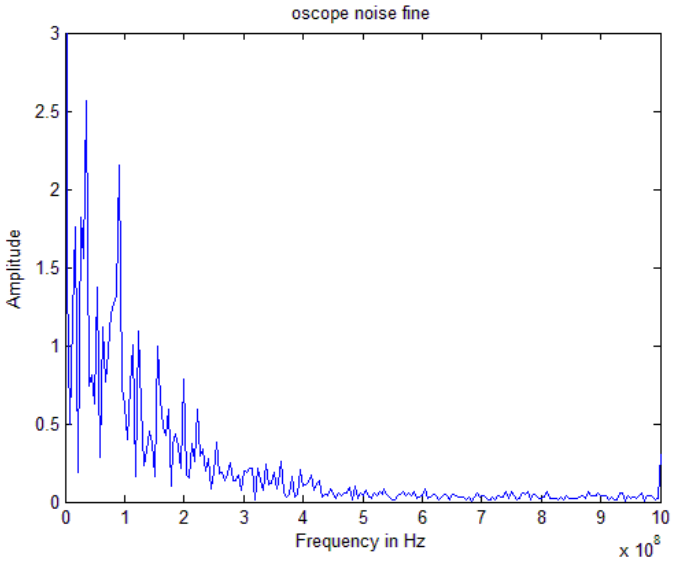


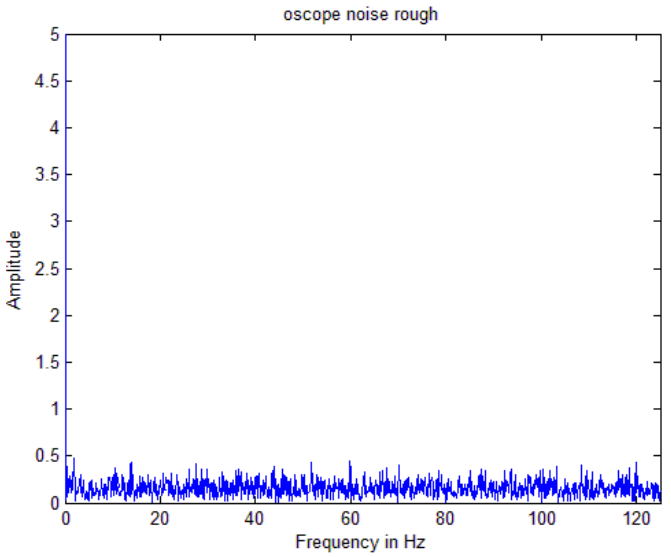

full filter prolonged contraction
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
0
50
100
150
200
250

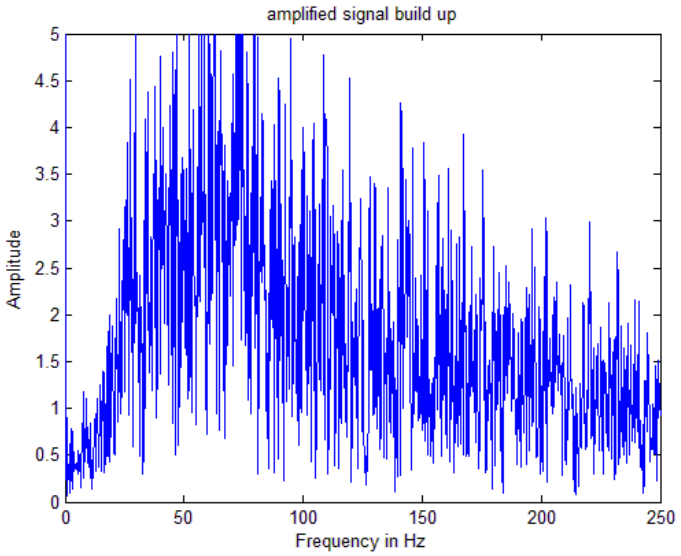

full filter prolonged contraction
0
2
4
6
8
10
12
14
16
18
20
0
50
100
150
200
250

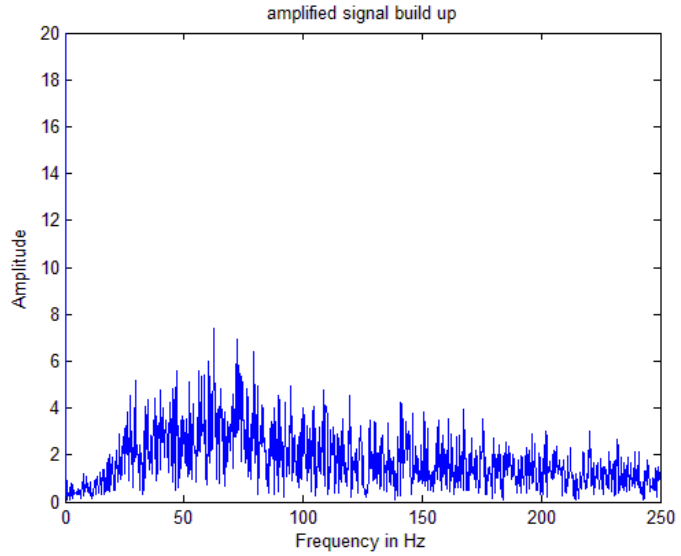

full filter twitches
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
0
50
100
150
200
250

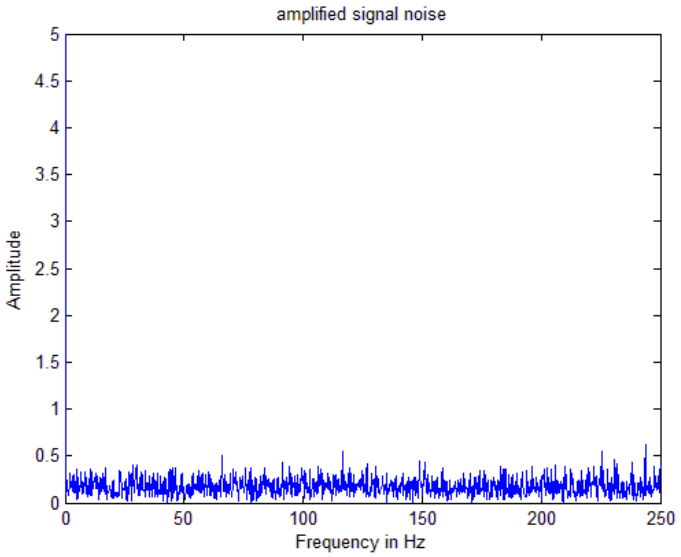

oscope noise fine
Amplitude
Frequency in Hz
x 10^9

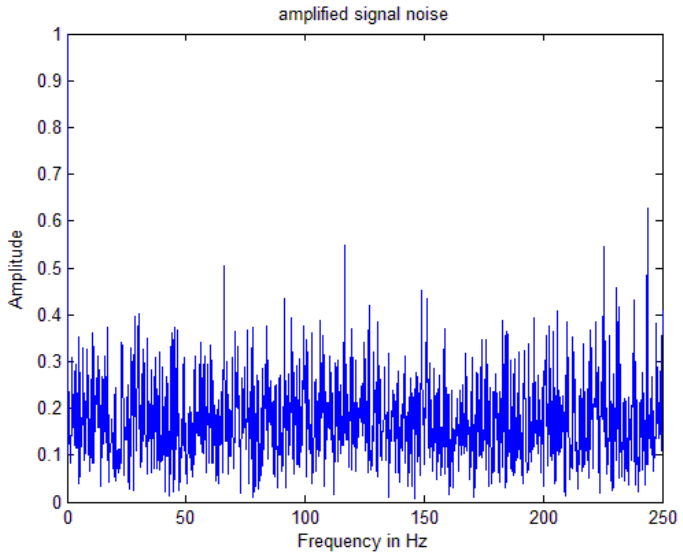

oscope noise fine
Amplitude
Frequency in Hz
x 10^8

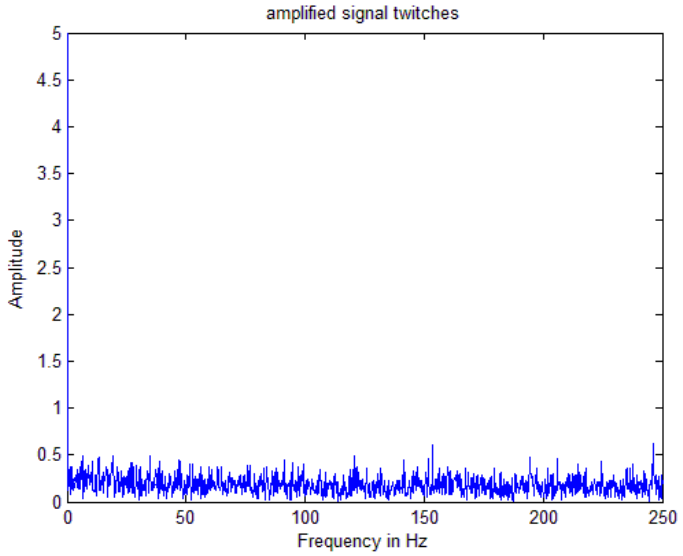

oscope noise rough
Amplitude
Frequency in Hz

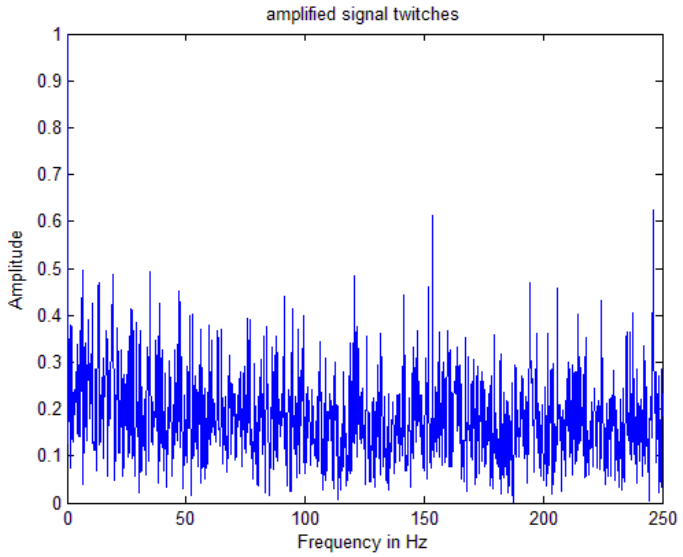

amplified signal build up
Amplitude
Frequency in Hz
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
0
50
100
150
200
250


amplified signal build up
Amplitude
Frequency in Hz
0
2
4
6
8
10
12
14
16
18
20
0
50
100
150
200
250


amplified signal noise
Amplitude
Frequency in Hz
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
0
50
100
150
200
250


amplified signal noise
Amplitude
Frequency in Hz
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0
50
100
150
200
250


amplified signal twitches
Amplitude
Frequency in Hz
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
0
50
100
150
200
250


amplified signal twitches
Amplitude
Frequency in Hz
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0
50
100
150
200
250

9.2) Actuator Data Collection

Data from 4 inch muscle under varying loads

Pressure	Zero g	100 g	500g	1000g	Compliance
0	3.41	3.422	3.451	3.456	0.009791
0.5	3.402	3.417	3.429	3.456	0.011231
1	3.393	3.414	3.421	3.455	0.011807
1.5	3.268	3.271	3.278	3.271	0.000000
2	3.127	3.064	3.098	3.098	0.009791
2.5	3.046	2.974	2.982	3.042	
3	2.979	2.93	2.945	2.944	0.004032
3.5	2.927	2.878	2.884	2.89	0.003456
4	2.86	2.839	2.884	2.837	-0.000576
4.5	2.811	2.801	2.797	2.787	-0.004032
5	2.77	2.745	2.748	2.765	

Data from 2, 4 and 6 inch muscle under varying pressures

Pressure	4 Inch	Contraction	6 inch	Contraction	2 Inch	Contraciton
0	3.451	0.00%	5.611	0.00%	2.193	0.00%
0.5	3.429	0.64%	5.520677	1.61%	2.181067428	0.54%
1	3.421	0.87%	5.558741	0.93%	2.186728312	0.29%
1.5	3.278	5.01%	5.12664	8.63%	2.139166596	2.45%
2	3.098	10.23%	4.613087	17.78%	2.011536466	8.27%
2.5	2.982	13.59%	4.295019	23.45%	1.95861927	10.69%
3	2.945	14.66%	4.193567	25.26%	1.958550854	10.69%
3.5	2.884	16.43%	4.116307	26.64%	1.915465088	12.66%
4	2.884	16.43%	4.056307	27.71%	1.895465088	13.57%
4.5	2.797	18.95%	3.887756	30.71%	1.848277191	15.72%
5	2.748	20.37%	3.6934	34.18%	1.8217001	16.93%

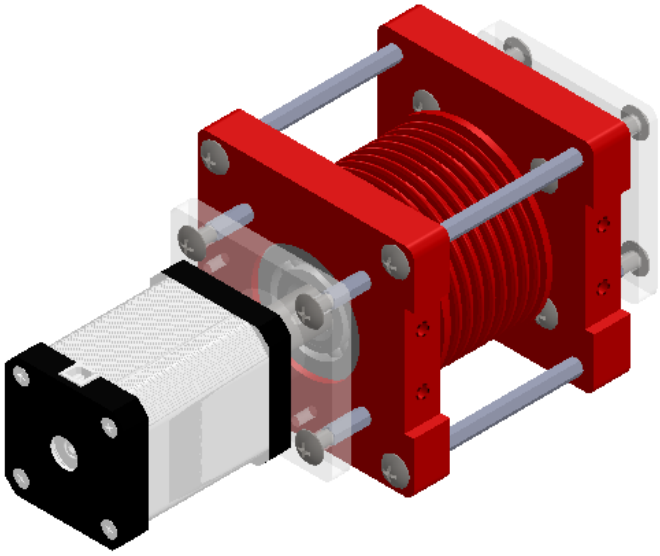
Data From Series Parallel Test Muscles 1 is in series with 2, 3 is in series with 4. The two series pairs are connected to each other in Parallel.

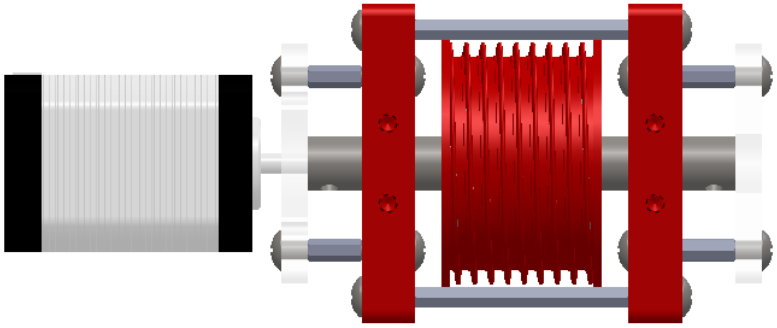
Pressure Muscle 1	Pressure Muscle 2	Pressure Muscle 3	Pressure Muscle 4	Length Muscle 1	Length Muscle 2	Length Muscle 3	Length Muscle 4	Length Total
0	2	4	4	3.215	2.967	2.341	2.852	7.6875
2	0	4	4	2.398	3.361	2.788	2.873	7.71
2	4	4	4	2.946	2.784	2.863	2.869	7.731
4	2	4	4	2.738	3.022	2.846	2.883	7.7445

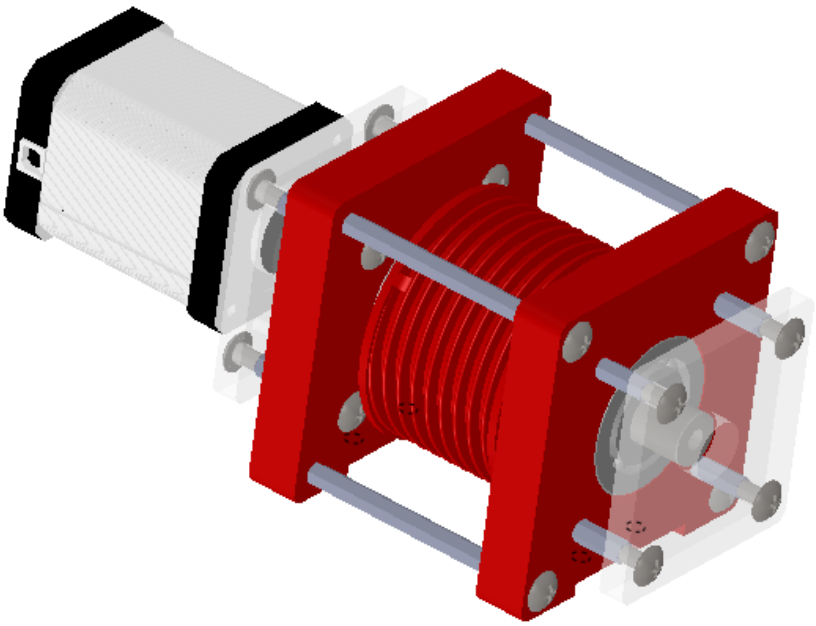
4	4	4	4	2.884	2.884	2.884	2.884	7.768
2	2	4	4	2.751	3.029	2.874	2.91	7.782
4	4	4	2	2.926	2.887	2.754	3.055	7.811
0	0	4	4	3.058	2.759	2.843	2.977	7.8185
0	4	4	4	3.016	2.8	2.886	2.935	7.8185
4	4	2	4	3.051	2.884	2.864	3.068	7.9335
4	0	4	2	2.57	3.3992	2.859	3.104	7.9661
0	0	4	2	2.955	3.304	2.552	3.131	7.971
4	0	2	4	2.66	3.3952	3.017	2.876	7.9741
2	4	2	4	3.096	2.884	3.098	2.884	7.981
2	4	4	2	3.096	2.884	2.885	3.098	7.9815
4	2	4	2	2.882	3.098	2.885	3.098	7.9815
4	2	2	4	2.884	3.098	3.098	2.884	7.982
2	2	4	2	2.967	3.018	2.873	3.111	7.9845
2	2	2	4	2.953	3.083	3.053	2.896	7.9925
0	0	2	4	2.962	3.133	2.998	2.9	7.9965
0	4	4	2	3.3772	2.713	2.816	3.181	8.0436
0	4	2	4	3.2922	2.853	3.006	2.957	8.0541
4	2	2	2	2.992	3.088	3.062	3.015	8.0785
4	4	2	2	3.218	2.874	3.066	3.022	8.09
2	0	2	4	2.813	3.291	3.234	2.87	8.104
4	0	4	4	2.882	3.222	2.975	3.129	8.104
4	4	4	0	2.912	3.222	2.87	3.261	8.1325
2	4	2	2	3.283	2.878	3.08	3.077	8.159
2	2	2	2	3.096	3.098	3.098	3.098	8.195
0	0	2	2	2.792	3.402	3.098	3.101	8.1965
2	0	2	2	2.931	3.271	3.093	3.101	8.198
2	0	4	2	2.762	3.436	3.099	3.1	8.1985
4	2	0	4	3.09	3.125	3.3572	2.856	8.2141
4	4	0	4	3.224	3.001	3.421	2.802	8.224
4	0	2	2	2.826	3.404	3.122	3.113	8.2325
2	2	0	4	3.121	3.118	3.434	2.803	8.238
0	2	4	2	3.391	2.848	2.936	3.305	8.24
4	2	4	0	2.939	3.315	2.856	3.3962	8.2531
0	4	4	0	3.421	2.835	2.839	3.422	8.2585
0	4	0	4	3.422	2.839	3.422	2.839	8.261
4	0	0	4	2.839	3.422	3.422	2.839	8.261
4	0	4	0	2.839	3.422	2.839	3.422	8.261
2	4	0	4	3.398	2.875	3.4202	2.849	8.2711
2	4	4	0	3.318	2.97	2.861	3.4242	8.2866
0	4	2	2	3.416	2.884	3.183	3.119	8.301

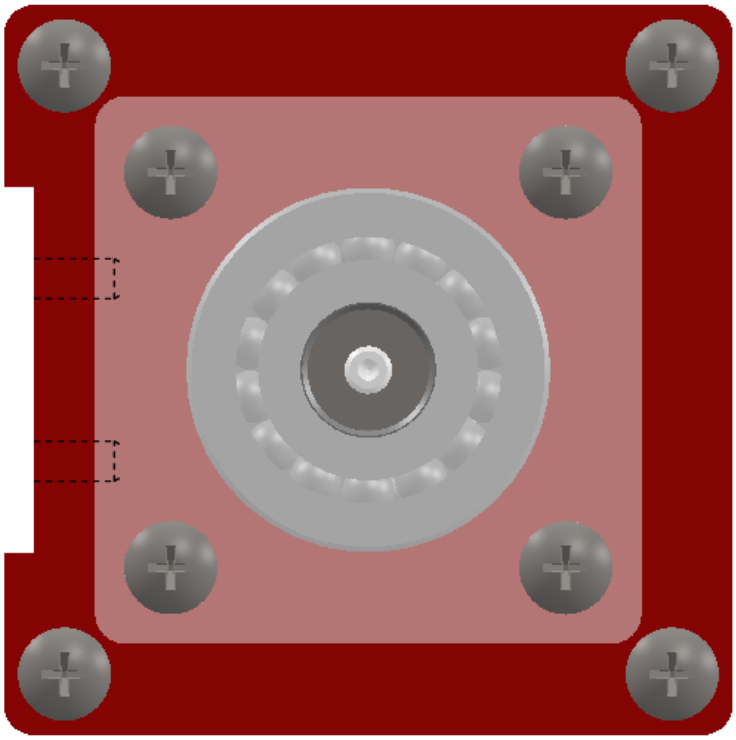
2	2	4	0	3.19	3.114	2.854	3.445	8.3015
0	2	4	0	3.375	2.961	2.862	3.453	8.3255
4	2	2	0	2.903	3.428	3.071	3.257	8.3295
0	2	0	4	3.26	3.074	3.448	2.88	8.331
2	0	0	4	3.046	3.348	3.408	2.903	8.3525
2	0	4	0	2.957	3.421	2.856	3.486	8.36
0	2	2	4	3.438	2.927	3.432	2.934	8.3655
4	4	2	0	3.006	3.366	3.039	3.333	8.372
4	4	0	2	3.334	3.049	3.355	3.023	8.3805
0	2	2	2	3.416	2.977	3.095	3.293	8.3905
4	0	0	2	2.884	3.517	3.359	3.037	8.3985
2	2	2	0	3.281	3.123	3.015	3.386	8.4025
0	4	0	2	3.535	2.893	3.363	3.063	8.427
2	4	2	0	3.151	3.289	3.027	3.411	8.439
2	2	0	2	3.142	3.333	3.39	3.083	8.474
0	4	2	0	3.598	2.887	3.051	3.43	8.483
4	0	2	0	2.93	3.561	3.08	3.408	8.4895
4	2	0	2	3.386	3.106	3.44	3.052	8.492
2	4	0	2	3.561	2.962	3.439	3.082	8.522
4	0	0	0	2.999	3.541	3.288	3.252	8.54
0	2	0	2	3.45	3.097	3.451	3.098	8.548
0	2	2	0	3.449	3.097	3.1	3.451	8.5485
2	0	2	0	3.098	3.449	3.098	3.452	8.5485
2	0	0	2	3.098	3.451	3.455	3.099	8.5515
0	0	0	2	3.387	3.237	3.463	3.125	8.606
0	0	2	0	3.288	3.378	3.046	3.519	8.6155
0	0	0	4	3.346	3.377	3.817	2.907	8.7235
0	0	4	0	3.431	3.377	3.288	3.524	8.81
4	4	0	0	3.63	3.194	3.39	3.434	8.824
2	2	0	0	3.697	3.154	3.411	3.436	8.849
0	4	0	0	3.477	3.378	3.429	3.424	8.854
4	2	0	0	3.539	3.318	3.43	3.426	8.8565
2	4	0	0	3.521	3.341	3.437	3.424	8.8615
2	0	0	0	3.384	3.481	3.429	3.436	8.865
0	2	0	0	3.777	3.099	3.437	3.434	8.8735
0	0	0	0	3.447	3.45	3.451	3.451	8.8995

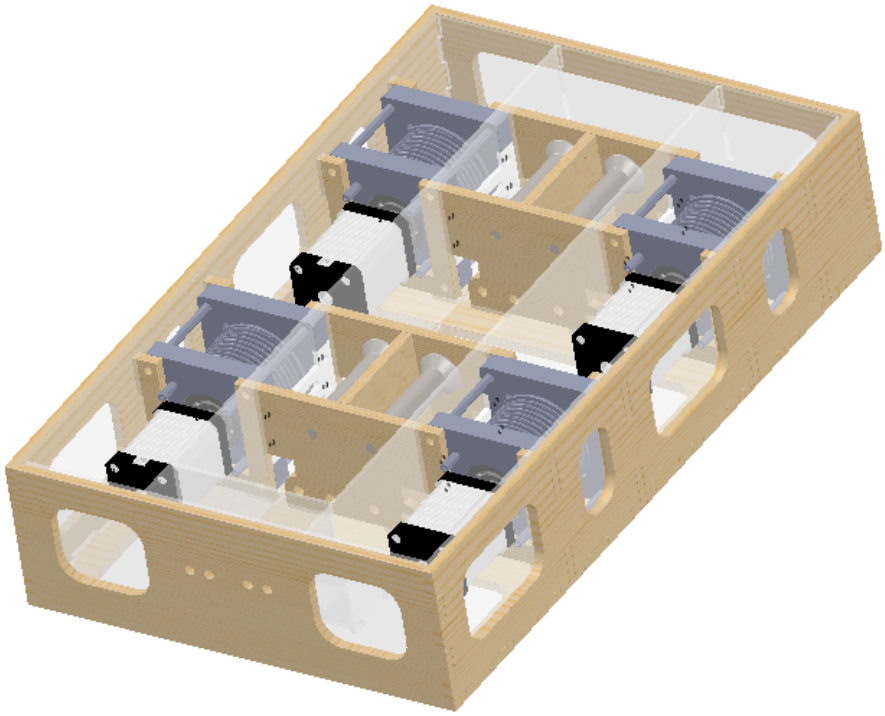
9.3) SolidWorks Models

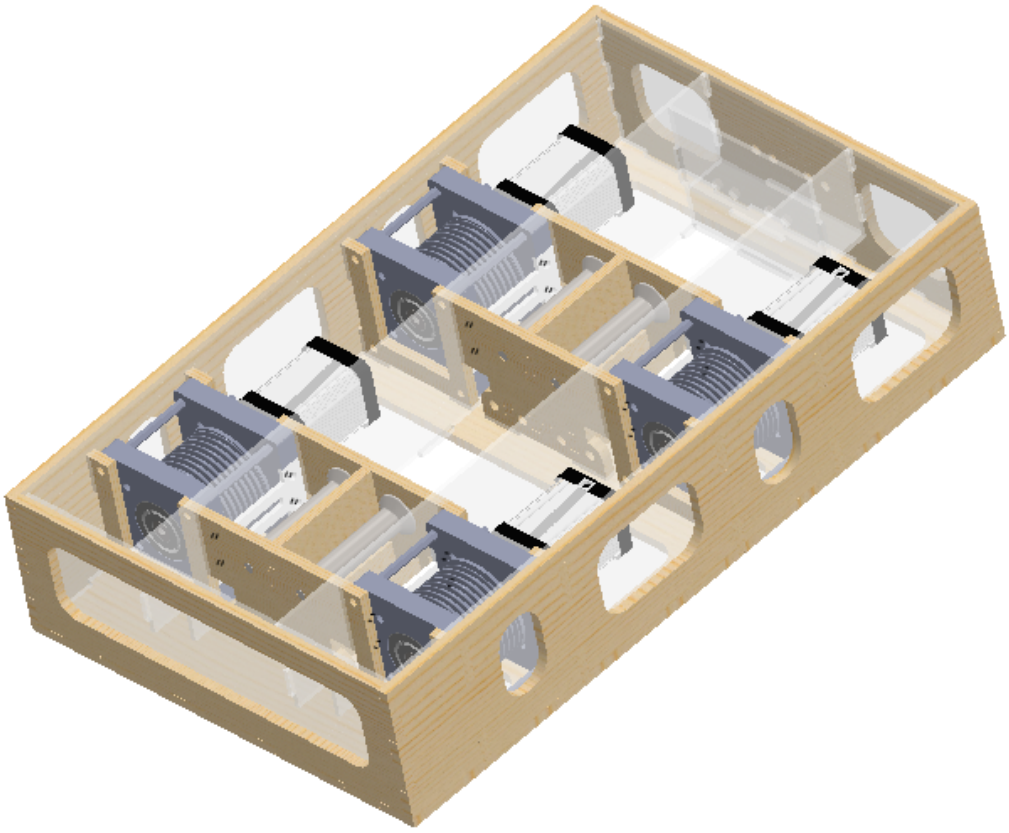


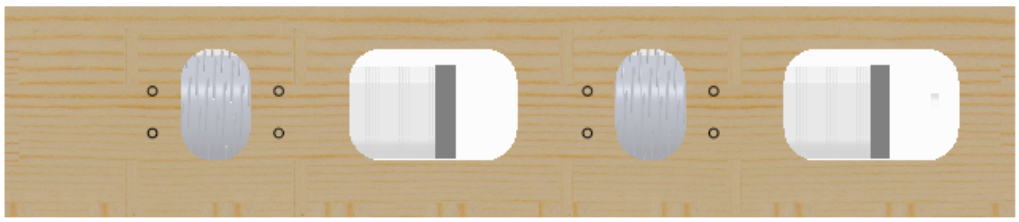


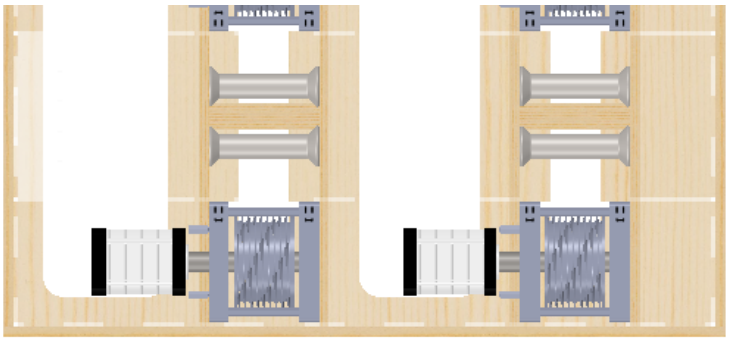


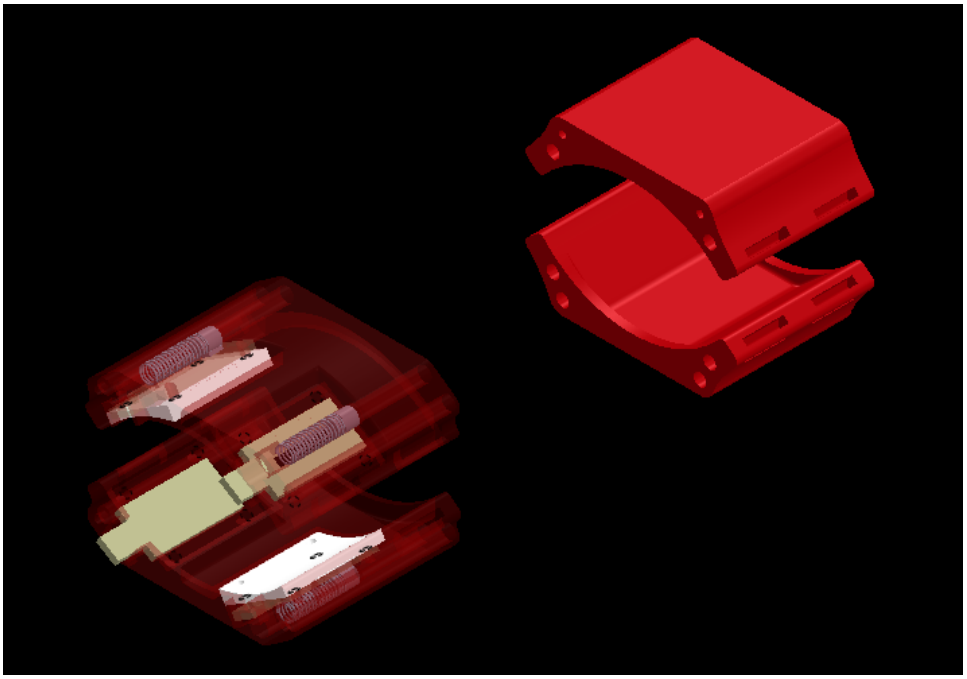


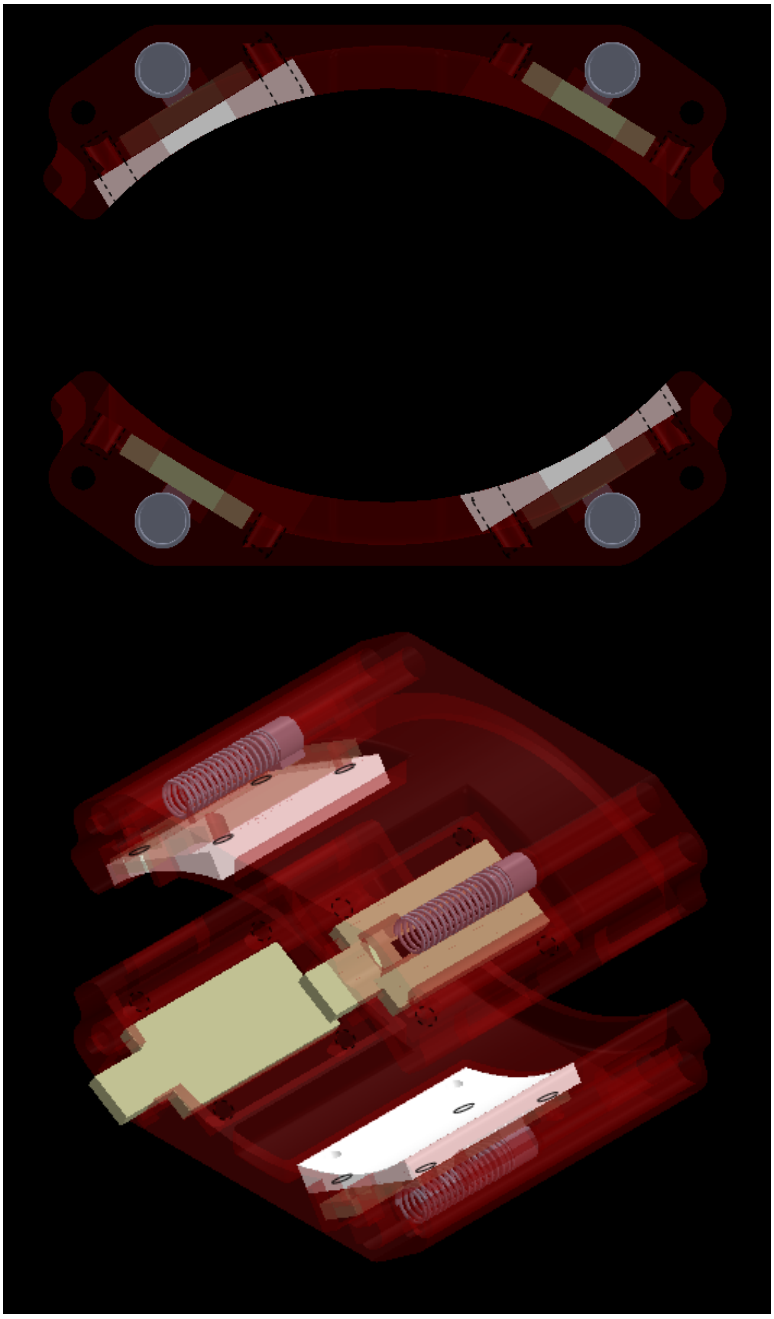


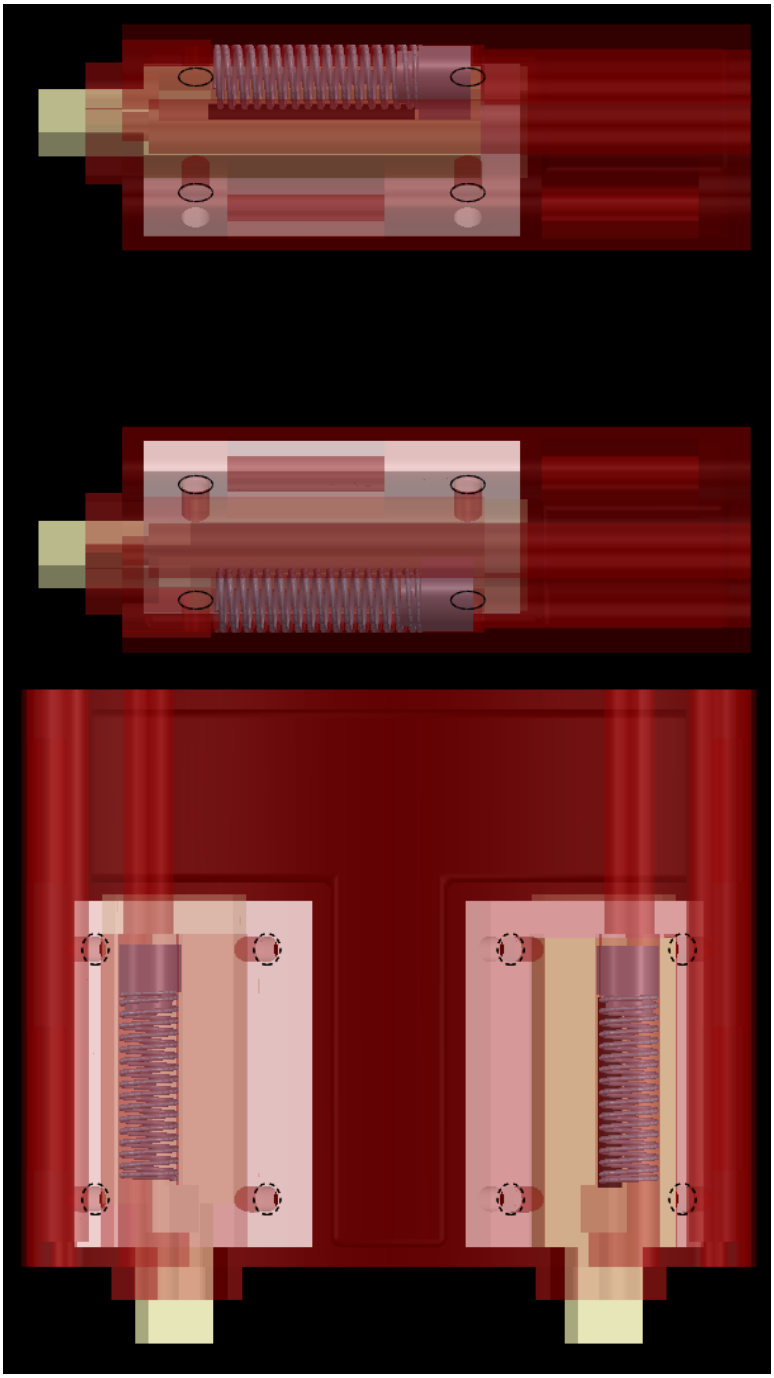












9.4) Code

9.4.1) Linear discriminant classifier analysis code

```
%%
% Linear Discrimination Classifier Code

clc; close all; clear;


%% Variables

%Rest
Rest1 = importdata('rest01.csv');
Rest2 = importdata('rest02.csv');
[R1_BB, R1_T1, R1_T2, R1_P, R1_S, R1_PQ] = getVariables(Rest1);
[R2_BB, R2_T1, R2_T2, R2_P, R2_S, R2_PQ] = getVariables(Rest2);

%Flexsion
Flexsion1 = importdata('flex01.csv');
Flexsion2 = importdata('flex02.csv');
[F1_BB, F1_T1, F1_T2, F1_P, F1_S, F1_PQ] = getVariables(Flexsion1);
[F2_BB, F2_T1, F2_T2, F2_P, F2_S, F2_PQ] = getVariables(Flexsion2);

%Extension
Extension1 = importdata('ex01.csv');
Extension2 = importdata('ex02.csv');
[E1_BB, E1_T1, E1_T2, E1_P, E1_S, E1_PQ] = getVariables(Extension1);
[E2_BB, E2_T1, E2_T2, E2_P, E2_S, E2_PQ] = getVariables(Extension2);

%Supination
Supination1 = importdata('sup01.csv');
Supination2 = importdata('sup02.csv');
[S1_BB, S1_T1, S1_T2, S1_P, S1_S, S1_PQ] = getVariables(Supination1);
[S2_BB, S2_T1, S2_T2, S2_P, S2_S, S2_PQ] = getVariables(Supination2);

%Pronation
Pronation1 = importdata('pro01.csv');
```

```
Pronation2 = importdata('pro02.csv');
[P1_BB, P1_T1, P1_T2, P1_P, P1_S, P1_PQ] = getVariables(Pronation1);
[P2_BB, P2_T1, P2_T2, P2_P, P2_S, P2_PQ] = getVariables(Pronation2);


%% Upsample and Filter

%Rest
[Rest1_BB, Rest1_T1,...
   Rest1_T2, Rest1_P,...
   Rest1_S, Rest1_PQ] = upsampleFilter(R1_BB, R1_T1, R1_T2, R1_P, R1_S,
R1_PQ);
[Rest2_BB, Rest2_T1,...
   Rest2_T2, Rest2_P,...
   Rest2_S, Rest2_PQ] = upsampleFilter(R2_BB, R2_T1, R2_T2, R2_P, R2_S,
R2_PQ);

%Flexsion
[Flexsion1_BB, Flexsion1_T1,...
   Flexsion1_T2, Flexsion1_P,....
   Flexsion1_S, Flexsion1_PQ] = upsampleFilter(F1_BB, F1_T1, F1_T2, F1_P,
F1_S, F1_PQ);
[Flexsion2_BB, Flexsion2_T1,...
   Flexsion2_T2, Flexsion2_P,...
   Flexsion2_S, Flexsion2_PQ] = upsampleFilter(F2_BB, F2_T1, F2_T2, F2_P,
F2_S, F2_PQ);

%Extension
[Extension1_BB, Extension1_T1,...
   Extension1_T2, Extension1_P,...
   Extension1_S, Extension1_PQ] = upsampleFilter(E1_BB, E1_T1, E1_T2,
E1_P, E1_S, E1_PQ);
[Extension2_BB, Extension2_T1,...
   Extension2_T2, Extension2_P,...
   Extension2_S, Extension2_PQ] = upsampleFilter(E2_BB, E2_T1, E2_T2,
E2_P, E2_S, E2_PQ);

%Supination
[Supination1_BB ,Supination1_T1,...
   Supination1_T2, Supination1_P,...
```

```
    Supination1_S, Supination1_PQ] = upsampleFilter(S1_BB, S1_T1, S1_T2,
S1_P, S1_S, S1_PQ);
    [Supination2_BB, Supination2_T1,...
    Supination2_T2, Supination2_P,...
    Supination2_S, Supination2_PQ] = upsampleFilter(S2_BB, S2_T1, S2_T2,
S2_P, S2_S, S2_PQ);

    %Pronation
    [Pronation1_BB, Pronation1_T1,...
    Pronation1_T2, Pronation1_P,...
    Pronation1_S, Pronation1_PQ] = upsampleFilter(P1_BB, P1_T1, P1_T2, P1_P,
P1_S, P1_PQ);
    [Pronation2_BB, Pronation2_T1,...
    Pronation2_T2, Pronation2_P,...
    Pronation2_S, Pronation2_PQ] = upsampleFilter(P2_BB, P2_T1, P2_T2, P2_P,
P2_S, P2_PQ);


    %% Remove Start-Up Transients & Create Small Sample Test

    %Rest
    [R1_BB,R1_T1,R1_T2,R1_P,R1_S,R1_PQ] = createTest(Rest1_BB, Rest1_T1,
Rest1_T2, Rest1_P, Rest1_S, Rest1_PQ);
    [R2_BB,R2_T1,R2_T2,R2_P,R2_S,R2_PQ] = createTest(Rest2_BB, Rest2_T1,
Rest2_T2, Rest2_P, Rest2_S, Rest2_PQ);

    %Flexsion
    [F1_BB,F1_T1,F1_T2,F1_P,F1_S,F1_PQ] = createTest(Flexsion1_BB,
Flexsion1_T1, Flexsion1_T2, Flexsion1_P, Flexsion1_S, Flexsion1_PQ);
    [F2_BB,F2_T1,F2_T2,F2_P,F2_S,F2_PQ] = createTest(Flexsion2_BB,
Flexsion2_T1, Flexsion2_T2, Flexsion2_P, Flexsion2_S, Flexsion2_PQ);

    %Extension
    [E1_BB,E1_T1,E1_T2,E1_P,E1_S,E1_PQ] = createTest(Extension1_BB,
Extension1_T1, Extension1_T2, Extension1_P, Extension1_S, Extension1_PQ);
    [E2_BB,E2_T1,E2_T2,E2_P,E2_S,E2_PQ] = createTest(Extension2_BB,
Extension2_T1, Extension2_T2, Extension2_P, Extension2_S, Extension2_PQ);

    %Supination
```

```
[S1_BB,S1_T1,S1_T2,S1_P,S1_S,S1_PQ] = createTest(Supination1_BB, Supination1_T1, Supination1_T2, Supination1_P, Supination1_S, Supination1_PQ);
[S2_BB,S2_T1,S2_T2,S2_P,S2_S,S2_PQ] = createTest(Supination2_BB, Supination2_T1, Supination2_T2, Supination2_P, Supination2_S, Supination2_PQ);

%Pronation
[P1_BB,P1_T1,P1_T2,P1_P,P1_S,P1_PQ] = createTest(Pronation1_BB, Pronation1_T1, Pronation1_T2, Pronation1_P, Pronation1_S, Pronation1_PQ);
[P2_BB,P2_T1,P2_T2,P2_P,P2_S,P2_PQ] = createTest(Pronation2_BB, Pronation2_T1, Pronation2_T2, Pronation2_P, Pronation2_S, Pronation2_PQ);


%% RMS

%Rest
[RMS_R1_BB,RMS_R1_T1,RMS_R1_T2,RMS_R1_P,RMS_R1_S,RMS_R1_PQ] = rmsEMG(R1_BB,R1_T1,R1_T2,R1_P,R1_S,R1_PQ);
[RMS_R2_BB,RMS_R2_T1,RMS_R2_T2,RMS_R2_P,RMS_R2_S,RMS_R2_PQ] = rmsEMG(R2_BB,R2_T1,R2_T2,R2_P,R2_S,R2_PQ);

%Flexsion
[RMS_F1_BB,RMS_F1_T1,RMS_F1_T2,RMS_F1_P,RMS_F1_S,RMS_F1_PQ] = rmsEMG(F1_BB,F1_T1,F1_T2,F1_P,F1_S,F1_PQ);
[RMS_F2_BB,RMS_F2_T1,RMS_F2_T2,RMS_F2_P,RMS_F2_S,RMS_F2_PQ] = rmsEMG(F2_BB,F2_T1,F2_T2,F2_P,F2_S,F2_PQ);

%Extension
[RMS_E1_BB,RMS_E1_T1,RMS_E1_T2,RMS_E1_P,RMS_E1_S,RMS_E1_PQ] = rmsEMG(E1_BB,E1_T1,E1_T2,E1_P,E1_S,E1_PQ);
[RMS_E2_BB,RMS_E2_T1,RMS_E2_T2,RMS_E2_P,RMS_E2_S,RMS_E2_PQ] = rmsEMG(E2_BB,E2_T1,E2_T2,E2_P,E2_S,E2_PQ);

%Supination
[RMS_S1_BB,RMS_S1_T1,RMS_S1_T2,RMS_S1_P,RMS_S1_S,RMS_S1_PQ] = rmsEMG(S1_BB,S1_T1,S1_T2,S1_P,S1_S,S1_PQ);
[RMS_S2_BB,RMS_S2_T1,RMS_S2_T2,RMS_S2_P,RMS_S2_S,RMS_S2_PQ] = rmsEMG(S2_BB,S2_T1,S2_T2,S2_P,S2_S,S2_PQ);

%Pronation
```

```
        [RMS_P1_BB,RMS_P1_T1,RMS_P1_T2,RMS_P1_P,RMS_P1_S,RMS_P1_PQ]
= rmsEMG(P1_BB,P1_T1,P1_T2,P1_P,P1_S,P1_PQ);
        [RMS_P2_BB,RMS_P2_T1,RMS_P2_T2,RMS_P2_P,RMS_P2_S,RMS_P2_PQ]
= rmsEMG(P2_BB,P2_T1,P2_T2,P2_P,P2_S,P2_PQ);

        %% Classifying

        % Flexsion vs. Extension
        [trainingFE] =...
           createMatrixFE(RMS_R1_BB,RMS_R1_T1,RMS_R1_T2,...
           RMS_F1_BB,RMS_F1_T1,RMS_F1_T2,...
           RMS_E1_BB,RMS_E1_T1,RMS_E1_T2);
        [sampleFE] =...
           createMatrixFE(RMS_R2_BB,RMS_R2_T1,RMS_R2_T2,...
           RMS_F2_BB,RMS_F2_T1,RMS_F2_T2,...
           RMS_E2_BB,RMS_E2_T1,RMS_E2_T2);
        [groupFE] = createGroupMatrix(3);
        %groupFE, trainingFE, sampleFE

        [resultFE, errFE, POSTERIORFE] = classify(sampleFE, trainingFE, groupFE,
'linear');
        %resultFE, errFE, POSTERIORFE

        [accuracyFE] = accuratePercent(groupFE, resultFE);
        accuracyFE

        [motionFE] = chooseMotion(resultFE);
        motionFE

        confusionFE = confusionmat(groupFE, resultFE);
        confusionFE

        % Supination vs. Pronation
        [trainingSP] =...
           createMatrixFE(RMS_R1_S,RMS_R1_P,RMS_R1_PQ,...
           RMS_F1_S,RMS_F1_P,RMS_F1_PQ,...
           RMS_E1_S,RMS_E1_P,RMS_E1_PQ);
        [sampleSP] =...
           createMatrixFE(RMS_R2_S,RMS_R2_P,RMS_R2_PQ,...
           RMS_F2_S,RMS_F2_P,RMS_F2_PQ,...
```

```
    RMS_E2_S,RMS_E2_P,RMS_E2_PQ);
[groupSP] = createGroupMatrix(3);
%groupSP, trainingSP, sampleSP

[resultSP, errSP, POSTERIORSP] = classify(sampleSP, trainingSP, groupSP,
'linear');
%resultSP, errSP, POSTERIORSP

[accuracySP] = accuratePercent(groupSP, resultSP);
accuracySP

[motionSP] = chooseMotion(resultSP);
motionSP

confusionSP = confusionmat(groupSP, resultSP);
confusionSP

% All Movements
trainingAll
=createMatrix(RMS_R1_BB,RMS_R1_T1,RMS_R1_T2,RMS_R1_P,RMS_R1_S,RMS_
R1_PQ,...

RMS_F1_BB,RMS_F1_T1,RMS_F1_T2,RMS_F1_P,RMS_F1_S,RMS_F1_PQ,...

RMS_E1_BB,RMS_E1_T1,RMS_E1_T2,RMS_E1_P,RMS_E1_S,RMS_E1_PQ,...

RMS_S1_BB,RMS_S1_T1,RMS_S1_T2,RMS_S1_P,RMS_S1_S,RMS_S1_PQ,...

RMS_P1_BB,RMS_P1_T1,RMS_P1_T2,RMS_P1_P,RMS_P1_S,RMS_P1_PQ);

sampleAll                                                    =
createMatrix(RMS_R2_BB,RMS_R2_T1,RMS_R2_T2,RMS_R2_P,RMS_R2_S,RMS_
R2_PQ,...

RMS_F2_BB,RMS_F2_T1,RMS_F2_T2,RMS_F2_P,RMS_F2_S,RMS_F2_PQ,...

RMS_E2_BB,RMS_E2_T1,RMS_E2_T2,RMS_E2_P,RMS_E2_S,RMS_E2_PQ,...

RMS_S2_BB,RMS_S2_T1,RMS_S2_T2,RMS_S2_P,RMS_S2_S,RMS_S2_PQ,...
```

```
RMS_P2_BB,RMS_P2_T1,RMS_P2_T2,RMS_P2_P,RMS_P2_S,RMS_P2_PQ);

        groupAll = createGroupMatrix(5);

        [resultAll, errAll, POSTERIORAll] = classify(sampleAll, trainingAll, groupAll,
'linear');

        [accuracyAll] = accuratePercent(groupAll, resultAll);
        accuracyAll

        [motionAll] = choose1Motion(resultAll);
        motionAll

        confusionAll = confusionmat(groupAll, resultAll);
        confusionAll
```

```
        function [BB, T1, T2, P, S, PQ] = getVariables(tf)

        % grabs the first 10,000 samples of each muscle for a test

        Full = getfield(tf,'data');
        BB = Full(20001:30000,3);
        T1 = Full(20001:30000,1);
        T2 = Full(20001:30000,2);
        P = Full(20001:30000,5);
        S = Full(20001:30000,4);
        PQ = Full(20001:30000,6);

        end
```

```
        function [BB, T1, T2, P, S, PQ] = upsampleFilter(bb, t1, t2, p, s, pq)

        % upsamples all files of one test by 2

        uBB = interp(bb,2);
```

```
uT1 = interp(t1,2);
uT2 = interp(t2,2);
uP = interp(p,2);
uS = interp(s,2);
uPQ = interp(pq,2);


% highpass filters all files of one test by 2

hpf_fc = 15;        % cut-off freq (Hz)
hpf_fs = 4000;      % sampling rate (Hz)
hpf_order = 2;      % filter order
hpf_w = 2*hpf_fc/hpf_fs;
[b,a] = butter(hpf_order,hpf_w,'high');

hpBB = filter(b,a,uBB);
hpT1 = filter(b,a,uT1);
hpT2 = filter(b,a,uT2);
hpP = filter(b,a,uP);
hpS = filter(b,a,uS);
hpPQ = filter(b,a,uPQ);


% lowpass filters all files of one test by 2

lpf_fc = 1000;      % cut-off freq (Hz)
lpf_fs = 4000;      % sampling rate (Hz)
lpf_order = 2;      % filter order
lpf_w = 2*lpf_fc/lpf_fs;
[d,c] = butter(lpf_order,lpf_w,'low');

BB = filter(d,c,hpBB);
T1 = filter(d,c,hpT1);
T2 = filter(d,c,hpT2);
P = filter(d,c,hpP);
S = filter(d,c,hpS);
PQ = filter(d,c,hpPQ);

end
```

```
function [BB,T1,T2,P,S,PQ] = createTest(bb, t1, t2, p, s, pq)

% creates multiple test segments for each
BB1 = bb(1:250,1);
BB2 = bb(251:500,1);
BB3 = bb(501:750,1);
…                                          % BB4-B79 done is similar order
BB80 = bb(19751:20000,1);

BB = zeros(250,80);
BB(:,1)=BB1; BB(:,2)=BB2; BB(:,3)=BB3; …
 BB(:,20)=BB80;

 % continue for each muscle

end
```

```
function [BB,T1,T2,P,S,PQ] = rmsEMG(bb,t1,t2,p,s,pq)

% create RMS

rms = dsp.RMS();


% compute RMS

BB = step(rms,bb);
T1 = step(rms,t1);
T2 = step(rms,t2);
P = step(rms,p);
S = step(rms,s);
PQ = step(rms,pq);

end
```

```
function [A] = createMatrixFE(BBa, T1a, T2a, BBb, T1b, T2b, BBc, T1c, T2c)
```

```
% make a training matrix

% create matrix
A = zeros(240,3);

for i=1:80
   A(i,1)=BBa(1,i);
   A((i+80),1)=BBb(1,i);
   A((i+160),1)=BBc(1,i);

   A(i,2)=T1a(1,i);
   A((i+80),2)=T1b(1,i);
   A((i+160),2)=T1c(1,i);

   A(i,3)=T2a(1,i);
   A((i+80),3)=T2b(1,i);
   A((i+160),3)=T2c(1,i);
end

end
```

```
function [A] = createGroupMatrix(stages)

% make a group matrix

A = zeros((stages*80),1);

stage = 1;
placement = 0;

while stage < (stages+1)
   while placement < (stage*80)
      A((placement+1),1)=stage;
      placement=placement+1;
   end
   stage=stage+1;
end
```

```
end
```

```
function [accurate] = accuratePercent(group, result)

[total,x] = size(group);
y = 1;
correct = 0;
wrong = 0;

while y < (total+1)
   if group(y,1) == result(y,1)
      correct = correct +1;
   else
      wrong = wrong + 1;
   end
   y = y + 1;
end

accurate = (correct/total)*100;

end
```

```
function [motion] = chooseMotion(result)

% choose a motion for each tested group

a = result(1:80,:);
b = result(81:160,:);
c = result(161:240,:);

recorded = zeros(80,3);
recorded(:,1) = a;
recorded(:,2) = b;
recorded(:,3) = c;
%recorded

motion = mode(recorded);
```

end

9.4.2) PWM motor driver test code

```
#include "mbed.h"

#define L_forward 0
#define R_forward 1
#define L_backward 1
#define R_backward 0
#define L_EN 0
#define R_EN 0
#define L_DIS 1
#define R_DIS 1
#define PERIOD 500
#define DUTY 0.3



DigitalOut EN_L(p30);
DigitalOut EN_R(p29);
DigitalOut DIR_L(p28);
DigitalOut DIR_R(p27);
PwmOut PWM_L(p26);
PwmOut PWM_R(p25);

Serial pc(USBTX, USBRX);
//DigitalOut test(p21);
DigitalOut led1(LED1);
char c = ' ';

void drive(void)
{
int p=90;
   EN_L = L_DIS;
   EN_R = R_DIS;
   pc.baud(115200);
   while(1) {
      pc.printf("enter command: ");
      c = pc.getc();
      pc.printf("\r\nwe got: %c\r\n", c);
      switch(c) {
         case 'q':
```

```
case 'Q':
   EN_L = L_DIS;
   EN_R = R_DIS;
   DIR_L = L_forward;
   PWM_L.write(DUTY);
   PWM_L.period_us(10000);
   EN_L = L_EN;
   break;
case 'w':
case 'W':
   EN_L = L_DIS;
   EN_R = R_DIS;
   DIR_R = R_forward;
   DIR_L = L_forward;
   PWM_L.write(DUTY);
   PWM_L.period_us(PERIOD);
   PWM_R.write(DUTY);
   PWM_R.period_us(PERIOD);
   EN_L = L_EN;
   EN_R = R_EN;
   break;
case 'e':
case 'E':
   EN_L = L_DIS;
   EN_R = R_DIS;
   DIR_R = R_forward;
   PWM_R.write(DUTY);
   PWM_R.period_us(PERIOD);
   EN_R = R_EN;
   break;
case 's':
case 'S':
   EN_L = L_DIS;
   EN_R = R_DIS;
   break;
case'z':
case'Z':
   EN_L = L_DIS;
   EN_R = R_DIS;
   DIR_L = L_backward;
   PWM_L.write(DUTY);
   PWM_L.period_us(PERIOD);
   EN_L = L_EN;
   break;
case 'x':
case 'X':
```

```
            EN_L = L_DIS;
            EN_R = R_DIS;
            DIR_R = R_backward;
            DIR_L = L_backward;
            PWM_L.write(DUTY);
            PWM_L.period_us(PERIOD);
            PWM_R.write(DUTY);
            PWM_R.period_us(PERIOD);
            EN_L = L_EN;
            EN_R = R_EN;
            break;
        case 'c':
        case 'C':
            EN_L = L_DIS;
            EN_R = R_DIS;
            DIR_R = R_backward;
            PWM_R.write(DUTY);
            PWM_R.period_us(PERIOD);
            EN_R = R_EN;
            break;
        case 'p':
            p = p + 10;
            pc.printf("PERIOD: %d",p);
            break;
        default:
            EN_L = L_DIS;
            EN_R = R_DIS;
            break;

    }

  }
}

int main()
{
pc.baud(115200);
  int p = 90;
  float duty = 0.1;

  for(p=5960; p>5500; p=p-20) {
    for(duty=0.1; duty<0.4; duty=duty+0.1) {
      pc.printf("p=%d, d=%f\r\n", p, duty);
      EN_L = L_DIS;
      EN_R = R_DIS;
      DIR_R = R_forward;
```

```
            DIR_L = L_forward;
            PWM_L.write(DUTY);
            PWM_L.period_us(PERIOD);
            PWM_R.write(DUTY);
            PWM_R.period_us(PERIOD);
            EN_L = L_EN;
            EN_R = R_EN;
            wait(4);
        }
    }
}
```

9.4.3) EMG acquisition code

```
#include "mbed.h"
//p5  --spi
//p6  --spi
//p7  --spi
//p8  --CS
//p9  --busy
//p10 --CNVST

SPI spi(p5, p6, p7); // mosi, miso, sclk
//DigitalOut SCLK(p7);
//DigitalIn CHA(p6);
//DigitalIn CHB(p5);
DigitalOut CS1(p18);
DigitalOut CS2(p19);
DigitalOut CS3(p20);
InterruptIn BUSY1(p12);
InterruptIn BUSY2(p11);
InterruptIn BUSY3(p13);
DigitalOut CNVST(p14);
Serial pc(USBTX, USBRX);
DigitalOut led1(LED1);
DigitalOut led2(LED2);
DigitalOut led3(LED3);
Ticker EMGTrigger;
//union Bint { int x; unsigned char bytes[4]; } Bint;
int x,j;
bool NewData1, NewData2, NewData3, ConvStart1, ConvStart2, ConvStart3;
#define CIR_C_Q_SIZE 500;
#define NUM_CHAN 6;
#define RMS_WINDOW 250;
int signalData[NUM_CHAN][CIRC_Q_SIZE];       //raw digitized signal -- make
int16_t
```

```
        int writeIndex[NUM_CHAN];//, readIndex[NUM_CHAN];          //last write and
read positions to signalData[][], zero indexed
        int rmsStart[NUM_CHAN], rmsStop[NUM_CHAN];              //Indices denoting
current root mean square window
        int dtatPoint[NUM_CHAN];
        float currentMS[NUM_CHAN];        //current mean square values

        void startConvert();
        void initADC();
        //int readADC();
        void readADC(int ADC, unsigned int *ch1, unsigned int *ch2);
        void SPIinit();
        int dataShift(unsigned int data);
        void BusyRose();
        void BusyFell();


        int main() {
        unsigned int cha, chb, chc, chd, che, chf;
        for (int i=0; i<NUM_CHAN; i++;){
        dtatPoint[i] = 0;
        currentMS = 0;
        }
        led1=0;
        led2=0;
        led3=0;
        SPIinit();
        initADC();
        BUSY1.rise(&BusyRose);
        BUSY2.rise(&BusyRose);
        BUSY3.rise(&BusyRose);
        BUSY1.fall(&BusyFell);
        BUSY2.fall(&BusyFell);
        BUSY3.fall(&BusyFell);
        EMGTrigger.attach(&startConvert, 0.00025);

        pc.printf("hello\n");
        int newlinecounter = 0;
        while(1){
           led1=0;
           led2=0;
           led3=0;
            //Bint.x=readADC();
            //startConvert();
            led1=1;
            led2=1;
```

```
            if (ConvStart1 && NewData1 && ConvStart2 && NewData2 && ConvStart3
&& NewData3){
                readADC(1, &cha, &chb);
                readADC(2, &chc, &chd);
                readADC(3, &che, &chf);}
            //pc.printf("0x%x%x%x\n\r", Bint.bytes[2],Bint.bytes[1],Bint.bytes[0]);
            led3=1;
            if (!ConvStart1 && NewData1 && !ConvStart2 && NewData2 &&
!ConvStart3 && NewData3){
            pc.printf("%u, %u, %u, %u, %u, %u \r\n", dataShift(cha), dataShift(chb),
dataShift(chc),dataShift(chd),dataShift(che),dataShift(chf),);
            dataPoint[0] = dataShift(cha);
            dataPoint[1] = dataShift(chb);
            dataPoint[2] = dataShift(chc);
            dataPoint[3] = dataShift(chd);
            dataPoint[4] = dataShift(che);
            dataPoint[5] = dataShift(chf);
            for(int EMGline=0; EMGline <=NUM_CHAN; EMGline++){
              for(;                                (rmsStop[EMGline]-1)!=writeIndex[EMGline];
incCircArray(rmsStart+EMGline, CIRC_Q_SIZE)){       //iterate through new values
                currentMS[EMGline]=currentMS[EMGline]
                                  +
(signalData[EMGline][rmsStop[EMGline]]*signalData[EMGline][rmsStop[EMGline]])/
RMS_WINDOW
                                  -
(signalData[EMGline][rmsStop[EMGline]]*signalData[EMGline][rmsStop[EMGline]])/
RMS_WINDOW;
                  incCircArray(rmsStop+EMGline, CIRC_Q_SIZE);
              }
              NewData1 = false;
              NewData2 = false;
              NewData3 = false;
            }

           }
        //    if(newlinecounter == 3){
        //      pc.printf("\r\n");
        //      newlinecounter = 0;
        //    }
            //pc.printf("hello\n");
            for(x=0;x<5000;x++){
            for(j=0;j<1000;j++);
            }
        }
        }
```

```
void BusyRose(){
ConvStart1 = BUSY1;
ConvStart2 = BUSY2;
ConvStart3 = BUSY3;
}

void BusyFell(){
NewData1 = !BUSY1;
NewData2 = !BUSY2;
NewData3 = !BUSY3;}

void initADC(){
   CNVST=1;
   CS1=1;
   CS2=1;
   CS3=1;
   //SCLK=0;
}

//int readADC(){
//    int inBits = 0;
//    CNVST = 0;
//    int idleVariable = 0;
//    CNVST = 1;
//    while(BUSY);
//    CS = 0;
//    inBits = spi.write(0);
//    //need to bit shift, but usure extactly
//    CS = 1;
//
//    return inBits;
//}


//starts conversions
void startConvert(){
   CNVST = 0;
   int idleVar =1;
//    while(!flag){
//    pc.printf("not yet");}
   CNVST = 1;
}

//read adc and return results as unsigned ints for processing
void readADC(int ADC, unsigned int *ch1, unsigned int *ch2){
//    //int[14] cha, chb;
```

```
//    //bzero(cha, sizeof(int)*14);
//    //bzero(chb, sizeof(int)*14);
    unsigned int in;
    int pin1 = 0, pin2 = 0;
    switch(ADC){
        case 0:
            break;
        case 1:
            CS1=0;
            ConvStart1=false;
            break;
        case 2:
            CS2=0;
            ConvStart2=false;
            break;
        case 3:
            CS3=0;
            ConvStart3=false;
            break;
        default:
            break;
    }
    in = spi.write(0);
    wait_us(1);
    *ch1 = in >> 1;
    in = spi.write(0);
    wait_us(1);
    *ch2 = (in & 0x00003fff) ;
    CS1=1;
    CS2=1;
    CS3=1;
}

//change fourteen bit unsigned to 32 bit int
int dataShift(unsigned int data){
    if(data & 0x2000){ //check if negative
        data = data & !0x2000; //get first thriteen bits
        data = ((!data) & !0x2000) + 0x1; //invert and add one
        int DATA = data; //moved to signed int
        DATA = DATA * -1;
        return DATA;
    }else{
        return data;
    }
}
```

```
        void SPIinit(){
           spi.format(14,2);
           spi.frequency(1000000);
        }

        void incrementMeanSquareWindow(){
           for(int EMGline=0; EMGline <=NUM_CHAN; EMGline++){
              for(dataChange=False;          (rmsStop[EMGline]-1)!=writeIndex[EMGline];
incCircArray(rmsStart+EMGline, CIRC_Q_SIZE)){       //iterate through new values
                 currentMS[EMGline]=currentMS[EMGline]
                                 +
(signalData[EMGline][rmsStop[EMGline]]*signalData[EMGline][rmsStop[EMGline]])/
RMS_WINDOW
                                 -
(signalData[EMGline][rmsStop[EMGline]]*signalData[EMGline][rmsStop[EMGline]])/
RMS_WINDOW;
                 incCircArray(rmsStop+EMGline, CIRC_Q_SIZE);
              }
           }
        }
```

#.#.# Java classifier building code [3 .class files]

```
        import java.io.BufferedReader;
        import java.io.BufferedWriter;
        import java.io.File;
        import java.io.FileNotFoundException;
        import java.io.FileReader;
        import java.io.FileWriter;
        import java.io.IOException;
        import java.io.InputStreamReader;
        import java.io.PrintWriter;
        import java.io.StringReader;
        import java.lang.management.ManagementFactory;
        import java.lang.management.OperatingSystemMXBean;
        import java.lang.reflect.Array;
        import java.net.InetAddress;
        import java.net.ServerSocket;
        import java.net.Socket;
        import java.net.UnknownHostException;
        import java.util.ArrayList;
        import java.util.Collections;
        import java.util.Iterator;
        import java.util.Random;
        import java.util.Vector;
        import java.util.concurrent.ArrayBlockingQueue;
```

```
import javax.xml.crypto.Data;



import weka.*;
import weka.classifiers.Evaluation;
import weka.classifiers.trees.J48;
import weka.classifiers.trees.m5.CorrelationSplitInfo;
import weka.core.Instances;
import weka.core.converters.CSVLoader;

public class Main {

	/**
	 * @param args
	 */
	public static void main(String[] args) {//args [master/slave/analyze] [ece/linux/ccc] [#ofthreads]
		boolean master = false;
		if(args[0].toLowerCase().contains("master")){
			System.out.println("starting master");
			master = true;
			Master();
		}
		if(args[0].toLowerCase().equals("slave")){
			master = false;
			slave(args[1], Integer.parseInt(args[2]));
		}
		if(args[0].toLowerCase().equals("analyze")){
			Analyzer();
		}


	}

	public static void Analyzer(){

	}

	public static void Master(){
		int error =0;
		int correct =0;
		int STRATA_START = 5;
		int STRATA_MAX = 300;
		int STRATA_GAP = 5;
```

```
int WINDOW_START = 200;
int WINDOW_MAX = 1500;
int WINDOW_GAP = 10;
ArrayList<Socket> sockets = new ArrayList<Socket>();
//		ArrayList<Socket> readysockets = new ArrayList<Socket>();
ServerSocket serverSocket;
int i=5, k=50, count =0;
serverSocket = null;
//		System.out.println("master making socket");
//		try {
//			serverSocket = new ServerSocket(2500);
//		} catch (IOException e) {
//			System.err.println("Could not listen on port: 2500.");
//			System.exit(1);
//		}
System.out.println("master making locals");
OperatingSystemMXBean OSMXB = ManagementFactory.getOperatingSystemMXBean();
ServerConnectionHandler SCH = new ServerConnectionHandler();
Thread SCH_Thread = new Thread(SCH);
SCH_Thread.start();
ArrayList<Thread> threads = new ArrayList<Thread>();
ArrayList<DT_OldHist> DTs = new ArrayList<DT_OldHist>();
ArrayList<ArrayList<Integer>> ordersToBeSent = new ArrayList<ArrayList<Integer>>();
ArrayList<ArrayList<Integer>> ordersOut = new ArrayList<ArrayList<Integer>>();
ArrayList<ArrayList<Double>> results = new ArrayList<ArrayList<Double>>();
System.out.println("master making space for instructions and results");
for(i = STRATA_START;i<=STRATA_MAX;i=i+STRATA_GAP){ //strata 300
	for(k = WINDOW_START; k<=WINDOW_MAX; k=k+WINDOW_GAP){ //window 1000
		ArrayList<Integer> temp = new ArrayList<Integer>();
		temp.add(i);
		temp.add(k);
		ordersToBeSent.add(new ArrayList<Integer>(temp));
		ArrayList<Double> Rtemp = new ArrayList<Double>();
```

```
                        Rtemp.add(-2.0); //acc
                        Rtemp.add(-2.0); //strata
                        Rtemp.add(-2.0); //window
                        Rtemp.add(-2.0); //Algorith
                        results.add(new ArrayList<Double>(Rtemp));
                        //results.add(-2.0);
                }
        }
        //TODO testserver only
        //              for(int t =0; t<2000; t++){
        //                      results.add(-2.0);
        //              }

        System.out.println("starting master loop");

        while(ordersToBeSent.size()>0 || ordersOut.size()>0){

                while(SCH.Sockets.size()>0){
                        System.out.println("master adding socket");
                        sockets.add(SCH.Sockets.remove());
                }


                //check sockets

                Iterator<Socket> SocketItr = sockets.iterator();
                while(SocketItr.hasNext()){
                        Socket s = SocketItr.next();
                        BufferedReader in = null;
                        PrintWriter out = null;
                        try{
                                out = new PrintWriter(s.getOutputStream(),
true); //make write
                                in = new BufferedReader( //make reader
                                                new InputStreamReader(

        s.getInputStream()));


                                while(in.ready()){ //while socket has
something to say
                                        System.out.println("we got a
message");
                                        String msg = in.readLine(); //get the
return message
                                        if(msg.contains("sendTask")){
```

```
                                                        System.out.println(msg);
                                                        System.out.println("sending
order");
                                                        if(ordersToBeSent.size()>0){
                                                                ArrayList<Integer>
AI = ordersToBeSent.get(0);

        ordersToBeSent.remove(0);
                                                                ordersOut.add(new
ArrayList<Integer>(AI));

        System.out.println("sent Order: Strata: " + AI.get(0) + " window: " + AI.get(1));
                                                                out.println("order:"  +
AI.get(0) + ":" + AI.get(1));
                                                        }else if(ordersOut.size()>0){
                                                                Random generator =
new Random();
                                                                ArrayList<Integer>
AI = ordersOut.get(generator.nextInt( ordersOut.size()));

        System.out.println("re-sent Order: Strata: " + AI.get(0) + " window: " +
AI.get(1));
                                                                out.println("re-order:"
+ AI.get(0) + ":" + AI.get(1));
                                                        }else{
                                                                out.println("none");

                                                        }
                                                        System.out.println("we  still
have " + ordersOut.size() + " out and "+ ordersToBeSent.size() + " orders left to be
sent");
                                                        System.out.println("error = "
+ error + " correct = " +correct);
                                                }
                                                if(msg.contains("return")){   //TODO
deal with return
                                                        System.out.println("value
incoming : " + msg);
                                                        double d = 0.0;
                                                        int I=0, K=0;
                                                        String[] res = msg.split(":");
//return:double:int:int
                                                        d                       =
Double.parseDouble(res[1]);
                                                        if(d>0.0){
```

```
                                                    I = Integer.parseInt(res[2]); //strata
                                                    K = Integer.parseInt(res[3]); //window
                                                    int index = ((I-STRATA_START)/STRATA_GAP);
                                                    //index *= 100;
                                                    index *= ((WINDOW_MAX-WINDOW_START)/WINDOW_GAP);
                                                    index += ((K-WINDOW_START)/WINDOW_GAP);

        System.out.println("returned: Strata: " + I + " window: " + K + " resultIndex: " + index + " result: " + d);

        if(d>results.get(index).get(0)){

        results.get(index).set(0, d); //set acc

        results.get(index).set(1, Double.parseDouble(res[2])); //set strata

        results.get(index).set(2, Double.parseDouble(res[3])); //set window

        results.get(index).set(3, 0.0); //set algorithm type
                                                    }
                                                    try {
                                                        //
                                            BufferedWriter csvout = new BufferedWriter(new FileWriter(new File("M:\\Ex2_csv\\endresults.csv")));

        BufferedWriter csvout = new BufferedWriter(new FileWriter(new File("endresults.csv")));

        for(ArrayList<Double> AD : results){
                                                                int counntt =0;

        for(Double dd : AD){

        if(count ==0){

        csvout.write(dd + ", ");

        counntt++;

        }else{
```

```
        csvout.write(dd.intValue() + ", ");

        }
                                                                    }

        csvout.write("\n");
                                                                }
                                                                csvout.close();
                                                            } catch (IOException
e) {

        System.out.println("shit the output failed. we cant make the csv!");

        e.printStackTrace();
                                                            }

        Iterator<ArrayList<Integer>> itr = ordersOut.iterator();
                                                            while(itr.hasNext()){

        ArrayList<Integer> AI = itr.next();

        if(AI.get(0)==I && AI.get(1)==K){

        System.out.println("removed: Strata: " + AI.get(0) + " Window: " + AI.get(1));

        itr.remove();

        System.out.println("we still have " + ordersOut.size() + " out and "+
ordersToBeSent.size() + " orders left to be sent. " );

        System.out.println("error = " + error + " correct = " +correct);
                                                                    break;
                                                                }
                                                            }

                                                        }else{

        System.out.println("we had an error returned");
                                                        }
                                                    }
                                                }
                                            } catch (IOException e) { // our socket is dead
                                                SocketItr.remove();
                                                System.out.println("a Socket is dead");
```

```
                                        //
        e.printStackTrace();
                                }

                        }




                }
                System.out.println("master done");

                Iterator<Socket> SocketItr = sockets.iterator();
                while(SocketItr.hasNext()){
                        Socket s = SocketItr.next();
                        BufferedReader in = null;
                        PrintWriter out = null;
                        try{
                                out = new PrintWriter(s.getOutputStream(), true);
//make write
                                in = new BufferedReader( //make reader
                                                new InputStreamReader(
                                                                s.getInputStream()));

                                out.write("done");
                        }catch(Exception e){

                        }
                }

                try {
                        //                      BufferedWriter csvout = new
BufferedWriter(new FileWriter(new File("M:\\Ex2_csv\\endresults.csv")));
                        BufferedWriter csvout = new BufferedWriter(new
FileWriter(new File("endresults.csv")));
                        for(ArrayList<Double> AD : results){
                                for(Double dd : AD){
                                        csvout.write(dd + ",");
                                }
                                csvout.write("\n");
                        }
                        csvout.close();
                } catch (IOException e) {
                        System.out.println("shit the output failed. we cant make the
csv!");
```

```
                    e.printStackTrace();
                }

            }


            public static void Master2(){
                int error =0;
                int correct =0;
                int STRATA_START = 5;
                int STRATA_MAX = 300;
                int STRATA_GAP = 5;
                int WINDOW_START = 200;
                int WINDOW_MAX = 1500;
                int WINDOW_GAP = 10;
                ArrayList<Socket> sockets = new ArrayList<Socket>();
                //              ArrayList<Socket>       readysockets        =       new
ArrayList<Socket>();
                ServerSocket serverSocket;
                int i=5, k=50, count =0;
                serverSocket = null;
                //              System.out.println("master making socket");
                //              try {
                //                      serverSocket = new ServerSocket(2500);
                //              } catch (IOException e) {
                //                      System.err.println("Could not listen on port:
2500.");
                //                      System.exit(1);
                //              }
                System.out.println("master making locals");
                OperatingSystemMXBean                   OSMXB                   =
ManagementFactory.getOperatingSystemMXBean();
                ServerConnectionHandler         SCH             =               new
ServerConnectionHandler();
                Thread SCH_Thread = new Thread(SCH);
                SCH_Thread.start();
                ArrayList<String> returns = new ArrayList<String>();
                ArrayList<Thread> threads = new ArrayList<Thread>();
                ArrayList<DT_OldHist> DTs = new ArrayList<DT_OldHist>();
                ArrayList<ArrayList<Integer>>       ordersToBeSent      =       new
ArrayList<ArrayList<Integer>>();
                ArrayList<ArrayList<Integer>>       ordersOut       =       new
ArrayList<ArrayList<Integer>>();
                ArrayList<ArrayList<Double>>        results         =       new
ArrayList<ArrayList<Double>>();
```

System.out.println("master making space for instructions and results");

Integer[] window = { 1130, 1330, 1370, 1430, 1440, 1450, 1460, 1490, 390, 430, 440, 450, 460, 470, 480, 490, 500, 510, 520, 530, 540, 550, 560, 570, 580, 590, 600, 610, 620, 630, 640, 650, 660, 670, 680, 690, 700, 710, 720, 730, 740, 750, 760, 770, 780, 790, 800, 810, 820, 830, 840, 850, 860, 870, 880, 890, 900, 910, 920, 930, 940, 950, 960, 970, 980, 990, 1000, 1010, 1020, 1030, 1040, 1050, 1060, 1070, 1080, 1090, 1100, 1110, 1120, 1130, 1140, 1150, 1160, 1170, 1180, 1190, 1200, 1210, 1220, 1230, 1240, 1250, 1260, 1270, 1280, 1290, 1300, 1310, 1320, 1330, 1340, 1350, 1360, 1370, 1380, 1390, 1400, 1410, 1420, 1430, 1440, 1450, 1460, 1470, 1480, 1490, 1500, 510, 520, 580, 590, 600, 610, 620, 630, 640, 650, 660, 670, 680, 690, 700, 710, 720, 740, 750, 760, 770, 780, 790, 800, 810, 820, 830, 840, 850, 860, 870, 880, 890, 900, 910, 920, 930, 940, 950, 960, 970, 980, 990, 1000, 1010, 1020, 1030, 1040, 1050, 1060, 1070, 1080, 1090, 1100, 1110, 1120, 1130, 1140, 1150, 1160, 1170, 1180, 1190, 1200, 1210, 1220, 1230, 1240, 1250, 1260, 1270, 1280, 1290, 1300, 1310, 1320, 1330, 1340, 1350, 1360, 1370, 1380, 1390, 1400, 1410, 1420, 1430, 1440, 1450, 1460, 1470, 1480, 1490, 1500, 360, 380, 390, 400, 420, 430, 440, 450, 460, 470, 480, 490, 500, 510, 520, 530, 540, 550, 560, 570, 580, 590, 600, 610, 620, 630, 640, 650, 660, 670, 680, 690, 700, 710, 720, 730, 740, 750, 760, 770, 780, 790, 800, 810, 820, 830, 840, 850, 860, 870, 880, 890, 900, 910, 920, 930, 940, 950, 960, 970, 980, 990, 1000, 1010, 1020, 1030, 1040, 1050, 1060, 1070, 1080, 1090, 1100, 1110, 1120, 1130, 1140, 1150, 1160, 1170, 1180, 1190, 1200, 1210, 1220, 1230, 1240, 1250, 1260, 1270, 1280, 1290, 1300, 1310, 1320, 1330, 1340, 1350, 1360, 1370, 1380, 1390, 1400, 1410, 1420, 1430, 1440, 1450, 1460, 1470, 1480, 1490, 1500, 410, 440, 450, 460, 470, 480, 490, 500, 510, 520, 530, 540, 550, 560, 570, 580, 590, 600, 610, 620, 630, 640, 650, 660, 670, 680, 690, 700, 710, 720, 730, 740, 750, 760, 770, 780, 790, 800, 810, 820, 830, 840, 850, 860, 870, 880, 890, 900, 910, 920, 930, 940, 950, 960, 970, 980, 990, 1000, 1010, 1020, 1030, 1040, 1050, 1060, 1070, 1080, 1090, 1100, 1110, 1120, 1130, 1140, 1150, 1160, 1170, 1180, 1190, 1200, 1210, 1220, 1230, 1240, 1250, 1260, 1270, 1280, 1290, 1300, 1310, 1320, 1330, 1340, 1350, 1360, 1370, 1380, 1390, 1400, 1410, 1420, 1430, 1440, 1450, 1460, 1470, 1480, 1490, 1500, 470, 480, 490, 500, 510, 520, 530, 540, 550, 560, 570, 580, 590, 600, 610, 620, 630, 640, 650, 660, 670, 680, 690, 700, 710, 720, 730, 740, 750, 760, 770, 780, 790, 800, 810, 820, 830, 840, 850, 860, 870, 880, 890, 900, 910, 920, 930, 940, 950, 960, 970, 980, 990, 1000, 1010, 1020, 1030, 1040, 1050, 1060, 1070, 1080, 1090, 1100, 1110, 1120, 1130, 1140, 1150, 1160, 1170, 1180, 1190, 1200, 1210, 1220, 1230, 1240, 1250, 1260, 1270, 1280, 1290, 1300, 1310, 1320, 1330, 1340, 1350, 1360, 1370, 1380, 1390, 1400, 1410, 1420, 1430, 1440, 1450, 1460, 1470, 1480, 1490, 1500, 500, 530, 540, 550, 570, 580, 590, 610, 620, 630, 640, 650, 660, 680, 690, 700, 710, 720, 730, 740, 750, 760, 770, 780, 790, 800, 810, 820, 830, 840, 850, 860, 870, 880, 890, 900, 910, 920, 930, 940, 950, 960, 970, 980, 990, 1000, 1010, 1020, 1030, 1040, 1050, 1060, 1070, 1080, 1090, 1100, 1110, 1120, 1130, 1140, 1150, 1160, 1170, 1180, 1190, 1200, 1210, 1220, 1230, 1240, 1250, 1260, 1270, 1280, 1290, 1300, 1310, 1320, 1330, 1340, 1350, 1360, 1370, 1380, 1390, 1400, 1410, 1420, 1430, 1440, 1450, 1460, 1470, 1480, 1490, 1500, 470, 480, 490, 510, 530, 540, 550, 560, 570, 580, 590, 600, 610, 620, 630, 640, 650, 660, 670, 680, 690, 700, 710, 720, 730, 740, 750, 760, 770, 780, 790, 800, 810, 820, 830, 840, 850, 860, 870, 880, 890, 900, 910, 920, 930, 940, 950, 960, 970, 980, 990, 1000, 1010, 1020, 1030,

1040, 1050, 1060, 1070, 1080, 1090, 1100, 1110, 1120, 1130, 1140, 1150, 1160, 1170, 1180, 1190, 1200, 1210, 1220, 1230, 1240, 1250, 1260, 1270, 1280, 1290, 1300, 1310, 1320, 1330, 1340, 1350, 1360, 1370, 1380, 1390, 1400, 1410, 1420, 1430, 1440, 1450, 1460, 1470, 1480, 1490, 1500, 580, 590, 600, 610, 620, 630, 640, 650, 660, 670, 680, 690, 700, 710, 720, 730, 740, 750, 760, 770, 780, 790, 800, 810, 820, 830, 840, 850, 860, 870, 880, 890, 900, 910, 920, 930, 940, 950, 960, 970, 980, 990, 1000, 1010, 1020, 1030, 1040, 1050, 1060, 1070, 1080, 1090, 1100, 1110, 1120, 1130, 1140, 1150, 1160, 1170, 1180, 1190, 1200, 1210, 1220, 1230, 1240, 1250, 1260, 1270, 1280, 1290, 1300, 1310, 1320, 1330, 1340, 1350, 1360, 1370, 1380, 1390, 1400, 1410, 1420, 1430, 1440, 1450, 1460, 1470, 1480, 1490, 1500, 580, 590, 600, 620, 630, 640, 650, 660, 670, 680, 690, 700, 710, 720, 730, 740, 750, 760, 770, 780, 790, 800, 810, 820, 830, 840, 850, 860, 870, 880, 890, 900, 910, 920, 930, 940, 950, 960, 970, 980, 990, 1000, 1010, 1020, 1030, 1040, 1050, 1060, 1070, 1080, 1090, 1100, 1110, 1120, 1130, 1140, 1150, 1160, 1170, 1180, 1190, 1200, 1210, 1220, 1230, 1240, 1250, 1260, 1270, 1280, 1290, 1300, 1310, 1320, 1330, 1340, 1350, 1360, 1370, 1380, 1390, 1400, 1410, 1420, 1430, 1440, 1450, 1460, 1470, 1480, 1490, 1500, 480, 510, 580, 590, 600, 610, 630, 640, 650, 660, 680, 690, 700, 710, 720, 730, 740, 750, 760, 770, 780, 790, 800, 810, 820, 830, 840, 850, 860, 870, 880, 890, 900, 910, 920, 930, 940, 950, 960, 970, 980, 990, 1000, 1010, 1020, 1030, 1040, 1050, 1060, 1070, 1080, 1090, 1100, 1110, 1120, 1130, 1140, 1150, 1160, 1170, 1180, 1190, 1200, 1210, 1220, 1230, 1240, 1250, 1260, 1270, 1280, 1290, 1300, 1310, 1320, 1330, 1340, 1350, 1360, 1370, 1380, 1390, 1400, 1410, 1420, 1430, 1440, 1450, 1460, 1470, 1480, 1490, 1500, 610, 640, 660, 670, 680, 690, 700, 710, 720, 730, 750, 760, 770, 780, 790, 800, 810, 820, 830, 840, 850, 860, 870, 880, 890, 900, 910, 920, 930, 940, 950, 960, 970, 980, 990, 1000, 1010, 1020, 1030, 1040, 1050, 1060, 1070, 1080, 1090, 1100, 1110, 1120, 1130, 1140, 1150, 1160, 1170, 1180, 1190, 1200, 1210, 1220, 1230, 1240, 1250, 1260, 1270, 1280, 1290, 1300, 1310, 1320, 1330, 1340, 1350, 1360, 1370, 1380, 1390, 1400, 1410, 1420, 1430, 1440, 1450, 1460, 1470, 1480, 1490, 1500, 640, 670, 690, 700, 710, 720, 730, 740, 760, 770, 780, 790, 800, 810, 820, 830, 840, 850, 870, 880, 890, 900, 910, 920, 930, 940, 950, 960, 970, 980, 990, 1000, 1010, 1020, 1030, 1040, 1050, 1060, 1070, 1080, 1090, 1100, 1110, 1120, 1130, 1140, 1150, 1160, 1170, 1180, 1190, 1200, 1210, 1220, 1230, 1240, 1250, 1260, 1270, 1280, 1290, 1300, 1310, 1320, 1330, 1340, 1350, 1360, 1370, 1380, 1390, 1400, 1410, 1420, 1430, 1440, 1450, 1460, 1470, 1480, 1490, 1500, 680, 690, 700, 740, 750, 760, 770, 780, 790, 800, 810, 820, 830, 840, 850, 870, 890, 900, 910, 920, 930, 940, 950, 960, 970, 980, 990, 1000, 1010, 1020, 1030, 1040, 1050, 1060, 1070, 1080, 1090, 1100, 1110, 1120, 1130, 1140, 1150, 1160, 1170, 1180, 1190, 1200, 1210, 1220, 1230, 1240, 1250, 1260, 1270, 1280, 1290, 1300, 1310, 1320, 1330, 1340, 1350, 1360, 1370, 1380, 1390, 1400, 1410, 1420, 1430, 1440, 1450, 1460, 1470, 1480, 1490, 1500, 750, 770, 790, 800, 810, 820, 830, 840, 850, 860, 880, 890, 910, 920, 930, 940, 950, 960, 970, 980, 990, 1000, 1010, 1020, 1030, 1040, 1050, 1060, 1070, 1080, 1090, 1100, 1110, 1120, 1130, 1140, 1150, 1160, 1170, 1180, 1190, 1200, 1210, 1220, 1230, 1240, 1250, 1260, 1270, 1280, 1290, 1300, 1310, 1320, 1330, 1340, 1350, 1360, 1370, 1380, 1390, 1400, 1410, 1420, 1430, 1440, 1450, 1460, 1470, 1480, 1490, 1500, 780, 790, 800, 810, 820, 830, 860, 870, 880, 900, 910, 930, 940, 950, 960, 970, 990, 1010, 1020, 1030, 1040, 1050, 1060, 1070, 1080, 1090, 1100, 1110, 1120, 1130, 1150, 1160, 1170, 1180, 1200, 1210, 1220, 1230, 1240, 1250, 1260, 1270, 1280, 1290, 1300, 1310, 1320, 1330, 1340, 1350, 1360, 1370, 1380, 1390,

```
1400, 1410, 1420, 1430, 1440, 1450, 1460, 1470, 1480, 1490, 1500, 750, 810, 820, 840,
860, 870, 880, 910, 920, 930, 940, 950, 970, 980, 990, 1000, 1010, 1020, 1030, 1040,
1050, 1060, 1070, 1080, 1090, 1100, 1110, 1120, 1130, 1140, 1150, 1160, 1170, 1180,
1190, 1200, 1210, 1220, 1230, 1240, 1250, 1260, 1270, 1280, 1290, 1300, 1310, 1320,
1330, 1340, 1350, 1360, 1370, 1380, 1390, 1400, 1410, 1420, 1430, 1440, 1450, 1460,
1470, 1480, 1490, 1500, 790, 900, 930, 940, 960, 970, 990, 1000, 1010, 1020, 1030,
1040, 1060, 1070, 1080, 1090, 1100, 1120, 1130, 1160, 1170, 1180, 1190, 1200, 1210,
1240, 1250, 1260, 1270, 1280, 1290, 1300, 1310, 1320, 1330, 1340, 1350, 1360, 1370,
1380, 1390, 1400, 1410, 1420, 1430, 1440, 1450, 1460, 1470, 1480, 1490, 1500, 830,
840, 890, 910, 930, 940, 960, 970, 980, 990, 1000, 1010, 1020, 1030, 1040, 1060, 1070,
1080, 1090, 1100, 1110, 1120, 1130, 1140, 1150, 1160, 1170, 1180, 1190, 1200, 1210,
1220, 1230, 1240, 1250, 1260, 1290, 1300, 1310, 1320, 1330, 1340, 1350, 1360, 1370,
1380, 1390, 1400, 1410, 1420, 1430, 1440, 1450, 1460, 1470, 1480, 1490, 1500, 950,
960, 980, 990, 1030, 1070, 1080, 1090, 1100, 1120, 1130, 1140, 1160, 1170, 1190, 1200,
1210, 1220, 1230, 1240, 1250, 1260, 1270, 1280, 1290, 1300, 1310, 1340, 1350, 1360,
1370, 1380, 1390, 1400, 1410, 1420, 1430, 1440, 1450, 1460, 1470, 1480, 1490, 1500,
970, 990, 1010, 1050, 1060, 1080, 1090, 1120, 1140, 1160, 1170, 1180, 1190, 1200,
1210, 1220, 1230, 1240, 1270, 1290, 1300, 1310, 1320, 1330, 1360, 1390, 1400, 1410,
1420, 1430, 1440, 1450, 1460, 1470, 1480, 1490, 1500, 940, 1000, 1010, 1020, 1040,
1060, 1070, 1080, 1100, 1110, 1120, 1130, 1140, 1160, 1170, 1180, 1190, 1200, 1210,
1220, 1230, 1240, 1250, 1260, 1270, 1280, 1290, 1300, 1320, 1330, 1340, 1350, 1360,
1370, 1380, 1390, 1400, 1410, 1420, 1430, 1440, 1450, 1460, 1470, 1480, 1490, 1500,
850, 1030, 1070, 1100, 1120, 1140, 1150, 1160, 1190, 1200, 1210, 1220, 1230, 1280,
1290, 1300, 1320, 1330, 1340, 1360, 1400, 1410, 1420, 1430, 1450, 1460, 1470, 1480,
1500, 1160, 1200, 1210, 1230, 1270, 1310, 1320, 1330, 1340, 1360, 1370, 1390, 1400,
1410, 1450, 1460, 1480, 1490, 1500, 1040, 1090, 1150, 1180, 1190, 1220, 1280, 1290,
1320, 1360, 1370, 1380, 1390, 1400, 1410, 1440, 1460, 1470, 1480, 1490, 1500, 1160,
1280, 1300, 1310, 1320, 1350, 1360, 1370, 1390, 1410, 1420, 1450, 1460, 1470, 1480,
1490, 930, 1210, 1220, 1250, 1370, 1400, 1410, 1430, 1450, 1470, 1480, 1490, 1110,
1180, 1190, 1200, 1290, 1370, 1400, 1410, 1420, 1430, 1440, 1450, 1460, 1470, 1480,
1130, 1410, 1460, 1480, 1490, 1130, 1150, 1230, 1260, 1310, 1360, 1370, 1390, 1400,
1410, 1430, 1450, 1460, 1490, 1500, 1340, 1360, 1480, 1470, 1490, 1340, 1400, 1500,
1500, 1470, 1500, 1410, 1450, 1390 };
        Integer[] strata = { 5, 5, 5, 5, 5, 5, 5, 5, 10, 10, 10, 10, 10, 10, 10,
10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10,
10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10,
10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10,
10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10,
10, 10, 10, 10, 10, 10, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20,
20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20,
20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20,
20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20,
```

20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 25, 30, 35, 40, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 55, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 70, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 75, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85,

```
85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85,
85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85,
85, 85, 85, 85, 85, 85, 90, 90, 90, 90, 90, 90, 90, 90, 90, 90, 90, 90, 90, 90, 90, 90, 90, 90, 90, 90, 90,
90, 90, 90, 90, 90, 90, 90, 90, 90, 90, 90, 90, 90, 90, 90, 90, 90, 90, 90, 90, 90, 90, 90, 90, 90, 90, 90,
90, 90, 90, 90, 90, 90, 90, 90, 90, 90, 95, 95, 95, 95, 95, 95, 95, 95, 95, 95, 95, 95, 95, 95, 95, 95, 95,
95, 95, 95, 95, 95, 95, 95, 95, 95, 95, 95, 95, 95, 95, 95, 95, 95, 95, 95, 95, 95, 95, 95, 95, 95, 95, 95,
95, 95, 95, 95, 95, 95, 95, 95, 95, 95, 95, 95, 95, 95, 95, 95, 95, 95, 95, 95, 95, 100, 100, 100,
100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100,
100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100,
100, 100, 100, 100, 100, 105, 105, 105, 105, 105, 105, 105, 105, 105, 105, 105, 105, 105, 105, 105, 105,
105, 105, 105, 105, 105, 105, 105, 105, 105, 105, 105, 105, 105, 105, 105, 105, 105, 105, 105, 105, 105,
105, 105, 105, 105, 105, 105, 110, 110, 110, 110, 110, 110, 110, 110, 110, 110, 110, 110, 110,
110, 110, 110, 110, 110, 110, 110, 110, 110, 110, 110, 110, 110, 110, 110, 110, 110, 110, 110, 110, 110,
110, 110, 110, 110, 110, 110, 110, 110, 110, 110, 110, 110, 110, 110, 110, 110, 110, 110, 115,
115, 115, 115, 115, 115, 115, 115, 115, 115, 115, 115, 115, 115, 115, 115, 115, 115, 115, 115, 115, 115,
115, 115, 115, 115, 115, 115, 115, 115, 115, 115, 120, 120, 120, 120, 120, 120, 120, 120, 120,
120, 120, 120, 120, 120, 120, 120, 120, 120, 120, 120, 125, 125, 125, 125, 125, 125, 125, 125,
125, 125, 125, 125, 125, 125, 125, 125, 125, 125, 125, 125, 125, 125, 130, 130, 130, 130,
130, 130, 130, 130, 130, 130, 130, 130, 130, 130, 130, 130, 135, 135, 135, 135, 135, 135, 135,
135, 135, 135, 135, 135, 135, 140, 140, 140, 140, 140, 140, 140, 140, 140, 140, 140, 140, 140,
140, 140, 140, 145, 145, 145, 145, 145, 150, 150, 150, 150, 150, 150, 150, 150, 150, 150, 150,
150, 150, 150, 150, 150, 155, 155, 155, 160, 165, 170, 170, 170, 175, 180, 185, 210, 215,
245 };
                        for(i=0;i<window.length;i++){
                                for(int model=0;model<=3;model++){
                                        for(int                         preProcess=0;preProcess<=0;
preProcess++){
                                                ArrayList<Integer>      temp    =       new
ArrayList<Integer>();
                                                temp.add(strata[i]);
                                                temp.add(window[i]);
                                                temp.add(model);
                                                temp.add(preProcess);
                                                ordersToBeSent.add(new
ArrayList<Integer>(temp));
                                        }
                                }

                        }

                        System.out.println("starting master loop");

                        while(ordersToBeSent.size()>0 || ordersOut.size()>0){

                                while(SCH.Sockets.size()>0){
                                        System.out.println("master adding socket");
```

```
                        sockets.add(SCH.Sockets.remove());
                    }


                    //check sockets

                    Iterator<Socket> SocketItr = sockets.iterator();
                    while(SocketItr.hasNext()){
                        Socket s = SocketItr.next();
                        BufferedReader in = null;
                        PrintWriter out = null;
                        try{
                            out = new PrintWriter(s.getOutputStream(),
true); //make write
                            in = new BufferedReader( //make reader
                                    new InputStreamReader(

        s.getInputStream()));


                            while(in.ready()){ //while socket has
something to say
                                System.out.println("we got a
message");
                                String msg = in.readLine(); //get the
return message
                                if(msg.contains("sendTask")){
                                    System.out.println(msg);
                                    System.out.println("sending
order");
                                    if(ordersToBeSent.size()>0){
                                        ArrayList<Integer>
AI = ordersToBeSent.get(0);

        ordersToBeSent.remove(0);
                                        ordersOut.add(new
ArrayList<Integer>(AI));

        System.out.println("sent Order: Strata: " + AI.get(0) + " window: " + AI.get(1) + "
Model: " + AI.get(2) + " PreProcess: "  + AI.get(3));
                                        out.println("order:"  +
AI.get(0) + ":" + AI.get(1) + ":" + AI.get(2) + ":" + AI.get(3));
                                    }else if(ordersOut.size()>0){
                                        Random generator =
new Random();
```

```
                                                        ArrayList<Integer>
AI = ordersOut.get(generator.nextInt( ordersOut.size()));

        System.out.println("re-sent Order: Strata: " + AI.get(0) + " window: " + AI.get(1)
+ " Model: " + AI.get(2) + " PreProcess: "  + AI.get(3) );
                                                        out.println("re-order:"
+ AI.get(0) + ":" + AI.get(1) + ":" + AI.get(2) + ":" + AI.get(3));
                                                    }else{
                                                        out.println("none");

                                                    }
                                                    System.out.println("we  still
have " + ordersOut.size() + " out and "+ ordersToBeSent.size() + " orders left to be
sent");
                                                    System.out.println("error  =  "
+ error + " correct = " +correct);
                                                }
                                                if(msg.contains("return")){   //TODO
deal with return
                                                    System.out.println("value
incoming : " + msg);
                                                    double d = 0.0;
                                                    int          incomingStrata=0,
incomingWindow=0, incomingModel=0, incomingPreprocessing=0;
                                                    String[] res = msg.split(":");
//return:double:int:int
                                                    d                          =
Double.parseDouble(res[1]);
                                                    if(d>0.0){
                                                        incomingStrata      =
Integer.parseInt(res[2]); //strata
                                                        incomingWindow    =
Integer.parseInt(res[3]); //window
                                                        incomingModel      =
Integer.parseInt(res[4]); //model

        incomingPreprocessing = Integer.parseInt(res[5]); //preprocess
        //                                              int    index    =    ((I-
STRATA_START)/STRATA_GAP);
        //                                              //index *= 100;
        //                                              index                 *=
((WINDOW_MAX-WINDOW_START)/WINDOW_GAP);
        //                                              index      +=      ((K-
WINDOW_START)/WINDOW_GAP);
```

```
        //
        System.out.println("returned: Strata: " + I + " window: " + K + " resultIndex: " +
index + " result: " + d);
        //
        if(d>results.get(index).get(0)){
        //
        results.get(index).set(0, d); //set acc
        //
        results.get(index).set(1, Double.parseDouble(res[2])); //set strata
        //
        results.get(index).set(2, Double.parseDouble(res[3])); //set window
        //
        results.get(index).set(3, 0.0); //set algorithm type
        //                                                              }
        //                                                              String  tempWrite[]  =
msg.split(":");
        //                                                              try {
        //                                                                      //
                                                BufferedWriter  csvout  =  new
BufferedWriter(new FileWriter(new File("M:\\Ex2_csv\\endresults.csv")));
        //
        BufferedWriter  csvout  =  new  BufferedWriter(new  FileWriter(new
File("endresults.csv")));
        //
        for(ArrayList<Double> AD : results){
        //                                                                              int
countt =0;
        //
        for(Double dd : AD){
        //
        if(count ==0){
        //
        csvout.write(dd + ", ");
        //
        countt++;
        //
        }else{
        //
        csvout.write(dd.intValue() + ", ");
        //
        }
        //                                                                              }
        //
        csvout.write("\n");
        //                                                                      }
        //                                                                      csvout.close();
```

```
        //                                                          } catch (IOException
e) {
        //
        System.out.println("shit the output failed. we cant make the csv!");
        //
        e.printStackTrace();
        //                                                          }
                                                                    //TODO add the result
to the array list of results
        //                                                          returns.add(res[1]+
"," + res[2] + "," + res[3] + "," + res[4] + "," + res[5]);

        Iterator<ArrayList<Integer>> itr = ordersOut.iterator();
                                                                    while(itr.hasNext()){

        ArrayList<Integer> AI = itr.next();

        if(AI.get(0)==incomingStrata && AI.get(1)==incomingWindow &&
AI.get(2)==incomingModel && AI.get(3)==incomingPreprocessing){

        System.out.println("removed: Strata: " + AI.get(0) + " Window: " + AI.get(1) + "
model: " + AI.get(2) + " preProcessing: " + AI.get(3));

        itr.remove();

        returns.add(res[1]+ "," + res[2] + "," + res[3] + "," + res[4] + "," + res[5]);

        System.out.println("we still have " + ordersOut.size() + " out and "+
ordersToBeSent.size() + " orders left to be sent. " );

        System.out.println("error = " + error + " correct = " +correct);
                                                                                    break;
                                                                            }
                                                                    }

                                                                }else{

        System.out.println("we had an error returned");
                                                                }
                                                            }
                                                    }
                                                } catch (IOException e) { // our socket is dead
                                                    SocketItr.remove();
                                                    System.out.println("a Socket is dead");
                                                    //
        e.printStackTrace();
```

```
                        }

                    }




                }
                System.out.println("master done");

                Iterator<Socket> SocketItr = sockets.iterator();
                while(SocketItr.hasNext()){
                    Socket s = SocketItr.next();
                    BufferedReader in = null;
                    PrintWriter out = null;
                    try{
                        out = new PrintWriter(s.getOutputStream(), true);
//make write
                        in = new BufferedReader( //make reader
                                new InputStreamReader(
                                        s.getInputStream()));

                        out.write("done");
                    }catch(Exception e){

                    }
                }

                try {
                    //                        BufferedWriter csvout = new
BufferedWriter(new FileWriter(new File("M:\\Ex2_csv\\endresults.csv")));
                    BufferedWriter csvout = new BufferedWriter(new
FileWriter(new File("endresults.csv")));
                    for(ArrayList<Double> AD : results){
                        for(Double dd : AD){
                            csvout.write(dd + ",");
                        }
                        csvout.write("\n");
                    }
                    csvout.close();
                } catch (IOException e) {
                    System.out.println("shit the output failed. we cant make the
csv!");
                    e.printStackTrace();
                }
```

```
		}

		public static void slave(String loc, int ThreadLimit){
			boolean getCmd = true;
			String computername = "defualt";
			try { //get computer name

	computername=InetAddress.getLocalHost().getHostName();
			} catch (UnknownHostException e) {
			}
			if(loc.toLowerCase().equals("linux")){ //set linux timer for getting
threads

			}
			long lastorder = 0;
			System.out.println("started slave");
			OperatingSystemMXBean OSMXB =
ManagementFactory.getOperatingSystemMXBean();
		//		ArrayList<ArrayList<Object>> results = new
ArrayList<ArrayList<Object>>(30);
			Socket server = null;
			BufferedReader in = null;
			PrintWriter out = null;
			String msg= "";
			System.out.println("starting connect");
			boolean nconnect = true;
			while(nconnect){
				try {

					server = new Socket("130.215.17.17", 2500);
					nconnect = false;
				} catch (UnknownHostException e) {
					// TODO Auto-generated catch block
					//				e.printStackTrace();
				} catch (IOException e) {
					e.printStackTrace();
					System.exit(0);
				}
			}
			System.out.println("connected");
			ArrayList<Thread> threads = new ArrayList<Thread>();
			ArrayList<MachineLearn> DTs = new
ArrayList<MachineLearn>();
			try{
				out = new PrintWriter(server.getOutputStream(), true);
```

```
                    in = new BufferedReader(
                                new InputStreamReader(
                                        server.getInputStream()));
                }catch(Exception e){
                    System.out.println("socket failed");
                    System.exit(0);
                }

                while(true){
                    if(!server.isConnected()){
                        System.exit(0);
                    }
                    try{
                        if(in.ready()){
                            msg = in.readLine();
                            if(msg.contains("done")){
                                System.exit(0);
                            }
                        }
                    }catch(Exception e){
                        System.exit(0);

                    }
                    for(int index=0;index<DTs.size();index++){//see if threads
done
                        MachineLearn dt = DTs.get(index);
                        if(dt.done){

                            out.println("return:" + dt.result + ":"  +
dt.strata + ":" + dt.window + ":" + dt.model + ":" + dt.preProcess + ":" + computername);
                            getCmd=true;
                            DTs.remove(index);
                            threads.remove(index);
                            System.out.println("ACTIVE THREADS:");
                            int thr = 0;
                            for(MachineLearn dtt : DTs){
                                System.out.println("Thread: " + thr +
" is " + dtt.strata + ", " + dtt.window + ", " + dt.model + ", " + dt.window);
                                thr++;
                            }
                            break;
                        }
                    }


                    try{ //test to get new load
```

```
                                        if(threads.size()<ThreadLimit && getCmd){
                                                int count =0;
                                                getTask: {
                                                        count++;
                                                        if(count>5){
                                                                getCmd=false;
                                                                System.out.println("we could
not get a command!");
                                                                break;
                                                        }


                                                        System.out.println("getting        new
task");
                                                        out.println("sendTask:"                +
computername); //TODO retrive task
                                                        while(!in.ready());
                                                        String msg1 = in.readLine();

        if(msg1.toLowerCase().equals("none")){
                                                                System.out.println("noe more
comands!!");
                                                                getCmd=false;
                                                                break;
                                                        }
                                                        String[] goals = msg1.split(":");
                                                        for(MachineLearn dt : DTs){

        if(dt.strata==Integer.parseInt(goals[1])                                        &&
dt.window==Integer.parseInt(goals[2])  &&  dt.model==Integer.parseInt(goals[3])  &&
dt.preProcess==Integer.parseInt(goals[4])){
                                                                        getCmd=false;
                                                                        break getTask;
                                                                }
                                                        }

                                                        if(getCmd){
                                                                System.out.println("started
new cmd");
                                                                DTs.add(new
MachineLearn(Integer.parseInt(goals[1]),                    Integer.parseInt(goals[2]),
Integer.parseInt(goals[3]), Integer.parseInt(goals[4]), loc));
                                                                threads.add(new
Thread(DTs.get(DTs.size()-1)));
```

```
                                                        threads.get(threads.size()-
1).start();
                                                        System.out.println("ACTIVE
THREADS:");
                                                        int thr = 0;
                                                        for(MachineLearn dt : DTs){

        System.out.println("Thread: " + thr + " is " + dt.strata + ", " + dt.window + ", " +
dt.model + ", " + dt.preProcess);
                                                                thr++;
                                                        }
                                                }

                                        }
                                        System.out.println("get cmd block exited");
                                }
                        }catch(Exception e){
                                System.out.println("socket failed");
                                System.exit(0);
                        }



                }
        }



        }


        import java.io.BufferedReader;
        import java.io.IOException;
        import java.io.InputStreamReader;
        import java.io.PrintWriter;
        import java.net.*;
        import java.util.ArrayList;
        import java.util.concurrent.ArrayBlockingQueue;




        public class ServerConnectionHandler implements Runnable {
                volatile public ArrayBlockingQueue<Socket> Sockets;
                private ServerSocket serverSocket;
```

```
		public ServerConnectionHandler() {
			Sockets = new ArrayBlockingQueue<Socket>(100);
			try {
				serverSocket = new ServerSocket(2500);
			} catch (IOException e) {
				System.out.println("server socket failed");
				e.printStackTrace();
			}
		}

		void getConnection(){
			Socket clientSocket = null;
			try {
				System.out.println("ready to accept");
				clientSocket = serverSocket.accept();
				Sockets.add(clientSocket);
	//			out = new PrintWriter(clientSocket.getOutputStream(),
true);
	//			in = new BufferedReader(
	//					new InputStreamReader(
	//
	clientSocket.getInputStream()));

			} catch (Exception e) {
	//			System.out.println("Accept failed. one of our lackeys is an
idiot");
				//System.exit(1);
			}
		}

		@Override
		public void run() {
			InetAddress ip;
			  try {

				ip = InetAddress.getLocalHost();
				System.out.println("Current IP address : " +
ip.getHostAddress());

			  } catch (UnknownHostException e) {

				e.printStackTrace();

			  }
			while(true){
				System.out.println("checking for socket");
```

```
                        getConnection();
                }

        }

}

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.lang.reflect.Array;
import java.net.InetAddress;
import java.net.UnknownHostException;
import java.util.ArrayList;
import java.util.Random;
import java.util.concurrent.atomic.AtomicBoolean;

import weka.classifiers.Evaluation;
import weka.classifiers.trees.J48;
import weka.core.Instances;
import weka.core.converters.CSVLoader;


public class MachineLearn implements Runnable{
        private Instances ins;
        public volatile boolean done;
        public volatile boolean reading;
        public volatile int strata;
        public volatile int window;
        public volatile int model;
        public volatile int preProcess;
        public volatile double result;
        private String name;
        private String location;

        public MachineLearn(int st, int win, int mod, int pp, String loc) {
                location = loc;
                strata = st;
                window = win;
                //                  ins=in;
                done = false;
                result=-1.0;
```

```
                reading = false;
                name = st + ", " + win + ":";
        }

        @Override
        public void run() {
                reading = true;
                System.out.println(name + "makeing instances");
                Instances ins = makeSTRA(strata, window, window/2, preProcess,
location);
                System.out.println(name + "Insances done");
                reading = false;
                System.out.println(name + "starting classifier");
                result=DTmethod(ins);
                System.out.println(name + "done");
                done = true;


        }

        public double DTmethod(Instances DATA){
                try{
                        //              Instances DATA = ins;
                        DATA.setClassIndex(DATA.numAttributes()-1);
                        //              System.out.println(DATA.numInstances());
                        System.out.println(name + " making tree");
                        J48 DT = new J48();

                        DT.setNumFolds(10);
                        try {
                                DT.buildClassifier(DATA);
                        } catch (Exception e) {
                                System.out.println("tree crash");
                                e.printStackTrace();
                        }
                        System.out.println(name + "starting cross folds");
                        Evaluation eval = new Evaluation(DATA);
                        eval.crossValidateModel(DT,     DATA,     10,     new
Random(1234523465));

                        return eval.correct()/(eval.correct()+eval.incorrect());
```

```
                }catch(Exception e){
                        System.out.println("failed DT!!");

                }
                return -1.0;
        }

        public Instances makeSTRA(int Astrata, int Awindow, int Aoffset, int pp,
String location){
                String outPath = "M:\\Ex2_csv\\";
                String inPath = "M:\\Experiment2\\";
                String csvPath = "C:\\Users\\bleone\\Documents\\";
                String computername = "defualt";
                try {

        computername=InetAddress.getLocalHost().getHostName();
                } catch (UnknownHostException e) {
                }
                if(location.toLowerCase().equals("ece")){
                        outPath = "M:\\Ex2_csv\\";
                        inPath = "M:\\Experiment2\\";


                        new File("C:\\temp\\").mkdirs();
                        csvPath = "C:\\temp\\";
                        if(computername.toLowerCase().contains("hertz")){
                                csvPath = "M:\\Ex2_csv\\";
                        }
                }
                if(location.toLowerCase().equals("linux")){
                        //                      outPath = "~/Ex2_csv/";
                        //                      inPath = "~/Experiment2/";
                        outPath = "Ex2_csv/";
                        inPath = "Experiment2/";
                        csvPath = "/tmp/";
                        if(computername.toLowerCase().contains("hpc")){
                                System.out.println("WE     are     on     an     HPC
computer!!");
                                csvPath = "Ex2_csv/";
                        }
                }

                if(location.toLowerCase().equals("ccc")){
                        //set ccc paths
                }
```

```
//		String outPath = "";
//		String inPath = "";
BufferedReader read = null;
BufferedWriter writer = null;
File folder = new File(inPath);
File[] listOfFiles = folder.listFiles();
ArrayList<ArrayList<Double>> queues = null;
ArrayList<ArrayList<String>> buffer = null;
double min = 0, max =0;
ArrayList<Double> stratas = null;
ArrayList<ArrayList<ArrayList<Double>>> RMS = new ArrayList<ArrayList<ArrayList<Double>>>(0);
ArrayList<ArrayList<ArrayList<Double>>> Data_by_Row = new ArrayList<ArrayList<ArrayList<Double>>>(0);
ArrayList<ArrayList<ArrayList<Double>>> Data_by_Column = new ArrayList<ArrayList<ArrayList<Double>>>();
ArrayList<ArrayList<ArrayList<ArrayList<Integer>>>> STRATA = new ArrayList<ArrayList<ArrayList<ArrayList<Integer>>>>(0);
ArrayList<String> Classification = new ArrayList<String>(0);




//		try {
//			buffer = new ArrayList<ArrayList<String>>(0);
//			writer = new BufferedWriter(new FileWriter(outPath));
//
	writer.write("sensor1[RMS],sensor2[RMS],sensor3[RMS],sensor4[RMS],classification\n");
//		} catch (IOException e1) {
//			e1.printStackTrace();
//		}

//make strata
for(File file : listOfFiles){
	if(file.isFile()){
		try{
			read = new BufferedReader(new FileReader(inPath + file.getName()));
			while(read.ready()){
				String in = read.readLine();
				String[] inSplit = in.split("\\s+");
				for(String s: inSplit){
					Double d = null;
```

```
                                                try{
                                                        d                       =
Double.parseDouble(s);
                                                }catch(Exception e){
                                                        continue;
                                                }
                                                if(d<min){
                                                        min = d;
                                                }
                                                if(d>max){
                                                        max=d;
                                                }
                                        }
                                }
                        }catch(Exception e){
                                //TODO nothing
                        }
                }

        }
        //here we have min and max
        //System.out.println("min=" + min +" max=" + max);
        //min
        double gap = (max-min)/(double)Astrata;
        //System.out.println("Strata=" + (max-min)/Astrata);
        //System.exit(0);
        stratas = new ArrayList<Double>();
        stratas.add(min-gap*25);
        stratas.add(min);
        for(int i=1;i<Astrata;i++){
                stratas.add(stratas.get(i)+gap);
        }
        stratas.add(max+gap*25);



        for (File file : listOfFiles) {
                if (file.isFile()) {
                        int index =0;
                        //                      ArrayList<String>   temp   =
new ArrayList<String>();
                        ArrayList<ArrayList<Double>> RMS_file = new
ArrayList<ArrayList<Double>>(0);
                        ArrayList<ArrayList<Double>> RAW_file_by_row
= new ArrayList<ArrayList<Double>>(0);
```

```
                                        ArrayList<ArrayList<Double>>
RAW_file_by_column = new ArrayList<ArrayList<Double>>(0);
                                        for(int count=0;count <5;count++){
                                                RAW_file_by_column.add(new
ArrayList<Double>(0));
                                        }
                                        ArrayList<ArrayList<Integer>> STRATA_FILE =
new ArrayList<ArrayList<Integer>>(0);
                                        queues = new ArrayList<ArrayList<Double>>(0);
                                        String csv = file.getName();
                                        //System.out.println("starting " + file.getName());
                                        csv = csv.split(".txt")[0] + ".csv";


                                        try {
                                                read = new BufferedReader(new
FileReader(inPath + file.getName()));
                                        } catch (FileNotFoundException e1) {
                                                // TODO Auto-generated catch block
                                                e1.printStackTrace();
                                        }

                                        if(csv.contains("LWR_REST"))
                                                Classification.add("LWR_REST");
                                        //index = 1;
                                        //in += "1\n";
                                        if(csv.contains("ELB_FLEX_A"))
                                                Classification.add("ELB_FLEX_A");
                                        //                              index=0;
                                        //in += "0\n";
                                        if(csv.contains("ELB_FLEX_B"))
                                                Classification.add("ELB_FLEX_B");
                                        //                              index=2;
                                        //                              in += "2\n";
                                        if(csv.contains("ELB_FLEX_C"))
                                                Classification.add("ELB_FLEX_C");
                                        //                              index=3;
                                        //                              in += "3\n";
                                        if(csv.contains("ELB_EXT_A"))
                                                Classification.add("ELB_EXT_A");
                                        //                              index=4;
                                        //                              in += "4\n";
                                        if(csv.contains("ELB_EXT_B"))
                                                Classification.add("ELB_EXT_B");
                                        //                              index=5;
                                        //                              in += "5\n";
```

```
                                        if(csv.contains("ELB_EXT_C"))
                                                Classification.add("ELB_EXT_C");
                                        //                              index=6;
                                        //                              in += "6\n";
                                        if(csv.contains("ELB_NO_LOAD"))
                                                Classification.add("ELB_NO_LOAD");

                                        try {
                                                while(read.ready()){
                                                        String in = read.readLine();
                                                        String[] inSplit = in.split("\\s+");
                                                        if(inSplit.length<5){
                                                                //System.out.println(in);
                                                                //System.out.println("yeah!");
                                                                continue;
                                                        }

                                                        in = "";



                                                        ArrayList<Double>    tempRAW    =
new ArrayList<Double>(0);
                                                        for(String s : inSplit){

        tempRAW.add(Double.parseDouble(s));
                                                        }

                                                        RAW_file_by_row.add(new
ArrayList<Double>(tempRAW));

                                                        for(int i=0;i<5;i++){

        RAW_file_by_column.get(i).add(Double.parseDouble(inSplit[i]));
                                                        }




                                                }
                                                read.close();
                                                Data_by_Row.add(new
ArrayList<ArrayList<Double>>(RAW_file_by_row));
```

```
                                        Data_by_Column.add(new
ArrayList<ArrayList<Double>>(RAW_file_by_column));
                                    } catch (IOException e) {
                                        // TODO Auto-generated catch block
                                        e.printStackTrace();
                                    }
                                }
                            }

                            //calc stratas
                            int windowSize=Awindow;
                            switch (pp) {
                            case 0: //no rectifiying, basic stratification over raw window vaules
                                //basic

                                //              System.out.println("RAW      =      "      +
RAW.get(0).size());
                                for(int
fileIndex=0;fileIndex<Data_by_Row.size();fileIndex++){ // idex through each raw file
                                    ArrayList<ArrayList<Double>>      rawFile      =
Data_by_Row.get(fileIndex);//load raw file
                                    ArrayList<ArrayList<ArrayList<Integer>>>
StrataFile = new ArrayList<ArrayList<ArrayList<Integer>>>(0); //make the related
startified file
                                    for(int      dataIndex=0;dataIndex<rawFile.size()-
windowSize;dataIndex=dataIndex+Aoffset){ // make a window of the data lines in the
file to make the strata from
                                        ArrayList<ArrayList<Integer>>
StrataDataLine = new ArrayList<ArrayList<Integer>>(0); // make stata data line for the
window of data
                                        for(int  k=0;k<5;k++){//pre  load  the  strata
arrays with zeros so we can count the data
                                            StrataDataLine.add(new
ArrayList<Integer>(0));
                                            for(int      i=1;i<stratas.size();i++){
//this aray is stratas.size()-1 long
                                                StrataDataLine.get(k).add(0);
                                            }
                                        }
                                        for(int
DI=dataIndex;DI<dataIndex+windowSize;DI++){
                                            for(int  data=0;data<5;data++){  //this
will strafify each part of the selected data line
                                                for(int
strataInd=1;strataInd<stratas.size();strataInd++){//strataind-1 refers to the actual strata we
want, data referes to the strata set we should be in
```

```
        if(stratas.get(strataInd-1)<rawFile.get(DI).get(data)                    &&
stratas.get(strataInd)>rawFile.get(DI).get(data)  ){

        StrataDataLine.get(data).set(strataInd-1,     StrataDataLine.get(data).get(strataInd-
1)+1);
                                                                                    break;
                                                                            }
                                                                    }
                                                            }
                                                    }
                                                    StrataFile.add(new
ArrayList<ArrayList<Integer>>(StrataDataLine));

                                            }
                                            STRATA.add(new
ArrayList<ArrayList<ArrayList<Integer>>>(StrataFile));
                                    }
                                    ArrayList<String> file = new ArrayList<String>();
                                    String header = "";
                                    String nheader = "";
                                    String dbname = "";
                                    String attrs = "";
                                    try {

                                            String dataFile = "";
                                            ArrayList<String>       Attributes      =       new
ArrayList<String>();
                                            //                                                  writer
= new BufferedWriter(new FileWriter(outPath + "strated_data_W" + Awindow +
"_STR" + Astrata +".db"));



                                            System.out.println("made file: " + csvPath +
"input_" + strata + "_" + windowSize + "_" + pp + "_" + model + ".csv");
                                            writer      =      new      BufferedWriter(new
FileWriter(csvPath + "input_" + strata + "_" + windowSize + "_" + pp + "_" + model +
".csv"));
                                            //TODO make strata headers
                                            //                          nheader+="strated_data_W"
+ Awindow + "_STR" + Astrata + "\n";
                                            //
        writer.write("strated_data_W" + Awindow + "_STR" + Astrata + "\n");
                                            int str = STRATA.get(0).get(0).get(0).size();
                                            int DL = STRATA.get(0).get(0).size();
```

```
                                        for(int i=0;i<DL;i++){
                                                for(int k=0;k<str;k++){
                                                        writer.write("DataLine_"    +    i    +
"_Strata_" + k + ",");
                                                        //
        header+="DataLine_" + i + "_Strata_" + k + " numerical ";
                                                        //
        Attributes.add("DataLine_" + i + "_Strata_" + k);
                                                }
                                        }
                                        writer.write("Classification\n");
                                        //                      header+="Classification
symbolic\n";
                                        for(int
index=0;index<STRATA.size();index++){//STRATA
                                                ArrayList<ArrayList<ArrayList<Integer>>>
AAAD = STRATA.get(index);
                                                for(ArrayList<ArrayList<Integer>>  AAD  :
AAAD){//STRATA FILE
                                                        dataFile = "";
                                                        for(ArrayList<Integer> AD : AAD){
//STRATA data line
                                                                for(Integer    i    :    AD){    //
indivual count

        writer.write(i.toString() + ",");
                                                                                //
                                dataFile += i.toString() + " ";
                                                                        }
                                                        }
                                                        writer.write(Classification.get(index)
+ "\n");
                                                        //
        dataFile += Classification.get(index) + "\n";
                                                        file.add(new String(dataFile));
                                                }
                                        }

                                        writer.close();
                                } catch (Exception e) {
                                        // TODO Auto-generated catch block
                                        e.printStackTrace();
                                }

                                break;
```

```
case 1: //rectify and normalize, then stratify over window


                    for(int fileIndex=0;fileIndex<Data_by_Column.size();fileIndex++){ // idex through each raw file
                        ArrayList<ArrayList<Double>> rawFile = Data_by_Column.get(fileIndex);//load raw file
                        ArrayList<ArrayList<ArrayList<Integer>>> StrataFile = new ArrayList<ArrayList<ArrayList<Integer>>>(0); //make the related startified file
                        for(int dataIndex=0;dataIndex<rawFile.get(0).size()-windowSize;dataIndex=dataIndex+Aoffset){ // make a window of the data lines in the file to make the strata from
                            //since this is by column we have to grab the first array list's length rather then the length of the set of array lists
                            //now get the window strips as separate arrays.
                            ArrayList<ArrayList<Double>> dataToBeNormalized = new ArrayList<ArrayList<Double>>();
                            for(ArrayList<Double> AL : rawFile){ //here we grab the column strips, we need to pull out the section we want from each and make the histograms
                                ArrayList<Double> strip = new ArrayList<Double>(); //here will hold the window/strip of data
                                for(int index = dataIndex; index<dataIndex+windowSize; index++){//here we grab each element from the main file to form our window
                                    strip.add(AL.get(index));
                                }
                                //process the data
                                dataToBeNormalized.add(new ArrayList<Double>(rectifyWindowSingle(strip)));

                            }
        //                  normalizeWindow(dataToBeNormalized); // normalze the rectified sets
                            //now we need to make a histogram set.

        StrataFile.add(histogramByColumnNorm(normalizeWindow(dataToBeNormalized))); // here we have our histogram
                        }
                        STRATA.add(new ArrayList<ArrayList<ArrayList<Integer>>>(StrataFile));
```

```
                }
                try{
                System.out.println("made file: " + csvPath + "input_" +
strata + "_" + windowSize + "_" + pp + "_" + model + ".csv");
                writer = new BufferedWriter(new FileWriter(csvPath +
"input_" + strata + "_" + windowSize + "_" + pp + "_" + model + ".csv"));
                int numberOfHistograms = STRATA.get(0).get(0).size();
//the number of histograms in a window
                int binsInHistograms =
STRATA.get(0).get(0).get(0).size(); //the number of bins in each histogram
                for(int noh=0;noh<numberOfHistograms;noh++){
                        for(int bih=0;bih<binsInHistograms;bih++){
                                writer.write("DataLine_" + noh + "_Strata_"
+ bih + ",");
                        }
                }
                writer.write("Classification\n"); //the header for the csv is
done

                for(int
fileIndex=0;fileIndex<STRATA.size();fileIndex++){
                        ArrayList<ArrayList<ArrayList<Integer>>>
currentFile = STRATA.get(fileIndex);
                        String classification = Classification.get(fileIndex);
                        for(ArrayList<ArrayList<Integer>> AAI :
currentFile){ //iterate through data sets
                                for(ArrayList<Integer> AI : AAI){ //iterate
through histograms
                                        for(Integer I : AI){ //iterate through
                                                writer.write(I.toString() +
",");
                                        }
                                }
                                writer.write(classification + "\n");
                                //TODO test code!!!
                                writer.close();
                                System.exit(0);
                                //TODO end test code
                        }
                }


                }catch (Exception e) {
                        System.out.println("error makeing csv for
instances");
                        System.exit(0);
```

```
                }


                break;


            default:
                System.exit(0);
                break;
            }









                //classifier
            try{
                CSVLoader csvload = new CSVLoader();
                csvload.setFile(new File(csvPath + "input_" + strata + "_"
+ windowSize + "_" + pp + "_" + model + ".csv"));
                Instances DATA = csvload.getDataSet();
                DATA.setClassIndex(DATA.numAttributes()-1);
                File test = new File(csvPath + "input_" + strata + "_" +
windowSize + "_" + pp + "_" + model + ".csv");
                test.delete();
                return DATA;
            }catch(Exception e){
                System.exit(0);
            }









            System.out.println("Done!");
            return null;
        }

        /**
         * rectifies a signal set over a window. signal formated as:
         * (0) (0-...) <-- instance 0
         * (1) (0-...) <-- instance 1
         *
```

```
	 * @param input
	 * @return
	 */
	private ArrayList<ArrayList<Double>> rectifyWindow(ArrayList<ArrayList<Double>> input){
		ArrayList<ArrayList<Double>> Input = new ArrayList<ArrayList<Double>>(input);
		for(int outerIndex = 0; outerIndex<Input.size(); outerIndex++){ // we rectify one column at a time
			for(int innerIndex=0; innerIndex<Input.get(outerIndex).size(); innerIndex++){ // first find max in column
				Input.get(outerIndex).set(innerIndex, Math.abs(Input.get(outerIndex).get(innerIndex)));
			}
		}
		return Input;
	}

	private ArrayList<Double> rectifyWindowSingle(ArrayList<Double> input){
		ArrayList<Double> Input = new ArrayList<Double>(input);
		for(Double d : input){
			Input.add(Math.abs(d));
		}
		return new ArrayList<Double>(Input);
	}

	/**
	 * this method takes data as such that the inner arrlists are instances and the outer one is the data set over time. so:
	 * (0) (0-...) <-- instance 0
	 * (1) (0-...) <-- instance 1
	 *
	 * this method with normalize the instances to 0 .. 1 range.
	 * @param Input the unnormnalized instances
	 * @return the normalized instances
	 */
	private ArrayList<ArrayList<Double>> normalizeWindow(ArrayList<ArrayList<Double>> input){
		ArrayList<ArrayList<Double>> Input = new ArrayList<ArrayList<Double>>(input);
		//int streams = Input.get(0).size();
		double max = 0; // here we will store the max value we found
		for(int outerIndex = 0; outerIndex<Input.size(); outerIndex++){ // we normalizes one column at a time
			max=Input.get(outerIndex).get(0);
```

```
                        for(int innerIndex=0;
innerIndex<Input.get(outerIndex).size(); innerIndex++){ // first find max in column
                                if(max<Input.get(outerIndex).get(innerIndex)){    //
found a new max
                                        max =
Input.get(outerIndex).get(innerIndex); // store it
                                }
                        }
                        for(outerIndex = 0; outerIndex<Input.size();
outerIndex++){ //got thorugh values to normalize
                                for(int innerIndex=0;
innerIndex<Input.get(outerIndex).size(); innerIndex++){ // now we normalize
                                        Input.get(outerIndex).set(innerIndex,
Input.get(outerIndex).get(innerIndex)/max);
                                }
                        }
                }
                return Input;
        }

        private ArrayList<ArrayList<Integer>>
histogramByColumnNorm(ArrayList<ArrayList<Double>> input){
                ArrayList<ArrayList<Integer>> histograms = new
ArrayList<ArrayList<Integer>>();
                ArrayList<Double> bins = new ArrayList<Double>();
                bins.add(0.0);
                double currentBin=0.0, gap=1/strata;
                for(int i=0;i<strata-1;i++){
                        currentBin+=gap;
                        bins.add(currentBin);
                }
                bins.add(1.0);
                for(ArrayList<Double> AD : input){//make hists and set to zero
                        ArrayList<Integer> hist = new ArrayList<Integer>();
                        for(int i=0;i<strata;i++)//zero the histogram
                                hist.add(0);
                        for(double d : AD){//place each double in a bin
                                for(int i=1;i<bins.size();i++){//search all of the bins
                                        if(d>=bins.get(i-1) &&
d<=bins.get(i)){//check the bin we are at
                                                hist.set(i-1, hist.get(i-1)+1);
                                                break;
                                        }
                                }
                        }
                        histograms.add(new ArrayList<Integer>(hist));
```

```
                }
                return new ArrayList<ArrayList<Integer>>(histograms);
        }




}
```

#.#.# Rasbpery Pi test code for Ethernet mbed testing

```
/* A simple server in the internet domain using TCP
   The port number is passed as an argument */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <termios.h> // terminal io (serial port) interface
#include <fcntl.h>  // File control definitions
#include <errno.h>  // Error number definitions
#include <assert.h>
#include <stdio.h>
#include <string.h>
#include <pthread.h>
#include <semaphore.h>
#include <stdlib.h>
#include <unistd.h>


#define TRUE        1
#define FALSE       0
#define EXIT       -1
#define BUFFSIZE    4096
#define R2D 57.295827909f;

 typedef int ComPortHandle;
 typedef unsigned char Byte;
 FILE * fd;
 FILE * MBED;
 struct timeval start, end;
 long mtime, seconds, useconds;
```

```
int BytesRecived;
char Data[256];
int sockfd, newsockfd;
socklen_t clilen;
struct sockaddr_in serv_addr, cli_addr;
int loop = 1;
char reading[1000][255];

void getConnection(void);
void waitForData(void);
void processInput();
void stream();

void error(const char *msg)
{
   perror(msg);
   exit(1);
}


 //my code
float pBytesToFloat(char * bytes){
  float x = 0;
  union f {float r; unsigned char b[4];} f;
  // f.b[0]=bytes[0];
  // f.b[1]=bytes[1];
  // f.b[2]=bytes[2];
  // f.b[3]=bytes[3];
  f.b[0]=bytes[3];
  f.b[1]=bytes[2];
  f.b[2]=bytes[1];
  f.b[3]=bytes[0];
  //printf("\n0x%x,  0x%x  0x%x  0x%x\n",  (unsigned  int)(unsigned
char)bytes[0],(unsigned  int)(unsigned  char)bytes[1],(unsigned  int)(unsigned
char)bytes[2],(unsigned int)(unsigned char)bytes[3]);
  return f.r;

}

//function to remove whitespace from commands
 void remove_space(char *string_with_spaces){

   int i=0;
   int parse_index = 0;
   char *no_space_string;
```

```
  if(string_with_spaces==0)
   return;

 no_space_string=string_with_spaces;

 for(i;i<strlen(string_with_spaces);i++){
   if(!(string_with_spaces[i]==' '||string_with_spaces[i]=='\t'))
    *no_space_string++ = string_with_spaces[i];
 }

 *no_space_string = '\0';

 no_space_string = 0;

 return;
}

// Utility functions for working with a com port in Linux

// Purge
// Clears the com port's read and write buffers

int Purge(ComPortHandle comPortHandle){

 if (tcflush(comPortHandle,TCIOFLUSH)==-1){

  printf("flush failed\n");
  return FALSE;

}

return TRUE;

}

// OpenComPort
// Opens a com port with the correct settings for communicating with a
// MicroStrain 3DM-GX3-25 sensor

ComPortHandle OpenComPort(const char* comPortPath){

 ComPortHandle comPort = open(comPortPath, O_RDWR | O_NOCTTY);

 if (comPort== -1){ //Opening of port failed

  printf("Unable to open com Port %s\n Errno = %i\n", comPortPath, errno);
```

```
 return -1;

}

 //Get the current options for the port...
struct termios options;
tcgetattr(comPort, &options);

 //set the baud rate to 115200
int baudRate = B9600;
cfsetospeed(&options, baudRate);
cfsetispeed(&options, baudRate);

 //set the number of data bits.
 options.c_cflag &= ~CSIZE; // Mask the character size bits
 options.c_cflag |= CS8;

 //set the number of stop bits to 1
 options.c_cflag &= ~CSTOPB;

 //Set parity to None
 options.c_cflag &=~PARENB;

 //set for non-canonical (raw processing, no echo, etc.)
 options.c_iflag = IGNPAR; // ignore parity check close_port(int
 options.c_oflag = 0; // raw output
 options.c_lflag = 0; // raw input

 //Time-Outs -- won't work with NDELAY option in the call to open
 options.c_cc[VMIN] = 0;  // block reading until RX x characers. If x = 0,
            // it is non-blocking.
 options.c_cc[VTIME] = 1;  // Inter-Character Timer -- i.e. timeout= x*.1 s

 //Set local mode and enable the receiver
 options.c_cflag |= (CLOCAL | CREAD);

 //Purge serial port buffers
 Purge(comPort);

 //Set the new options for the port...
 int status=tcsetattr(comPort, TCSANOW, &options);

 if (status != 0){ //For error message

  printf("Configuring comport failed\n");
  return status;
```

```
}

 //Purge serial port buffers
Purge(comPort);

return comPort;

}

// CloseComPort
// Closes a port that was previously opened with OpenComPort
void CloseComPort(ComPortHandle comPort){

 close(comPort);

}

// readComPort
// read the specivied number of bytes from the com port
int readComPort(ComPortHandle comPort, Byte* bytes, int bytesToRead){

 int bytesRead = read(comPort, bytes, bytesToRead);
 return bytesRead;

}

// writeComPort
// send bytes to the com port
int writeComPort(ComPortHandle comPort, unsigned char* bytesToWrite, int
size){

 return write(comPort, bytesToWrite, size);

}

// Simple Linux Console interface function

//attemots to read BUFFSIZE bytes at a time from the open port
int readPort(ComPortHandle comPort, int i){

 int size;
 struct termios initial_settings, new_settings;
 char command_string[255];
 char c;
 int ret;
```

```
 Byte response[255] = {0};

 //get current terminal settings
 tcgetattr(0,&initial_settings);
 new_settings = initial_settings;
 new_settings.c_lflag &= ~ICANON;
 new_settings.c_cc[VMIN] = 0;
 new_settings.c_cc[VTIME] = 0;
 tcsetattr(0, TCSANOW, &new_settings);
 size=0;
 int current = 1;
 //determine if there are bytes to read
 while(current>0){
    current = readComPort(comPort, &reading[i][size], 255);
    size += current;
}
printf("%d, ", size);

 if(size<0)
  return FALSE;

 //if ther are bytes to read output them and check again
int resp;
for( resp=0; resp<size ;resp++){
  printf(" %d:0x%x", resp, response[i]);
}
printf("\n");
tcsetattr(0, TCSANOW, &initial_settings);

return TRUE;

}

//scandev
//finds attached microstrain devices and prompts user for choice then returns
//selected portname
char* scandev(){

 FILE *instream;
 char devnames[255][255];//allows for up to 256 devices with path links up to
           //255 characters long each
 int devct=0; //counter for number of devices
 int i=0;
 int j=0;
 int userchoice=0;
 char *device;
```

```
        //search /dev/serial for microstrain devices
        char *command = "find /dev/serial -print | grep -i mbed";

        printf("Searching for devices...\n");

        instream=popen(command, "r");//execute piped command in read mode

        if(!instream){//SOMETHING WRONG WITH THE SYSTEM COMMAND
PIPE...EXITING
         printf("ERROR BROKEN PIPELINE %s\n", command);
         return device;
        }

        //load char array of device addresses
        for(i=0;i<255&&(fgets(devnames[i],sizeof(devnames[i]), instream));i++){
         ++devct;
        }

        for(i=0;i<devct;i++){
         for(j=0;j<sizeof(devnames[i]);j++){
          if(devnames[i][j]=='\n'){
           devnames[i][j]='\0';//replaces newline inserted by pipe reader with
                   //char array terminator character
           break;//breaks loop after replacement
          }
        }
        printf("Device Found:\n%d: %s\n",i,devnames[i]);
        }

        //CHOOSE DEVICE TO CONNECT TO AND CONNECT TO IT (IF THERE
ARE ANY CONNECTED)

        if(devct>0){
         printf("Number of devices = %d\n", devct);
         if(devct>1){
          printf(
           "Please choose the number of the device to connect to (0 to %i):\n",
           devct-1);
           //check that there's input and in the correct range
          while(scanf("%i",&userchoice)==0||userchoice<0||userchoice>devct-1){
           printf("Invalid choice...Please choose again between 0 and %d:\n",
            devct-1);

           getchar();//clear carriage return from keyboard buffer after invalid
              //choice
```

```
  }
 }
 device=devnames[userchoice];
 return device;

}
else{
  printf("No MicroStrain devices found.\n");
  return device;
}

}

int send_command(char command_string[], ComPortHandle comPort){

 int number_hex_chars;
 int string_length;
 unsigned char* hexs;
 int i;

 if(strcmp(command_string,"exit")==0)
  return EXIT;

 //deterine length of command string
string_length = strlen(command_string);

 //each byte is represented by 2 ascii chars
 //so the length of the string should always be even
if(string_length%2!=0){
  printf("Invalid Command\n");
  return FALSE;
}

 //each hex char is only 1 byte from the two entered
number_hex_chars=string_length/2;

 //allocate memory for the byte-level command string
hexs=(unsigned char*)malloc(number_hex_chars*sizeof(unsigned char));

 //process each 2-char set from an ascii representation of the hexadecimal
 //command byte to the associated byte
for(i=0;i<number_hex_chars;i++){
  if(sscanf(&command_string[i*2],"%2x",(unsigned int *)&hexs[i])<1){
   printf("Invalid Command\n");
   return FALSE;
 }
```

```
}
 //write command
writeComPort(comPort, hexs, number_hex_chars);

 //deallocate the memory for the hex bytes
free(hexs);

return TRUE;
}


// int main(int argc, char *argv[]){
//   printf("fuck yeah\n");
//   stream();
//   printf("defineitly fuck yeah\n");
//   int i;
//        for(i=0;i<10;i++){
//           printf("we are printing: %s\n", reading[i]);
//        }
// }

int main(int argc, char *argv[])
{
    int portno;

    int n;
    if (argc < 2) {
        fprintf(stderr,"ERROR, no port provided\n");
        exit(1);
    }
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0)
       error("ERROR opening socket");
    bzero((char *) &serv_addr, sizeof(serv_addr));
    portno = atoi(argv[1]);
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = INADDR_ANY;
    serv_addr.sin_port = htons(portno);
    if (bind(sockfd, (struct sockaddr *) &serv_addr,
            sizeof(serv_addr)) < 0)
            error("ERROR on binding");
         getConnection();
   while(loop){
      waitForData();
      processInput();
   }
```

```
    close(newsockfd);
    close(sockfd);
    return 0;
}

void waitForData(){
   bzero(Data,256);
   BytesRecived=0;
   while(BytesRecived<1){
      BytesRecived = read(newsockfd,Data,255);
   }
   if (BytesRecived < 0) error("ERROR reading from socket");
}

void getConnection(){
   listen(sockfd,5);
    clilen = sizeof(cli_addr);
    newsockfd = accept(sockfd,
              (struct sockaddr *) &cli_addr,
              &clilen);
    if (newsockfd < 0)
        error("ERROR on accept");
}

void processInput(){
   if(strstr(Data,"::done::")!=NULL){
      loop = 0;
      printf("GOODBYE!!\n\n");
   }
   if(strstr(Data,"::stream::")){
      stream();
      // int i;
      // for(i=0;i<1000;i++){
      //     printf("we are printing: %s\n", reading[i]);
      //     write(newsockfd,reading[i],strlen(reading[i]));
      // }
   }
   printf("Here is the message: %s\n",Data);
   int n = write(newsockfd,"I got your message",18);

   if (n < 0) error("ERROR writing to socket");

}


void stream(){
```

```
 ComPortHandle comPort;
 int go = TRUE;
 char *dev;
 char a;
 char command_string[255];
 int ret;
 char* base = "/home/ben/MQP/tests/imu";
 char* extension = ".csv";
 char file[255];
 char number[10];
 //fd = fopen("/home/ben/imu.csv","w");
 //fd = fopen("/home/pi/imu.csv","w");

 a='\0';

 dev=&a;

  dev=scandev();
  if(strcmp(dev,"")!=0){
    printf("attempting to open: %s\n", dev);

   comPort = OpenComPort(dev);

  }
  else{

   printf("Failed to find attached device.\n");

  }


if(comPort > 0){
printf("%s\n", "readings are starting");
  int i;
  int r=0;
  char cmd = 0xcc;
  for(i=0;i<1000;i++){//continue until user chooses to exit
  //send_command("cf\r\n\r",comPort);
  //printf("%d\n", sizeof(char));
  write(comPort, "c", 1);
  //fprintf(comPort, "c\n", );
  printf("%s", "write complete");
  readPort(comPort,i);
  write(newsockfd,reading[i],strlen(reading[i]));
```

```
}
CloseComPort(comPort);
}

}

#.#.# IMU collection code

//Example code for interfacing with the Microstrain Inertial Sensor through its
//serial connection

/* compile using:


 gcc linux_serial_hex_driver.c -lpthread -o BINFILENAME

 Once compiled the desired device can be specified using a command line
 argument.

EXAMPLES:(FOR USB)

./BINFILENAME /dev/ttyACM0

or: (FOR SERIAL)

./BINFILENAME /dev/ttyS0


 -OR-

 On some linux distributions, (specifically ubuntu/debian) the program will
 scan /dev/serial/by-id for attached devices by name if connected to the USB
 port AND no device is specified at the command line. The USB connected
 3DM-GX3-xx will usually show up in /dev/ttyACM0 to  ttyACM# where represents
 the device number by the order the devices were attached.

 Copyright 2012 Microstrain.

 THE PRESENT SOFTWARE WHICH IS FOR GUIDANCE ONLY AIMS AT PROVIDING
 CUSTOMERS WITH CODING INFORMATION REGARDING THEIR PRODUCTS IN ORDER
 FOR THEM TO SAVE TIME. AS A RESULT, MICROSTRAIN SHALL NOT BE HELD LIABLE
```

```
        FOR ANY DIRECT, INDIRECT OR CONSEQUENTIAL DAMAGES WITH RESPECT TO ANY
        CLAIMS ARISING FROM THE CONTENT OF SUCH SOFTWARE AND/OR THE USE MADE BY
        CUSTOMERS OF THE CODING INFORMATION CONTAINED HEREIN IN CONNECTION WITH
        THEIR PRODUCTS.

        */

        #include <termios.h> // terminal io (serial port) interface
        #include <fcntl.h>  // File control definitions
        #include <errno.h>  // Error number definitions
        #include <assert.h>
        #include <stdio.h>
        #include <string.h>
        #include <pthread.h>
        #include <semaphore.h>
        #include <stdlib.h>
        #include <unistd.h>


        #define TRUE                    1
        #define FALSE                   0
        #define EXIT            -1
        #define BUFFSIZE    4096
        #define R2D 57.295827909f;

         typedef int ComPortHandle;
         typedef unsigned char Byte;
         FILE * fd;
         struct timeval start, end;
         long mtime, seconds, useconds;

         //my code
        float pBytesToFloat(char * bytes){
          float x = 0;
          union f {float r; unsigned char b[4];} f;
          // f.b[0]=bytes[0];
          // f.b[1]=bytes[1];
          // f.b[2]=bytes[2];
          // f.b[3]=bytes[3];
          f.b[0]=bytes[3];
          f.b[1]=bytes[2];
          f.b[2]=bytes[1];
          f.b[3]=bytes[0];
```

```
        //printf("\n0x%x, 0x%x 0x%x 0x%x\n", (unsigned int)(unsigned
char)bytes[0],(unsigned int)(unsigned char)bytes[1],(unsigned int)(unsigned
char)bytes[2],(unsigned int)(unsigned char)bytes[3]);
         return f.r;

        }

        //function to remove whitespace from commands
         void remove_space(char *string_with_spaces){

          int i=0;
          int parse_index = 0;
          char *no_space_string;

          if(string_with_spaces==0)
           return;

         no_space_string=string_with_spaces;

         for(i;i<strlen(string_with_spaces);i++){
           if(!(string_with_spaces[i]==' '||string_with_spaces[i]=='\t'))
            *no_space_string++ = string_with_spaces[i];
         }

         *no_space_string = '\0';

         no_space_string = 0;

         return;
        }

        // Utility functions for working with a com port in Linux

        // Purge
        // Clears the com port's read and write buffers

        int Purge(ComPortHandle comPortHandle){

         if (tcflush(comPortHandle,TCIOFLUSH)==-1){

          printf("flush failed\n");
          return FALSE;

        }

        return TRUE;
```

```
}

// OpenComPort
// Opens a com port with the correct settings for communicating with a
// MicroStrain 3DM-GX3-25 sensor

ComPortHandle OpenComPort(const char* comPortPath){

 ComPortHandle comPort = open(comPortPath, O_RDWR | O_NOCTTY);

 if (comPort== -1){ //Opening of port failed

  printf("Unable to open com Port %s\n Errno = %i\n", comPortPath, errno);
  return -1;

}

 //Get the current options for the port...
struct termios options;
tcgetattr(comPort, &options);

 //set the baud rate to 115200
int baudRate = B115200;
cfsetospeed(&options, baudRate);
cfsetispeed(&options, baudRate);

 //set the number of data bits.
 options.c_cflag &= ~CSIZE; // Mask the character size bits
 options.c_cflag |= CS8;

 //set the number of stop bits to 1
 options.c_cflag &= ~CSTOPB;

 //Set parity to None
 options.c_cflag &=~PARENB;

 //set for non-canonical (raw processing, no echo, etc.)
 options.c_iflag = IGNPAR; // ignore parity check close_port(int
 options.c_oflag = 0; // raw output
 options.c_lflag = 0; // raw input

 //Time-Outs -- won't work with NDELAY option in the call to open
 options.c_cc[VMIN] = 0;  // block reading until RX x characers. If x = 0,
            // it is non-blocking.
 options.c_cc[VTIME] = 1;  // Inter-Character Timer -- i.e. timeout= x*.1 s
```

```
        //Set local mode and enable the receiver
        options.c_cflag |= (CLOCAL | CREAD);

        //Purge serial port buffers
        Purge(comPort);

        //Set the new options for the port...
        int status=tcsetattr(comPort, TCSANOW, &options);

        if (status != 0){ //For error message

         printf("Configuring comport failed\n");
         return status;

        }

        //Purge serial port buffers
       Purge(comPort);

       return comPort;

       }

       // CloseComPort
       // Closes a port that was previously opened with OpenComPort
       void CloseComPort(ComPortHandle comPort){

        close(comPort);

       }

       // readComPort
       // read the specivied number of bytes from the com port
       int readComPort(ComPortHandle comPort, Byte* bytes, int bytesToRead){

        int bytesRead = read(comPort, bytes, bytesToRead);
        return bytesRead;

       }

       // writeComPort
       // send bytes to the com port
       int writeComPort(ComPortHandle comPort, unsigned char* bytesToWrite, int
size){
```

```
 return write(comPort, bytesToWrite, size);

}

// Simple Linux Console interface function

//attemots to read BUFFSIZE bytes at a time from the open port
int readPort(ComPortHandle comPort){

 int i=0;
 int size;
 struct termios initial_settings, new_settings;
 char command_string[255];
 char c;
 int ret;
 Byte response[BUFFSIZE] = {0};

 //get current terminal settings
 tcgetattr(0,&initial_settings);

 new_settings = initial_settings;
 new_settings.c_lflag &= ~ICANON;
 new_settings.c_cc[VMIN] = 0;
 new_settings.c_cc[VTIME] = 0;

 tcsetattr(0, TCSANOW, &new_settings);

 //determine if there are bytes to read
 size = readComPort(comPort, &response[0], BUFFSIZE);

 if(size<0)
  return FALSE;

 //if ther are bytes to read output them and check again
while(size>0){

  //write out any read bytes
  if(response[0]==0xcf){
    gettimeofday(&end, NULL);

    seconds  = end.tv_sec  - start.tv_sec;
    useconds = end.tv_usec - start.tv_usec;
    mtime = ((seconds) * 1000 + useconds/1000.0) + 0.5;
  fprintf(fd, "%f,%f,%f,%f,%f,%f,%ld\n",
    pBytesToFloat(&(response[1])),
    pBytesToFloat(&(response[5])),
```

```
    pBytesToFloat(&(response[9])),
    pBytesToFloat(&(response[13])),
    pBytesToFloat(&(response[17])),
    pBytesToFloat(&(response[21])),
    mtime
    );
}else{
  for(i=0;i<size;i++){

    printf("%.2x ",response[i]|0x00);

  }
}
  c = getchar();

  if(c != EOF)
  {
    tcsetattr(0, TCSANOW, &initial_settings);
    ungetc(c,stdin);
    gets(command_string);
    remove_space(command_string);
    ret=send_command(command_string,comPort);
    if(ret==EXIT)
     return EXIT;
   tcsetattr(0, TCSANOW, &new_settings);
 }

 size = readComPort(comPort, &response[0], BUFFSIZE);

}

tcsetattr(0, TCSANOW, &initial_settings);

return TRUE;

}

//scandev
//finds attached microstrain devices and prompts user for choice then returns
//selected portname
char* scandev(){

 FILE *instream;
 char devnames[255][255];//allows for up to 256 devices with path links up to
          //255 characters long each
 int devct=0; //counter for number of devices
```

```
        int i=0;
        int j=0;
        int userchoice=0;
        char *device;

        //search /dev/serial for microstrain devices
        char *command = "find /dev/serial -print | grep -i microstrain";

        printf("Searching for devices...\n");

        instream=popen(command, "r");//execute piped command in read mode

        if(!instream){//SOMETHING WRONG WITH THE SYSTEM COMMAND
PIPE...EXITING
         printf("ERROR BROKEN PIPELINE %s\n", command);
         return device;
        }

        //load char array of device addresses
        for(i=0;i<255&&(fgets(devnames[i],sizeof(devnames[i]), instream));i++){
         ++devct;
        }

        for(i=0;i<devct;i++){
         for(j=0;j<sizeof(devnames[i]);j++){
          if(devnames[i][j]=='\n'){
           devnames[i][j]='\0';//replaces newline inserted by pipe reader with
                   //char array terminator character
           break;//breaks loop after replacement
          }
         }
        printf("Device Found:\n%d: %s\n",i,devnames[i]);
        }

        //CHOOSE DEVICE TO CONNECT TO AND CONNECT TO IT (IF THERE
ARE ANY CONNECTED)

        if(devct>0){
         printf("Number of devices = %d\n", devct);
         if(devct>1){
          printf(
           "Please choose the number of the device to connect to (0 to %i):\n",
           devct-1);
           //check that there's input and in the correct range
          while(scanf("%i",&userchoice)==0||userchoice<0||userchoice>devct-1){
           printf("Invalid choice...Please choose again between 0 and %d:\n",
```

```
      devct-1);

    getchar();//clear carriage return from keyboard buffer after invalid
        //choice
  }
 }
 device=devnames[userchoice];
 return device;

}
else{
  printf("No MicroStrain devices found.\n");
  return device;
}

}

int send_command(char command_string[], ComPortHandle comPort){

 int number_hex_chars;
 int string_length;
 unsigned char* hexs;
 int i;

 if(strcmp(command_string,"exit")==0)
  return EXIT;

 //deterine length of command string
string_length = strlen(command_string);

 //each byte is represented by 2 ascii chars
 //so the length of the string should always be even
if(string_length%2!=0){
  printf("Invalid Command\n");
  return FALSE;
}

 //each hex char is only 1 byte from the two entered
number_hex_chars=string_length/2;

 //allocate memory for the byte-level command string
hexs=(unsigned char*)malloc(number_hex_chars*sizeof(unsigned char));

 //process each 2-char set from an ascii representation of the hexadecimal
 //command byte to the associated byte
for(i=0;i<number_hex_chars;i++){
```

```
  if(sscanf(&command_string[i*2],"%2x",(unsigned int *)&hexs[i])<1){
   printf("Invalid Command\n");
   return FALSE;
  }
 }
 //write command
writeComPort(comPort, hexs, number_hex_chars);

 //deallocate the memory for the hex bytes
free(hexs);

return TRUE;
}




// main
int main(int argc, char* argv[]){

 ComPortHandle comPort;
 int go = TRUE;
 char *dev;
 char a;
 char command_string[255];
 int ret;
 char* base = "/home/ben/MQP/tests/imu";
 char* extension = ".csv";
 char file[255];
 char number[10];
 //fd = fopen("/home/ben/imu.csv","w");
 //fd = fopen("/home/pi/imu.csv","w");
 int ct = 0; //the number used to append to the output file's name
 for(ct=0; ct < 5000; ct++){
  strcpy ( file, base );
  //itoa(ct,number,10);
  snprintf(number, 10,"%d",ct);
  strcat(file,number);
  strcat(file,extension);
  if(fd = fopen(file,"r")){
    fclose(fd);
  }else{
    break;
  }
```

```
 }
 //fclose(fd);
  fd = fopen(file,"w");
 gettimeofday(&start, NULL);
 printf("%s\n", "starting");
fprintf(fd, "Roll,Pitch,Yaw,AngRateX,AngRateY,AngRateZ,timestamp\n");
printf("%s\n", "first file write");

 a='\0';

 dev=&a;

 if(argc<2){//No port specified at commandline so search for attached devices

  dev=scandev();
  if(strcmp(dev,"")!=0){

   printf("Attempting to open port...%s\n",dev);
   comPort = OpenComPort(dev);

  }
  else{

   printf("Failed to find attached device.\n");
   return FALSE;

  }

}
 else{//Open port specified at commandline

  printf("Attempting to open port...%s\n",argv[1]);
  comPort = OpenComPort(argv[1]);

}

//30

printf("comPort=%i\n",comPort);

if(comPort > 0){

  printf("Connected. \n\nEnter Command as specified in 3DM-GX3 MIP DCP");
  printf(" followed by an <ENTER> Then utility will attempt to read");
  printf("reply on port.\n The resulting read will read all characters ");
  printf(" available currently then return for user command.\n");
```

```
  int i;
  int r=0;
  char cmd = 0xcc;
  for(i=0;i<3000;i++){//continue until user chooses to exit


  usleep(1);

  // printf("\nEnter Command (\"exit\" to exit)> ");
   //get command from user
   //gets(command_string);
   //remove_space(command_string);
   //ret=send_command(command_string,comPort);
  send_command("cf",comPort);
  readPort(comPort);
   //exit if user specified this
   //if(ret==EXIT)
    //break;
   //read port if user didn't ask for exit
  //ret=readPort(comPort);
   //exit if user specified this
  //if(ret==EXIT)
   // break;

}

printf("EXITING\n");
CloseComPort(comPort);
fclose(fd);

}

return 0;

}

#.#.# output of Weka Classifier code
Accuracy,HistogramBins,WindowSize,Algorithm
0.9012931034482758,10.0,670.0,0.0,
0.9004464285714285,10.0,690.0,0.0,
0.9022321428571428,10.0,700.0,0.0,
0.9059090909090909,10.0,710.0,0.0,
0.904245283018868,10.0,730.0,0.0,
0.9132075471698113,10.0,740.0,0.0,
0.9149038461538461,10.0,750.0,0.0,
0.9,10.0,770.0,0.0,
```

0.926,10.0,780.0,0.0,
0.9183673469387755,10.0,790.0,0.0,
0.9161458333333333,10.0,800.0,0.0,
0.909375,10.0,810.0,0.0,
0.925,10.0,820.0,0.0,
0.9164893617021277,10.0,830.0,0.0,
0.9217391304347826,10.0,840.0,0.0,
0.9152173913043479,10.0,850.0,0.0,
0.9161111111111111,10.0,860.0,0.0,
0.9193181818181818,10.0,870.0,0.0,
0.9221590909090909,10.0,880.0,0.0,
0.9209302325581395,10.0,890.0,0.0,
0.9203488372093023,10.0,900.0,0.0,
0.924404761904762,10.0,910.0,0.0,
0.919047619047619,10.0,920.0,0.0,
0.9226190476190477,10.0,930.0,0.0,
0.9231707317073171,10.0,940.0,0.0,
0.9189024390243903,10.0,950.0,0.0,
0.918125,10.0,960.0,0.0,
0.930625,10.0,970.0,0.0,
0.9262820512820513,10.0,980.0,0.0,
0.9326923076923077,10.0,990.0,0.0,
0.9322368421052631,10.0,1000.0,0.0,
0.9269736842105263,10.0,1010.0,0.0,
0.9243421052631579,10.0,1020.0,0.0,
0.9263513513513514,10.0,1030.0,0.0,
0.9256756756756757,10.0,1040.0,0.0,
0.9263513513513514,10.0,1050.0,0.0,
0.9305555555555556,10.0,1060.0,0.0,
0.9270833333333334,10.0,1070.0,0.0,
0.9291666666666667,10.0,1080.0,0.0,
0.92,10.0,1090.0,0.0,
0.9364285714285714,10.0,1100.0,0.0,
0.9257142857142857,10.0,1110.0,0.0,
0.9095588235294118,10.0,1120.0,0.0,
0.9301470588235294,10.0,1130.0,0.0,
0.9272058823529412,10.0,1140.0,0.0,
0.9295454545454546,10.0,1150.0,0.0,
0.931060606060606,10.0,1160.0,0.0,
0.928030303030303,10.0,1170.0,0.0,
0.93125,10.0,1180.0,0.0,
0.92265625,10.0,1190.0,0.0,
0.93828125,10.0,1200.0,0.0,
0.93203125,10.0,1210.0,0.0,
0.9419354838709677,10.0,1220.0,0.0,
0.9217741935483871,10.0,1230.0,0.0,

0.9290322580645162,10.0,1240.0,0.0,
0.9316666666666666,10.0,1250.0,0.0,
0.9208333333333333,10.0,1260.0,0.0,
0.9416666666666667,10.0,1270.0,0.0,
0.9308333333333333,10.0,1280.0,0.0,
0.9275,10.0,1290.0,0.0,
0.9422413793103448,10.0,1300.0,0.0,
0.9431034482758621,10.0,1310.0,0.0,
0.9405172413793104,10.0,1320.0,0.0,
0.9456896551724138,10.0,1330.0,0.0,
0.9375,10.0,1340.0,0.0,
0.9464285714285714,10.0,1350.0,0.0,
0.9410714285714286,10.0,1360.0,0.0,
0.9455357142857143,10.0,1370.0,0.0,
0.937962962962963,10.0,1380.0,0.0,
0.9268518518518518,10.0,1390.0,0.0,
0.9222222222222223,10.0,1400.0,0.0,
0.9462962962962963,10.0,1410.0,0.0,
0.9342592592592592,10.0,1420.0,0.0,
0.9461538461538461,10.0,1430.0,0.0,
0.9336538461538462,10.0,1440.0,0.0,
0.9403846153846154,10.0,1450.0,0.0,
0.9403846153846154,10.0,1460.0,0.0,
0.9192307692307692,10.0,1470.0,0.0,
0.9384615384615385,10.0,1480.0,0.0,
0.945,10.0,1490.0,0.0,
0.934,10.0,1500.0,0.0,
0.9044871794871795,15.0,980.0,0.0,
0.906578947368421,15.0,1000.0,0.0,
0.9078947368421053,15.0,1010.0,0.0,
0.9111842105263158,15.0,1020.0,0.0,
0.9013513513513514,15.0,1040.0,0.0,
0.9094594594594595,15.0,1050.0,0.0,
0.9027777777777778,15.0,1060.0,0.0,
0.9125,15.0,1070.0,0.0,
0.9048611111111111,15.0,1080.0,0.0,
0.9014285714285715,15.0,1090.0,0.0,
0.9214285714285714,15.0,1100.0,0.0,
0.9058823529411765,15.0,1120.0,0.0,
0.9073529411764706,15.0,1130.0,0.0,
0.9191176470588235,15.0,1140.0,0.0,
0.9151515151515152,15.0,1150.0,0.0,
0.9037878787878788,15.0,1160.0,0.0,
0.91328125,15.0,1190.0,0.0,
0.928125,15.0,1210.0,0.0,
0.9064516129032258,15.0,1220.0,0.0,

0.9120967741935484,15.0,1230.0,0.0,
0.9120967741935484,15.0,1240.0,0.0,
0.9016666666666666,15.0,1250.0,0.0,
0.9283333333333333,15.0,1260.0,0.0,
0.9283333333333333,15.0,1270.0,0.0,
0.9058333333333334,15.0,1280.0,0.0,
0.9116666666666666,15.0,1290.0,0.0,
0.9077586206896552,15.0,1300.0,0.0,
0.9112068965517242,15.0,1310.0,0.0,
0.928448275862069,15.0,1320.0,0.0,
0.9172413793103448,15.0,1330.0,0.0,
0.9142857142857143,15.0,1340.0,0.0,
0.9071428571428571,15.0,1350.0,0.0,
0.9116071428571428,15.0,1360.0,0.0,
0.9089285714285714,15.0,1370.0,0.0,
0.9259259259259259,15.0,1380.0,0.0,
0.9175925925925926,15.0,1400.0,0.0,
0.9157407407407407,15.0,1410.0,0.0,
0.9148148148148149,15.0,1420.0,0.0,
0.9192307692307692,15.0,1430.0,0.0,
0.9288461538461539,15.0,1440.0,0.0,
0.9240384615384616,15.0,1450.0,0.0,
0.9259615384615385,15.0,1460.0,0.0,
0.9096153846153846,15.0,1470.0,0.0,
0.9057692307692308,15.0,1480.0,0.0,
0.92,15.0,1490.0,0.0,
0.921,15.0,1500.0,0.0,
0.9059701492537313,20.0,580.0,0.0,
0.902016129032258,20.0,630.0,0.0,
0.9016666666666666,20.0,650.0,0.0,
0.9114406779661017,20.0,660.0,0.0,
0.9081896551724138,20.0,670.0,0.0,
0.9043859649122807,20.0,680.0,0.0,
0.90625,20.0,690.0,0.0,
0.9044642857142857,20.0,700.0,0.0,
0.9045454545454545,20.0,710.0,0.0,
0.9013888888888889,20.0,720.0,0.0,
0.9183962264150943,20.0,730.0,0.0,
0.9193396226415095,20.0,740.0,0.0,
0.9105769230769231,20.0,750.0,0.0,
0.9166666666666666,20.0,760.0,0.0,
0.9185,20.0,770.0,0.0,
0.913,20.0,780.0,0.0,
0.9086734693877551,20.0,790.0,0.0,
0.9229166666666667,20.0,800.0,0.0,
0.9208333333333333,20.0,810.0,0.0,

0.9164893617021277,20.0,820.0,0.0,
0.9095744680851063,20.0,830.0,0.0,
0.9119565217391304,20.0,840.0,0.0,
0.907608695652174,20.0,850.0,0.0,
0.9055555555555556,20.0,860.0,0.0,
0.9227272727272727,20.0,870.0,0.0,
0.9176136363636364,20.0,880.0,0.0,
0.911046511627907,20.0,890.0,0.0,
0.9232558139534883,20.0,900.0,0.0,
0.9255952380952381,20.0,910.0,0.0,
0.9166666666666666,20.0,920.0,0.0,
0.9083333333333333,20.0,930.0,0.0,
0.9201219512195122,20.0,940.0,0.0,
0.9237804878048781,20.0,950.0,0.0,
0.9175,20.0,960.0,0.0,
0.921875,20.0,970.0,0.0,
0.9237179487179488,20.0,980.0,0.0,
0.9282051282051282,20.0,990.0,0.0,
0.9236842105263158,20.0,1000.0,0.0,
0.9157894736842105,20.0,1010.0,0.0,
0.9289473684210526,20.0,1020.0,0.0,
0.925,20.0,1030.0,0.0,
0.9277027027027027,20.0,1040.0,0.0,
0.9324324324324325,20.0,1050.0,0.0,
0.9270833333333334,20.0,1060.0,0.0,
0.9277777777777778,20.0,1070.0,0.0,
0.9284722222222223,20.0,1080.0,0.0,
0.9385714285714286,20.0,1090.0,0.0,
0.9264285714285714,20.0,1100.0,0.0,
0.9235714285714286,20.0,1110.0,0.0,
0.9264705882352942,20.0,1120.0,0.0,
0.9257352941176471,20.0,1130.0,0.0,
0.9330882352941177,20.0,1140.0,0.0,
0.9265151515151515,20.0,1150.0,0.0,
0.928030303030303,20.0,1160.0,0.0,
0.9340909090909091,20.0,1170.0,0.0,
0.934375,20.0,1180.0,0.0,
0.94453125,20.0,1190.0,0.0,
0.92421875,20.0,1200.0,0.0,
0.95234375,20.0,1210.0,0.0,
0.9354838709677419,20.0,1220.0,0.0,
0.9298387096774193,20.0,1230.0,0.0,
0.9233870967741935,20.0,1240.0,0.0,
0.945,20.0,1250.0,0.0,
0.9333333333333333,20.0,1260.0,0.0,
0.935,20.0,1270.0,0.0,

0.915,20.0,1280.0,0.0,
0.9191666666666667,20.0,1290.0,0.0,
0.9241379310344827,20.0,1300.0,0.0,
0.9296551724137931,20.0,1310.0,0.0,
0.9405172413793104,20.0,1320.0,0.0,
0.9353448275862069,20.0,1330.0,0.0,
0.9375,20.0,1340.0,0.0,
0.9464285714285714,20.0,1350.0,0.0,
0.9348214285714286,20.0,1360.0,0.0,
0.9410714285714286,20.0,1370.0,0.0,
0.9287037037037037,20.0,1380.0,0.0,
0.9407407407407408,20.0,1390.0,0.0,
0.9388888888888889,20.0,1400.0,0.0,
0.9268518518518518,20.0,1410.0,0.0,
0.9324074074074075,20.0,1420.0,0.0,
0.9423076923076923,20.0,1430.0,0.0,
0.9298076923076923,20.0,1440.0,0.0,
0.9173076923076923,20.0,1450.0,0.0,
0.9259615384615385,20.0,1460.0,0.0,
0.9403846153846154,20.0,1470.0,0.0,
0.9471153846153846,20.0,1480.0,0.0,
0.931,20.0,1490.0,0.0,
0.932,20.0,1500.0,0.0,
0.9018181818181819,25.0,710.0,0.0,
0.9105769230769231,25.0,750.0,0.0,
0.901,25.0,770.0,0.0,
0.9,25.0,790.0,0.0,
0.9036458333333334,25.0,800.0,0.0,
0.9067708333333333,25.0,810.0,0.0,
0.9069148936170213,25.0,820.0,0.0,
0.9042553191489362,25.0,830.0,0.0,
0.903804347826087,25.0,850.0,0.0,
0.9159090909090909,25.0,870.0,0.0,
0.9113636363636364,25.0,880.0,0.0,
0.9104651162790698,25.0,890.0,0.0,
0.9040697674418605,25.0,900.0,0.0,
0.9011904761904762,25.0,910.0,0.0,
0.9160714285714285,25.0,920.0,0.0,
0.9083333333333333,25.0,930.0,0.0,
0.9085365853658537,25.0,940.0,0.0,
0.9073170731707317,25.0,950.0,0.0,
0.915,25.0,960.0,0.0,
0.914375,25.0,970.0,0.0,
0.9192307692307692,25.0,980.0,0.0,
0.925,25.0,990.0,0.0,
0.9111842105263158,25.0,1000.0,0.0,

0.906578947368421,25.0,1010.0,0.0,
0.906578947368421,25.0,1020.0,0.0,
0.902027027027027,25.0,1030.0,0.0,
0.9108108108108108,25.0,1040.0,0.0,
0.9168918918918919,25.0,1050.0,0.0,
0.9201388888888888,25.0,1060.0,0.0,
0.9118055555555555,25.0,1070.0,0.0,
0.9083333333333333,25.0,1080.0,0.0,
0.93,25.0,1090.0,0.0,
0.9214285714285714,25.0,1100.0,0.0,
0.9207142857142857,25.0,1110.0,0.0,
0.9110294117647059,25.0,1120.0,0.0,
0.9198529411764705,25.0,1130.0,0.0,
0.9308823529411765,25.0,1140.0,0.0,
0.931060606060606,25.0,1150.0,0.0,
0.9363636363636364,25.0,1160.0,0.0,
0.9234848484848485,25.0,1170.0,0.0,
0.9203125,25.0,1180.0,0.0,
0.9203125,25.0,1190.0,0.0,
0.91171875,25.0,1200.0,0.0,
0.921875,25.0,1210.0,0.0,
0.9233870967741935,25.0,1220.0,0.0,
0.9161290322580645,25.0,1230.0,0.0,
0.9266129032258065,25.0,1240.0,0.0,
0.9141666666666667,25.0,1250.0,0.0,
0.9141666666666667,25.0,1260.0,0.0,
0.9066666666666666,25.0,1270.0,0.0,
0.9275,25.0,1280.0,0.0,
0.9141666666666667,25.0,1290.0,0.0,
0.9275862068965517,25.0,1300.0,0.0,
0.9241379310344827,25.0,1310.0,0.0,
0.9146551724137931,25.0,1320.0,0.0,
0.9189655172413793,25.0,1330.0,0.0,
0.9285714285714286,25.0,1340.0,0.0,
0.9160714285714285,25.0,1350.0,0.0,
0.9276785714285715,25.0,1360.0,0.0,
0.91875,25.0,1370.0,0.0,
0.9333333333333333,25.0,1380.0,0.0,
0.9175925925925926,25.0,1390.0,0.0,
0.9259259259259259,25.0,1400.0,0.0,
0.9259259259259259,25.0,1410.0,0.0,
0.9333333333333333,25.0,1420.0,0.0,
0.9240384615384616,25.0,1430.0,0.0,
0.925,25.0,1440.0,0.0,
0.9153846153846154,25.0,1450.0,0.0,
0.9153846153846154,25.0,1460.0,0.0,

0.9201923076923076,25.0,1470.0,0.0,
0.9394230769230769,25.0,1480.0,0.0,
0.929,25.0,1490.0,0.0,
0.928,25.0,1500.0,0.0,
0.9071428571428571,30.0,700.0,0.0,
0.905,30.0,710.0,0.0,
0.9061320754716982,30.0,730.0,0.0,
0.9045,30.0,770.0,0.0,
0.903,30.0,780.0,0.0,
0.9056122448979592,30.0,790.0,0.0,
0.9005208333333333,30.0,800.0,0.0,
0.9026041666666667,30.0,810.0,0.0,
0.9058510638297872,30.0,820.0,0.0,
0.9085106382978724,30.0,830.0,0.0,
0.9157608695652174,30.0,840.0,0.0,
0.9157608695652174,30.0,850.0,0.0,
0.905,30.0,860.0,0.0,
0.9045454545454545,30.0,870.0,0.0,
0.9028409090909091,30.0,880.0,0.0,
0.9116279069767442,30.0,900.0,0.0,
0.9077380952380952,30.0,910.0,0.0,
0.9083333333333333,30.0,920.0,0.0,
0.905952380952381,30.0,930.0,0.0,
0.9182926829268293,30.0,940.0,0.0,
0.9067073170731708,30.0,950.0,0.0,
0.9075,30.0,960.0,0.0,
0.919375,30.0,970.0,0.0,
0.9096153846153846,30.0,980.0,0.0,
0.9198717948717948,30.0,990.0,0.0,
0.9138157894736842,30.0,1000.0,0.0,
0.9197368421052632,30.0,1010.0,0.0,
0.9151315789473684,30.0,1020.0,0.0,
0.9067567567567567,30.0,1030.0,0.0,
0.925,30.0,1040.0,0.0,
0.9182432432432432,30.0,1050.0,0.0,
0.9145833333333333,30.0,1060.0,0.0,
0.9145833333333333,30.0,1070.0,0.0,
0.9201388888888888,30.0,1080.0,0.0,
0.9164285714285715,30.0,1090.0,0.0,
0.9214285714285714,30.0,1100.0,0.0,
0.9192857142857143,30.0,1110.0,0.0,
0.925,30.0,1120.0,0.0,
0.9294117647058824,30.0,1130.0,0.0,
0.924264705882353,30.0,1140.0,0.0,
0.9075757575757576,30.0,1150.0,0.0,
0.9143939393939394,30.0,1160.0,0.0,

0.9356060606060606,30.0,1170.0,0.0,
0.9296875,30.0,1180.0,0.0,
0.91484375,30.0,1190.0,0.0,
0.92265625,30.0,1200.0,0.0,
0.93359375,30.0,1210.0,0.0,
0.9330645161290323,30.0,1220.0,0.0,
0.9225806451612903,30.0,1230.0,0.0,
0.9290322580645162,30.0,1240.0,0.0,
0.9208333333333333,30.0,1250.0,0.0,
0.9158333333333334,30.0,1260.0,0.0,
0.9358333333333333,30.0,1270.0,0.0,
0.9158333333333334,30.0,1280.0,0.0,
0.9183333333333333,30.0,1290.0,0.0,
0.9206896551724137,30.0,1300.0,0.0,
0.9129310344827586,30.0,1310.0,0.0,
0.9267241379310345,30.0,1320.0,0.0,
0.9172413793103448,30.0,1330.0,0.0,
0.9303571428571429,30.0,1340.0,0.0,
0.9392857142857143,30.0,1350.0,0.0,
0.9169642857142857,30.0,1360.0,0.0,
0.9223214285714286,30.0,1370.0,0.0,
0.925,30.0,1380.0,0.0,
0.9111111111111111,30.0,1390.0,0.0,
0.9148148148148149,30.0,1400.0,0.0,
0.912962962962963,30.0,1410.0,0.0,
0.9157407407407407,30.0,1420.0,0.0,
0.9134615384615384,30.0,1430.0,0.0,
0.9192307692307692,30.0,1440.0,0.0,
0.9365384615384615,30.0,1450.0,0.0,
0.9240384615384616,30.0,1460.0,0.0,
0.9240384615384616,30.0,1470.0,0.0,
0.9173076923076923,30.0,1480.0,0.0,
0.921,30.0,1490.0,0.0,
0.94,30.0,1500.0,0.0,
0.9016304347826087,35.0,850.0,0.0,
0.9083333333333333,35.0,920.0,0.0,
0.9017857142857143,35.0,930.0,0.0,
0.9024390243902439,35.0,950.0,0.0,
0.90375,35.0,970.0,0.0,
0.9072368421052631,35.0,1010.0,0.0,
0.9148648648648648,35.0,1040.0,0.0,
0.9125,35.0,1070.0,0.0,
0.915,35.0,1090.0,0.0,
0.9071428571428571,35.0,1100.0,0.0,
0.905,35.0,1110.0,0.0,
0.9051470588235294,35.0,1120.0,0.0,

0.913235294117647,35.0,1130.0,0.0,
0.9227941176470589,35.0,1140.0,0.0,
0.9068181818181819,35.0,1170.0,0.0,
0.90078125,35.0,1190.0,0.0,
0.9203125,35.0,1200.0,0.0,
0.9015625,35.0,1210.0,0.0,
0.9088709677419354,35.0,1220.0,0.0,
0.9056451612903226,35.0,1230.0,0.0,
0.9058333333333334,35.0,1250.0,0.0,
0.9008333333333334,35.0,1270.0,0.0,
0.9091666666666667,35.0,1280.0,0.0,
0.9125,35.0,1290.0,0.0,
0.9008620689655172,35.0,1310.0,0.0,
0.9017241379310345,35.0,1320.0,0.0,
0.9060344827586206,35.0,1330.0,0.0,
0.9053571428571429,35.0,1340.0,0.0,
0.9089285714285714,35.0,1370.0,0.0,
0.9037037037037037,35.0,1380.0,0.0,
0.9009259259259259,35.0,1390.0,0.0,
0.912962962962963,35.0,1400.0,0.0,
0.9027777777777778,35.0,1410.0,0.0,
0.9055555555555556,35.0,1420.0,0.0,
0.9288461538461539,35.0,1430.0,0.0,
0.9096153846153846,35.0,1440.0,0.0,
0.9134615384615384,35.0,1450.0,0.0,
0.9038461538461539,35.0,1460.0,0.0,
0.9221153846153847,35.0,1470.0,0.0,
0.9086538461538461,35.0,1480.0,0.0,
0.9,35.0,1490.0,0.0,
0.912,35.0,1500.0,0.0,
0.9017045454545455,40.0,870.0,0.0,
0.9047619047619048,40.0,910.0,0.0,
0.905952380952381,40.0,930.0,0.0,
0.913125,40.0,960.0,0.0,
0.9019230769230769,40.0,990.0,0.0,
0.9,40.0,1000.0,0.0,
0.904054054054054,40.0,1030.0,0.0,
0.9094594594594595,40.0,1040.0,0.0,
0.9033783783783784,40.0,1050.0,0.0,
0.9027777777777778,40.0,1060.0,0.0,
0.9048611111111111,40.0,1070.0,0.0,
0.9152777777777777,40.0,1080.0,0.0,
0.9064285714285715,40.0,1100.0,0.0,
0.9102941176470588,40.0,1120.0,0.0,
0.9139705882352941,40.0,1130.0,0.0,
0.9213235294117647,40.0,1140.0,0.0,

0.9045454545454545,40.0,1160.0,0.0,
0.9166666666666666,40.0,1170.0,0.0,
0.90703125,40.0,1180.0,0.0,
0.90234375,40.0,1190.0,0.0,
0.92109375,40.0,1200.0,0.0,
0.92109375,40.0,1210.0,0.0,
0.917741935483871,40.0,1220.0,0.0,
0.9169354838709678,40.0,1230.0,0.0,
0.9282258064516129,40.0,1240.0,0.0,
0.915,40.0,1250.0,0.0,
0.9208333333333333,40.0,1260.0,0.0,
0.9091666666666667,40.0,1270.0,0.0,
0.905,40.0,1280.0,0.0,
0.925,40.0,1300.0,0.0,
0.9137931034482759,40.0,1310.0,0.0,
0.9232758620689655,40.0,1320.0,0.0,
0.9275862068965517,40.0,1330.0,0.0,
0.9133928571428571,40.0,1340.0,0.0,
0.9241071428571429,40.0,1350.0,0.0,
0.9133928571428571,40.0,1360.0,0.0,
0.9169642857142857,40.0,1370.0,0.0,
0.9185185185185185,40.0,1380.0,0.0,
0.9101851851851852,40.0,1390.0,0.0,
0.9111111111111111,40.0,1400.0,0.0,
0.9120370370370371,40.0,1410.0,0.0,
0.9120370370370371,40.0,1420.0,0.0,
0.9067307692307692,40.0,1430.0,0.0,
0.9182692307692307,40.0,1440.0,0.0,
0.9067307692307692,40.0,1450.0,0.0,
0.9221153846153847,40.0,1460.0,0.0,
0.9211538461538461,40.0,1470.0,0.0,
0.9192307692307692,40.0,1480.0,0.0,
0.916,40.0,1490.0,0.0,
0.914,40.0,1500.0,0.0,
0.9025641025641026,45.0,990.0,0.0,
0.9026315789473685,45.0,1020.0,0.0,
0.902027027027027,45.0,1030.0,0.0,
0.9,45.0,1040.0,0.0,
0.9027777777777778,45.0,1080.0,0.0,
0.9135714285714286,45.0,1090.0,0.0,
0.9028571428571428,45.0,1110.0,0.0,
0.9073529411764706,45.0,1120.0,0.0,
0.9051470588235294,45.0,1140.0,0.0,
0.9053030303030303,45.0,1150.0,0.0,
0.9030303030303031,45.0,1160.0,0.0,
0.909375,45.0,1180.0,0.0,

0.90546875,45.0,1190.0,0.0,
0.9140625,45.0,1200.0,0.0,
0.9109375,45.0,1210.0,0.0,
0.9016129032258065,45.0,1220.0,0.0,
0.9064516129032258,45.0,1230.0,0.0,
0.9048387096774193,45.0,1240.0,0.0,
0.9033333333333333,45.0,1250.0,0.0,
0.9075

10) Bibliography

National Spinal Cord Injury Statistical Center. www.NSCISC.uab.edu. [Online]. https://www.nscisc.uab.edu/PublicDocuments/fact_figures_docs/Facts%202012%20Feb%20Final.pdf
1]

Myomo. Myomo. [Online]. myomo.com
2]

Ute Eck et al., "OrthoJacket: an active FES-hybrid orthosis for the paralysed upper extremity," 2011.
3]

G. Yang, W. Lin, M. Shabbir, C. Pham, and S. Yeo, "Kinematic Design of a 7-DOF Cable-Driven Humanoid Arm," in *International Conference on ADvanced Intelligent Mechatronics*, Montery, CA, 2005.
4]

The Merck Manual. The Merck Manual. [Online]. Physical Therapy (PT)
5]

"Hill-type Muscle Model Parameters Determined from Experiments on Single Muscles Show Large Animal-to-animal Variation - Springer.," 2012.
6]

Glenn K. Klute, Joseph M. Czerniecki, and Blake Hannaford, "Artificial Muscles Actuators for Biorobotic Systems," Electrical Engineering, Rehabilitation Medicine, Veterans Affairs, University of Washington, Seattle, WA,.
7]

Frank Daerden and Dirk Lefber, "Pneumatic Artificial Muscles: actuators for robotics and automation," Mechanical Engineering, Vrije Universiteit Brussel, Brussel,.
8]

Ching-Ping Chou and Blake Hannaford, "Measurement and Modeling of McKibben Pnumatic Artificial Muscles," *IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION,* vol. 12, no. 1, pp. 90-102, Febuary 1996.
9]

Bar-Cohen and Yoseph,.
10]

C. Roentgen, ""About the changes in shape and volume of dielectrics caused by electricity"," *G. Wiedemann Ed. Annual Physics and Chemistry Series*, vol. 11, pp. 771-786,
11]

1880.

12] R. Kornbluh, R. Pelrine, Q. Pei, S. Oh, and J. Joseph, ""Ultrahigh strain response of field-actuated elastomeric polymers" Smart Structure and Materials 2000," *Electroactive Polymer Actuators and Devices*, vol. 3987, pp. 51-64, 200.

13] A. Ashley, "Artificial Muscles," *Scientific American*, pp. 52-59, 2003.

14] R. Pelrine et al., "Dielectric Elastomer Artificial Muscle Actuators: Toward Biomimetic Motion," *Smart Structures and Materials*, vol. 4695, pp. 126-137, 2002.

15] N. Vandesteeg, P. Madden, P. Anguetil, and I. Hunter, "Synthesis and characterization of EDOT-based conducting polymer actuators," *Proceedings of SPIE*, vol. 5051, pp. 349-356, 2003.

16] J. D. Madden, P. G. Madden, P. A. Anquetil, and I. W. Hunter, "Load and time dependence of displacement in a conducting polymer actuator," *Materials Research Society Proceedings*, vol. 698, pp. 137-144, 2002.

17] T. E. Herod and J. B. Schlenoff, "Doping induced strain in polyaniline: stretchoelectrochemistry," vol. 5, pp. 951-955, 1993.

18] M. Shahinpoor and K.J. Kim, "Ionic polymer-metal composites - I. Fundamentals," *Ionic polymer-metal composites - I. Fundamentals*, pp. 819-833, 2001.

19] K. J. Kim and M Shahinpoor, *Development of three dimensional ionic polymer-metal composites as artificial muscles*., 2002.

20] S. Nemat-Nasser, *Journal of Applied Physics*, vol. 92, pp. 2899-2915, 2002.

21] Wise Geek. Wise Geek. [Online]. http://www.wisegeek.com/what-is-a-bowden-cable.htm

22] Butera, 6,732,512 B2, 2004.

23] R. Ekkelenkamp, R. Kruidhof, F.C.T. van der Helm, H. van der Kooij J.F. Veneman, "Design of a Series Elastic- and Bowdencable-based actuation system for use as torque-actuator in exoskeleton-type training," in *9th International Conference on Rehabilitation Robotics*, Chicago, IL, 2005.

Frank L. Hammond III, Robert D. Howe, Marko B. Popovic Ignacio Galiana,
24] "Wearable Soft Robotic Device for Post-Stroke Shoulder," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Vilamoura, Algarve, Portugal, 2012.

M. Murphy, K. Sunnerhagen, B. Johnels, and C Willen, "Three-dimensional
25] kinematic motion analysis of a daily activity drinking from a glass: a pilot study," *Journal of NeuroEngineering and Rehabilitation*, vol. 3, no. 18, 2006.

L. A. Harvey, J. Batty, R. Jones, and J. Crosbie, "Hand function of C6 and C7
26] tetraplegics 1-16 years following injury," vol. 39, no. 1, pp. 37-43, January 2001.

Saxena and Shalini, "An EMG-controlled grasping system for tetraplegics," *Journal*
27] *of rehabilitation research and development* , vol. 32, no. 1, p. 17, Febuary 1995.

Diane Barus and Scott H. Kozin, "The Evaluation and Treatment of Elbow
28] Dysfunction Secondary to Spasticity and Paralysis," *Journal of Hand Therapy*, vol. 19, no. 2, pp. 192-205, June 2006.

Yiu-Ming Wong and Gabriel Y. F. Ng, "Surface electrode placement affects the
29] EMG recordings of the quadriceps muscles," *Physical Therapy in Sport*, vol. 7, no. 3, pp. 122-127, August 2006.

R Merletti, "Electrically evoked myoelectric signals," *Critical reviews in biomedical*
30] *engineering*, vol. 19, no. 4, p. 293, January 1992.

Edward A. Clancy, "Private Interview," September 2012.
31]

Raez, Hussain, and Mohd-Yasin, "Techniques of EMG signal analysis: detection,
32] processing, classification and applications," *Biol Proced Online*, vol. 16, no. 8, p. 163, October 2006.

Scott Day, "Important Factors in Surface EMG Measurement," Calgary, AB,.
33]

N. M. Sobahi, "Denoising of EMG Signals Based on Wavelet Transform," *Asian*
34] *Transactions on Engineering*, vol. 1, no. 5, November 2011.

Seedahmed S. Mahmoud, Zahir M. Hussain, Irena Cosic, and Qiang Fang, "Time-
35] Frequency Analysis of Normal and Abnormal Biological Signals," *Biomedical Signal Processing and Control*, vol. 1, no. 1, pp. 33-43, 2006.

Shahjahan Shahid, Jacqueline Walker, Gerard M. Lyons, Ciaran A. Byrne, and

36] Anand Vishwanath Nene, "Application of Higher Order Statistics Techniques to EMG Signals to Characterize the Motor Unit Action Potential," *IEEE Transactions on Biomedical Engineering*, vol. 52, no. 7, July 2005.

R. Thorsen, M. Ferrarin, R. Spadone, and C Frigo, "Functional Control of the Hand
37] in Tetraplegics Based on Residual Synergistic EMG Activity," *Artificial Organs*.

Anne M. Bryden, William D. Memberg, and Patrick E. Crago, "Electrically
38] stimulated elbow extension in persons with C5/C6 tetraplegia: A functional and physiological evaluation," *Archives of Physical Medicine and Rehabilitation*, vol. 81, no. 1, pp. 80-88, January 2000.

K.]Kiguchi and Qilong Quan, "Muscle-model-oriented EMG-based control of an
39] upper-limb power-assist exoskeleton with a neuro-fuzzy modifier," in *IEEE World Congress on Computational Intelligence*, 2008, pp. 1179-1184.

C. Martelloni, J. Carpaneto, and S. Micera, "Characterization of EMG Patterns From
40] Proximal Arm Muscles During Object- and Orientation-Specific Grasps," *Biomedical Engineering*, vol. 56, no. 10, pp. 2529-2536, October 2009.

Norman S. Nise, *Control Systems Engineering. 6th ed.*: Wiley, 2010.
41]

Kyoungchul Kong, Joonbum Bae, and M. Tomizuka, "Control of Rotary Series
42] Elastic Actuator for Ideal Force-Mode Actuation in Human–Robot Interaction Applications," *Mechatronics, IEEE/ASME Transactions on*, pp. 105, 118, 2009.

J.F. Veneman, R. Ekkelenkamp, R. Kruidhof, F. van der Helm, and H. van der Kooij,
43] "Design of a series elastic- and Bowden cable-based actuation system for use as torque-actuator in exoskeleton-type training," *Rehabilitation Robotics, 2005. ICORR 2005. 9th International Conference on* , pp. 496, 499, 2005.

R. F. Chandler, C. E. Clauser, J. T. McConville, H. M. Reynolds, and J. W. Young,
44] "Investigation of Inertial Properties of the Human Body," *AIR FORCE AEROSPACE MEDICAL RESEARCH LAB WRIGHT-PATTERSON AFB OH*, 1975.

et al. Dipietro, *IEEE Transactions on Neural Systems and Rehabilitation*
45] *Engineering*, pp. 325-334, 2005.

K. Englehart, B. Hudgins P. Parker, "Myoelectric signal processing for control of
46] powered limb prostheses," *Journal of Electromyography and Kinesiology*, pp. 541-548, December 2006.

Anne M. Bryden, Kevin L. Kilgore, Benjamin B. Lind, and David T. Yu, "Triceps
47] denervation as a predictor of elbow flexion contractures in C5 and C6 tetraplegia," *Archives of Physical Medicine and Rehabilitation*, vol. 85, no. 11, pp. 1880-1885, November 2004.

HowToMedia Inc. Muscles of the Arm and Hand. [Online].
48] http://www.innerbody.com/anatomy/muscular/arm-hand

D. Stauss, M. DeVivo, D. Paculdo, and R Shavelle, "Trends in Spinal Cord Life
49] Expectancy," *Arch Phys Med Rehabil*, vol. 87, no. 8, pp. 1079-1085, 2006.

Hinged Shoulder Braces.
50]

Che-Wei Wang, "Soft Pneumatic Exoskeleton,".
51]

Mao, Ying, and Agrawal, "Design of a Cable-Driven Arm Exoskeleton (CAREX) for
52] Neural Rehabilitation," *Robotics, IEEE Transactions on*, vol. 28, no. 4, pp. 922-931.

Caihua Xiong, Jiang Xianzhi, Sun Ronglei, XiaoLin Huang, and Youlun Xiong,
53] "Industrial Robot: An International Journal," *Emerald*, 2009.

Günter Hommel, Sheng Huanye, and Christian Fleische, "Embedded Systems -
54] Modeling, Technology, and Applications," in *7th International Workshop*.

E. E. Cavallaro, J. C. J. Rosen, and S. Burns Perry, "Real-Time Myoprocessors for a
55] Neural Controlled Powered Exoskeleton Arm," *IEEE Transactions on Biomedical Engineering*, vol. 53, no. 11, pp. 2387-2396, 2006.

Craig Carignan, "Design of an Arm Exoskeleton with Scapula Motion for Shoulder
56] Rehabilitation".

MD, FACP, FACR William C. Shiel Jr. MedicineNet.com. [Online].
57] http://www.medicinenet.com/electromyogram/article.htm