Mechanical EMG-Controlled Hand v2.0

Senior Design Project
# Table of Contents

Team Members 4  
Abstract 5  
Motivation 5  
Objective 5  
System Overview 7  
Mechanics 7  
  - Different Finger Mechanisms 7  
  - Open Hand Project 9  
  - Finger Motors 9  
  - Hand Improvements 11  
  - Wrist Design Ideas 12  
  - Wrist Design, Motors, and Gears 13  
  - Elbow Joint and Padding 16  
  - Forearm Design 18  
  - Grip 19  
Controls 20  
  - Version 0: Systems 20  
  - Version 1: First Prototype 20  
  - Version 2: Full Usage 21  
  - Version 2.5: EMG Automatic Calibration 23  
  - Version 3: Wrist Actuation 23  
  - Control Panel 24  
EMG Data Acquisition System 25
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analog Circuit Design Specs</td>
<td>26</td>
</tr>
<tr>
<td>Op-Amp Specs</td>
<td>27</td>
</tr>
<tr>
<td>High Pass Filter and Movement Analysis</td>
<td>27</td>
</tr>
<tr>
<td>Nyquist Filter and Sampling Rate</td>
<td>28</td>
</tr>
<tr>
<td>Smoothing algorithm</td>
<td>28</td>
</tr>
<tr>
<td>Placement of Electrodes</td>
<td>31</td>
</tr>
<tr>
<td>Digital Potentiometer</td>
<td>32</td>
</tr>
<tr>
<td>Automatic Calibration</td>
<td>33</td>
</tr>
<tr>
<td>Circuit Design</td>
<td>33</td>
</tr>
<tr>
<td>Power Budget</td>
<td>33</td>
</tr>
<tr>
<td>Battery/Power Supply</td>
<td>34</td>
</tr>
<tr>
<td>Motor Drivers</td>
<td>36</td>
</tr>
<tr>
<td>EMG Analog Input</td>
<td>37</td>
</tr>
<tr>
<td>Soft Power Switch</td>
<td>38</td>
</tr>
<tr>
<td>Haptic Motor Control</td>
<td>39</td>
</tr>
<tr>
<td>PCB Development</td>
<td>40</td>
</tr>
<tr>
<td>Hand Connector Board and Wire Harness</td>
<td>40</td>
</tr>
<tr>
<td>Console PCBs</td>
<td>40</td>
</tr>
<tr>
<td>Choice of PCB Design House and Hardware Considerations</td>
<td>42</td>
</tr>
<tr>
<td>Graphical User Interface</td>
<td>42</td>
</tr>
<tr>
<td>SSD1322 Driver</td>
<td>42</td>
</tr>
<tr>
<td>Graphics</td>
<td>42</td>
</tr>
<tr>
<td>Event-driven graphic framework</td>
<td>43</td>
</tr>
<tr>
<td>Inertial Measurement Unit</td>
<td>45</td>
</tr>
<tr>
<td>Future Works/Improvements</td>
<td>46</td>
</tr>
</tbody>
</table>
Team Members

D.J. Fuller  
Mechanical Design  
djfuller@ucsc.edu  
D.J. is a fourth year student majoring in Bioengineering with a concentration in rehabilitation. His primary task is to design the mechanical hand, wrist, and forearm while attempting to make the prosthetic as humanlike as possible. He has experience with CAD, mechatronics, sensor design, C language, and logic design.

Robert Hoffman  
Controls Systems  
rmhoffma@ucsc.edu  
Robert is a fifth year Computer Engineering major with a focus in robotics and control systems. His involvement in the project involves the control and software development in regards to actuation. He is an experienced C programmer, with interests in digital logic, automation, and embedded systems.

Patrick Ayers  
Power and PCB layout  
payers@ucsc.edu  
Patrick is a fifth year student of Computer Engineering with a focus on robotics and control. His task is develop the schematics and route the printed circuit boards to be used in final system design. He will design the power supply and motor driver circuits as well as choose components to be used on the official bill of material. He has experience in circuit analysis, digital logic, embedded systems programming and PCB design.

Andre Nguyen Van Qui  
Sensors and User Interface  
anguyenv@ucsc.edu  
Andre is a fourth year Robotics Engineering student. His tasks in this project are improving sensor interfacing designs, setting communication protocols, and designing graphical interface. He is experienced in C embedded programming, C++ graphics programming, algorithm designs, electronics, and robotics.
Abstract

We aim to create a prosthetic hand with the main purpose to mimic a human hand. Our goal is to create an expressive prosthetic with a focus on dexterity of fingers rather than a specific utility.

Motivation

Our group is made up of individuals interested in the betterment of man by application of technology. Our aim is that our research will help to provide a better and more affordable prosthetic hand for amputees or otherwise. Furthermore, we are all interested in the field of robotics, and even animatronics, so we are enthusiastic to be able to work on a project that focuses on these fields.

Objective

We want to continue the original MECH team’s goal of producing an inexpensive, motorized prosthetic, but also incorporate new systems to make the system more robust. We want to make our prosthetic comfortable while retaining a utility and not being overtly obvious.

In order to make the hand module as lifelike as possible, the mechanical hand will contain five individually actuated fingers. The hand will be able to perform more than just open and close, but it will be able to do up to eight different gestures. The eight most used hand gestures that consist of 85% of all daily gestures, known as the eight canonical gesture, will be programmed into our system allowing the user to choose their desired gesture/grip [1]. Furthermore, in order to perform some of these gestures, the mechanical hand will have a rotating thumb.

The system will provide the amputee with two forms of feedback. To begin with, the arm
portion will contain a small haptics to provide tactile feedback the fingers close upon an object. The haptics will initially send a vibration proportional to how strong the fingers are closed on the object. Current sensors for each finger motor will be used as the feedback to tell the user when the object is grasped. Next, the system will possess an organic light emitting diode (OLED) display controlled by buttons on the prosthetic arm. The GUI will display the current gesture that the hand is programmed to, and the next gesture that the user wishes to select. Also, the battery life of the system will be shown in the upper right corner.

We are continuing to use Electromyography (EMG) as a means of control in the system, but are redesigning most of the sensor systems. The speed of the fingers of the mechanical hand will be controlled by the different signals coming from the EMG sensor. When the user flexes with more force, the fingers will travel faster. When the user flexes with less force, the fingers will travel slower. The grip strength will be controlled by how long the amputee flexes his muscle (connected to the EMG sensor). Initializing the desired gesture and resetting the hand to its default position will be controlled by two muscle flexions; the first to gesture and the second to reinitialize. There will be a locking option on the hand which will allow the hand to remain closed and the fingers to stay in position regardless of muscle activity from the EMG sensor.

In order to make the prosthetic limb as comfortable as possible, the system will be made lighter than most prosthetic hands. This will be accomplished by limiting the total size of the hand and arm, as well as the electronics and internal PCBs. Also, mostly every part to construct the hand will be 3D printed with PLA plastic.

A wrist unit will be added to the hand module to allow the amputee to control the hand with a wider range of movements. The wrist unit will be able to perform two degrees of freedom. The first degree of freedom consist of flexion and extension. the second degree of freedom will be pronation and supination. In order to control the wrist movement, a gyroscope and an accelerometer will be mounted inside of the hand unit. With the addition of these sensors, the system will have an auto leveling option which will allow the hand to be locked in a balanced position.

The system will be able to last for a full days usage on power. To help with power efficiency, the wrist motions will be actuated with worm drives, allowing the hand to lock into place while deactivating the motors. Also, the fingers will be actuated utilizing a pulley system which will allow the fingers to stay tense without the motors constantly running. the current sensing on the motors will communicate with the system to turn the motors off when the fingers are stalled on an object.
System Overview

Mechanics

In order to design and build this mechanical arm, sections of the arm were focused on in one step at a time. Starting with the fingers, the hand was built and made durable. Next, the wrist mechanism was designed. Instead of continuing to work up the arm on to the forearm, the elbow joint was built next. Lastly, the forearm was designed to contain the battery, PCB stack, and OLED screen, and it connected the wrist to the elbow joint.

Different Finger Mechanisms

There are various types of mechanisms we could have implemented to control the finger flexion and extension. We first looked at the bar link technique and then the cable/pulley system.

Bar Link:

The bar link system has been used by many other prosthetic hands. A simple example is seen in Fig. 3 [3]. A bar is connected from the palm to the end segment. Another bar can be added to the end segment if one wishes to give the finger all three joints instead of two. There is a bar on the other side of the finger also. If the distance between the axis of rotation and the bar is equal in both segments, the bar causes an equal rotation in the end segment as the main segment is rotated. Each segment in this finger will rotate 90 degrees. More complex hands have this bar linkage system more compact and inside of the finger so the bars are not visible. The bar must be sturdy because it acts in tension and in compression. The procedure to install this system is fairly simple and it can be reliable and robust. The picture of the hand shows a bar linkage design doing the pinch grip. The two segments on the pointer finger move synchronously in a 1:1 ratio.
Cable Pulley System:
Given how small and light the cable system is, it can handle very high loads. There are also many other examples of prosthetic hands that use this technique to actuate the fingers. The motion of each preceding segment of the finger will not be limited by the previous segment. The system consumes very little space in the hand, and it achieves constant pulling force. A way to implement the system is to hold the cable tight by a compression spring tensioner. If the finger is bumped by the outside environment, instead of a tendon breaking, the compression spring will flex. Due to the cabling, setting up this system can be difficult and tensioning high strength cables in small areas may be hard to manage. The finger that uses the cables as tendons will adapt fit to any form that is placed in the hand. An example of a prosthetic hand that is tendon based with cables is the SmartHand, as seen below [4]. The image on the right shows the difference between a bar linked hand and a hand using the pulley system [5].

![Fig. 3: Mechanical Fingers Using Bar Linkage System](image)

![Fig. 4: SmartHand Using the Cable System](image)

![Fig. 5: Difference Between the Two Finger Driving Mechanisms](image)
Top = Cable System
Bottom = Bar Linkage System
We decided that we wanted our mechanical hand to utilize the cable system. The small and lightweight mechanism satisfied our requirements. Also, we benefited from the fact that the fingers would be able to wrap around any awkward object, if small enough. We then came across an open source design that would fit well with our apomorphic requirement.

**The Open Hand Project**

We came across an open source 3D printed design which contained CAD files for a mechanical hand, known as the Open Hand Project [5]. Not only did the Open Hand Project utilize the cable system, they incorporated a rotatable thumb, controlled by a small servo motor. This build of the hand would be able to perform the eight canonical postures. The Open Hand Project provided STL files for the hand and a set of directions on how to build the mechanical hand. We wanted to start with building the Open Hand Project prosthetic hand to test its functionality to see what improvements could be made on it to fit our needs. An image of the prosthetic hand found on the Open Hand Project is seen below.

![Fig. 6: Open Hand Project](image)

**Finger Motors**

Prior to constructing the hand, we had to choose motors which would work in the system and provide enough torque. The cabling system used in the Open Hand Project is fairly complex. To find the amount of torque we needed for our motor to actuate the fingers, we simplified the mechanism to a simple pulley. We found that for a properly functional prosthetic hand, the maximum finger force of the index finger should be 65 N. To find the torque that our motors would need,
we had to find the amount of torque required on the pulley. We started with this calculation:

<table>
<thead>
<tr>
<th>Variable</th>
<th>$t_f$</th>
<th>$F_f$</th>
<th>$L_f$</th>
<th>$T_c$</th>
<th>$R_j$</th>
<th>$t_p$</th>
<th>$R_p$</th>
<th>$t_m$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Meaning</td>
<td>torque on finger</td>
<td>force of finger</td>
<td>length of finger</td>
<td>tension on cable</td>
<td>radius of joint</td>
<td>torque on pulley</td>
<td>radius of pulley</td>
<td>torque of motor</td>
</tr>
</tbody>
</table>

$$t_f = F_f \times L_f$$

**where:** $F_f = 65.0 \text{ N}$ and $L_f = 103.2 \text{ mm}$

$$t_f = 65.0 \text{ N} \times 103.2 \text{ mm} = 6.7 \text{ Nm}$$

$$T_c = t_f/R_j$$

**where:** $R_j = 13 \text{ mm}$

$$T_c = 6.7 \text{ Nm/13 mm} = 515.9 \text{ N}$$

$$t_p = T_c \times R_p$$

**where:** $R_p = 9.5 \text{ mm}$

$$t_p = 515.9 \text{ N} \times 9.5 \text{ mm} = 4.9 \text{ Nm}$$

We researched different motors and found a motor that would work well for our application, the Pololu 115:1 metal gearmotor [7]. Next, we solved for the required torque using these variables ($G$=gear ratio=115, $\eta$=efficiency=80%):

$$t_m = t_p/(G \times \eta)$$

$$t_m = 4.9/(115 \times 80\%) = 0.05 \text{ Nm}$$

0.05 Nm would be the maximum amount of torque needed. The stall torque of the 115:1 Pololu motor is 0.25 Nm. The motor would have plenty of torque to drive the motors at our specifications.

The CAD used to build the Open Hand Project prosthetic was done in Blender. Our first step was to edit the motor housing so that it would fit the motors that we chose.
We were initially worried about the fingers not being able to hold a position if the motors were deactivated. After printing the hand out, implementing our finger motors, and putting the hand together, we tested to see how the fingers acted. We activated all of the finger motors and closed the fingers around small objects. The hand was able to hold onto small objects, without continuously running the motors, which would help with power efficiency.

**Hand Improvements**

There were a few mechanical bugs with the Open Hand Project that needed immediate fixing to continue testing our mechanical hand. The spool on the motor shaft was too loose and kept slipping which made the finger unable to move. The set screw on the spool was off center which caused an unbalanced rotation, helping the slippage. Next, when the voltage on the motors were reversed so that the figures could open, several times the fingers would hyperextend, causing the cable to snap. In the Blender files, the inner diameter of the spool was extruded inwards to make a tighter fit around the motor shaft. Also, the bridge across the motor mount which prevents the finger from hyperextending was slightly lowered to help decrease the chances of the finger from sliding beneath it. The main reason the fingers were hyperextending was because the dowel pins inside the knuckles were free to move. We made an extra edge within the dowel pin slot to help secure its placement. When each finger was put together, we used a heat gun to push together the knuckle housing so the dowel pin would not slip backwards, making hyperextension impossible. The next figure shows an image of a final finger. Here, you can see how the cable system works: the cable runs around the spool on the motor shaft, then separates to the two pulleys attached to compression springs as tensioners. One end is fed through the top of the finger and one end is fed through the bottom. The ends are crimped at the end of the finger. In the figure, the finger stays tense in this closed position. Also, notice how the end of the motor housing is pushed inwards to resist the slipping of the dowel pin which used to cause the hyperextension when the finger was opened to its maximum. The motors can be stalled out at both ends of the full range of movement.
Wrist Design Ideas

For our wrist design, we wanted to be able to create a simple design that could perform the flexion and rotation movements. We were aiming to make the prosthetic arm 7" in length in order to match our test subject’s active arm. We wanted the wrist module to be as compact as possible. Also, the two motors that would actuate the wrist would be connected to worm drives so that the hand placement could be locked into place. The wrist shown in Fig. 9 shows a two degree of freedom mechanical wrist that prompted our ideas for a design [8].

The flexion and extension axis is driven by a worm drive, which was a requirement for our system. For the pronation and supination movements, the wrist appears to be driven by a planetary gear. We did enjoy the extension motor assembly, however the planetary gear drive looked somewhat difficult to implement and
this does not ensure that the hand remained in a position if leveled to a certain angle. The orientation of the rotation motor would not allow us to use a worm drive because the motor would have to be perpendicular to the axis of rotation. We then thought about how a servo pan tilt works, shown below [9].

![Pan Tilt Servo](image)

**Fig. 10: Pan Tilt Servo**

Servos only have a limited amount of range, and they must be driven constantly in order to maintain position. We were inspired by the pan tilt design and wanted to convert it so that the two rotations would be driven by two DC motors, using worm drives. Most worm drives are made for DC motor shafts and this would allow us to be able to use worm drives with a full range of rotation.

**Wrist Design, Motors, and Gears**

By mounting the flexion/extension motor onto a rotation plate controlled by the rotation motor, we were able to succeed in driving each degree of freedom with a worm drive. Initially we thought we required room for encoders at the end of each motor to be able to communicate the hand’s current position. Instead, the gyroscope and accelerometer in the hand provide information needed to adjust to the desired position. This would save space in the width and length of the wrist. Fig. 11 shows the first revision of the wrist design done in SolidWorks.
From the end of the hand mount to the back end of the last motor, we were able to limit the length of the module to 3.4”. We had around 4” for the rest of the electronics to be placed in the forearm portion of the prosthetic limb. Fig. 12 shows the flexion/extension drive and Fig. 13 shows the rotation drive more clearly.

A gear train was implemented with the rotation worm drive in order to limit the width of the wrist. If the worm gear were directly connected to the rotation shaft, the wrist would have been too wide to appear as a normal human wrist. The train adds only a very small amount of weight, because the extra gears were 3D printed. We sacrificed the length of the wrist so that the wrist could fit into our desired width proportion. The gear train was also added on the left-hand side of the wrist to even out the weight distribution. This wrist is designed for a left hand and the hand’s center of mass favors the right side of the hand because of the servo controlling the thumb.
In order to find how much torque would be needed for each movement of rotation, free body diagrams of each degree were made. The hand weighed approximately 1 lb, or 4.5 N. The maximum amount of torque acting on the flexion/extension gear occurs when the hand is held straight out. For the rotation gear, the maximum torque occurs when the hand is completely flexed or extended. Also, because the rotation axis was placed farther back in the wrist, about 0.06 lb must be added to the weight of the hand for the rotation plate and rotation shaft. The free body diagrams are shown below, where $r$ is the perpendicular distance from the worm gear to the weight of the hand at its center of mass.

Using the equation for the torque, $t = F \cdot r$, the torque on the axis of rotation for flexion/extension is 0.36 Nm (51.0 oz-in) and 0.37 Nm (52.4 oz-in) for pronation/supination. We were able to find a worm drive set from Sterling Instrument which was very compact and lightweight. The worm is made of nylon while the worm gear is made of acetal. The worms used in the original MECH were 20.6 g, while the ones that used for MECH v2.0 were 0.6 g. An external 15:1 gear ratio would be used to reduce the required torque.
Taking the calculated torque from the pronation/supination rotation, the required torque for the motor would be the torque divided by the gear ratio: \[ \frac{52.4 \text{ oz-in}}{15} = 3.5 \text{ oz-in} \]. Because the speed of the wrist was not a requirement for our system, to be safe, we chose a Pololu 210:1 high power micro gearmotor with 50 oz-in stall torque at 6V for the actuation of both degrees of freedom. The micro gearmotors have dimensions of 0.94" x 0.39" x 0.47". Their small size made them an excellent choice for the limited space in the wrist design.

**Elbow Joint and Padding**

A difficult part of this project was to connect the arm to our test subject, as comfortably as possible. We wanted to be able to 3D print the custom cup portion of the prosthetic base, but designing this to fit from scratch in SolidWorks would be almost impossible. We were given our test subject’s pre-existing prosthetic base in order to make a 3D scan of the elbow joint. At first, we tried using a phone application called 123DCatch to scan the prosthetic. The application proved to be difficult to use, as the scans were missing several portions of the arm when converted to a STL file.

Next, we tried using a Xbox Kinect to do a 3D scan of the prosthetic base. A program called Skanect was used to capture the full 3D model of the prosthetic. To get the best results, the prosthetic arm was propped up on a rotating chair, while the Kinect was placed stationary on a desk, aiming toward the chair. The chair was rotated slowly as Skanect gathered its data from the Kinect. In order to scan the inside of the elbow joint, the Kinect was slowly lifted and moved to film the inner angles of the cup portion. This process took a few attempts for good data, but eventually the program produced a usable 3D mesh of the real prosthetic. Skanect gave options to fill in any holes that were not read properly in the initial scan. Additionally, the part was smoothed several times within the program. The next issue was figuring out a way to edit the 3D mesh to that the base could be cut and connected to the rest of the arm.

A crucial tool in SolidWorks that helped the manipulation of the prosthetic base was a wizard called the Mesh Prep Wizard. After converting the Skanect STL file to a SolidWorks part, there was no way to edit the part. Using the Mesh Prep Wizard allowed the part to be edited. The wizard smoothed out the part even more, and it automatically created small faces to form the complex part. Once the base went through the Mesh Prep Wizard, the part was cut so that only the elbow joint would be used. Being able to edit the elbow joint made it much more simple to connect the joint with the forearm, once the forearm was created in SolidWorks.
After this part was 3D printed, our test subject tried it on to test the comfortability. The PLA plastic was not great for comfort, and he expressed that some portions of the cup were too tight. Looking into how other prosthetics are mounted comfortably, we decided to use silicone. In order to utilize a silicone padding, much of the inside of the elbow joint part had to be sanded out to make room for the thick padding. The soft and stretchable silicone padding is shown in Fig. 17. The silicone was created using a compound that vulcanizes at room temperature. Because our test subject was not available for the amount of time it would take to create a silicone mold of his arm, clay was pressed into the cup to take the form of our test subject’s stump. Then, the clay was replaced and the silicone was then brushed onto the clay, making the mold. The silicone padding made the arm much more comfortable. The silicone was only $20 for a pound making it very affordable and could be used to create a custom mold with a patient present..
Forearm Design

The goal of the forearm design was to connect the wrist module to the elbow joint as with a small, lightweight form resembling an actual human wrist. The OLED display had to be oriented on the forearm in such a way that it would be easy for the user to view with the arm in a resting position. Special considerations to the dimensions of the inner console printed circuit boards (PCBs) were taken when designing the shape and inner dimensions of the forearm housing.

To begin, mock up parts of the electronics were created in SolidWorks so that the forearm could be built around those dimensions. Fig. 18 shows the forearm design as well as the inner electronics. The green object is the battery, the purple object is the PCB stack, and the black object is the OLED screen. There are two halves that are put together to enclose the electronics and form the forearm. The battery, PCB stack, and OLED screen are able to slide into their positions without any other connection to keep them in place. Also, there are tab-and-slots at each end of the forearm to ensure a tight connection between the different arm modules. All of the machine screws used to connect the two halves of the forearm are recessed so that they do not scrape on any outside surfaces when in use.
Holes were cut in order to secure the haptic feedback motor and to connect the EMG jack into the board. A hole was cut for the power switch to snap into place along the side of the forearm which is also recessed into the arm so that the switch can not be flipped by outer flat surfaces.

It was crucial that the OLED screen was angled so that the user could easily see it and use the buttons at all times. From above the top side of the forearm, like the angle of a wristwatch, the screen was angled inward towards the user’s body at 79 degrees. This allows the user to see the OLED screen even when the arm is resting at the side of the user’s body. When assembling the forearm, the red buttons are placed inside their slots first, and they are held in place by the pressure of the buttons on the PCB board. Each button has a ridge on the bottom, so that the buttons can not slide out of the top of the arm.

**Grip**

We analyzed the most essential points of contact for gripping gestures. For cylindrical gesture, they are the tip of fingers and edge of the palm toward fingers. For lateral gestures, they are tip of the thumb and the side surface of the index finger. We cut nitrile laboratory gloves and glued them on these surfaces. Experiments were done to test the grip strength along with the grip material. We tried with some common household objects such as a glass cup, an aluminum water bottle, a porcelain mug, and a plastic water bottle. Our hand is able to grasp statically at least 1.5 lb for all these objects without constantly applying power to motors. The 1.5 lb weight is the weight of a water bottle full of water. With motor being powered, heavier objects can be grasped steadily without slipping. However, the full capacity in terms of weight was not tested.
Controls

Version 0: Systems

The control systems, and programming started with the research and selection of the system we would be using. We compared a few chips, and borrowed some market research from the previous MECH team to come to the decision to use the Microchip PIC24. This decision turned out to be ill informed for a few reasons: the chip had no (easily) available dev boards, and it lacked the amount of IO pins used in our application. After not being able to properly develop on this system, a quick switch was made to the PIC32. This change was beneficial in that the UNO32 is a common and readily available development kit and met our specifications nicely.

The next issue that needed to be resolved was the lack of hardware pulse width modulation pins available with the UNO. It was initially proposed to use a PWM expanding chip, specifically the TLC5940. This was rejected after research revealed that the output of this pin was not compatible with the H-bridges our project used. The solution was to use a software generated pulse signal. The
concept is simple: using a timer interrupt, run a looping counter. The granularity of this signal is the timer’s speed, and the period is the point at which the counter rolls over. Using another variable value representing the Duty Cycle, we can generate a partial signal. One problem with the initial implementation was finding a high enough clock frequency that the module would be able to operate at, when the frequency was lower the motors would visibly and audibly start and stop. The following pseudo code illustrates this concept:

```c
ISR TIMER(void) {
    if(PWMcounter >= Period) PWMcounter = 0
    else increment PWMcounter
    if(PWMcounter < DutyCycle) set pins high
    else set pins low
}
```

This module was developed simultaneously to the first hardware test bench: a separate program that could move the fingers back and forth to demonstrate movement and basic control. The motors move at an acceptably slow rate when at 100% duty cycle, so the first program simply flipped output latch bits to drive the motors. This program also used a millisecond timer to control the position of the fingers since the amount of time it takes them to curl is very close to 1 second.

The most pertinent feature of the first program is the implementation of a serial monitor as a control panel. This was meant to model the final product of having a panel attached to the forearm and has persisted as a design feature up until this point. Using the serial terminal also was helpful for debugging and retrieving sensor data.

**Version 1: First Prototype**

The next goal in programming was to actually form all 8 gestures and report sensor data. This used the platform gesture (all fingers open and thumb flattened) as the ‘base’ gesture. That is, before forming any gesture the hand would have to first form the platform gesture first. This served to illustrate the basic movements in sync with one another, and the ability of the mechanical model to perform all the desired gestures. At this point the movement was still dictated by timers and output pins being set high. Below is a flowchart of this iteration of the program:
This stage introduced RC servo control and AD analysis. The RC servo library is a standardized PWM signal that uses the duty cycle as a variable that affects the angle that the servo maintains. The AD pins were used to measure the voltage across a shunt resistor attached to each H-bridge, allowing for preliminary measurements that would eventually form the current sensing. In this state, a very basic current sense was possible using a single threshold value that would stop an individual motor. This was accompanied by a startup timer which let the sensor system ignore the initial stall current when the motor started.

This was also repeated with the EMG sensor, which was then tied to the hook gesture specifically. This was implemented using a timer interrupt to sample periodically, which then applied a digital filter to it. After gathering a few measurements we determined a suitably high threshold that would toggle a gesture. There were a few holes in this program implementation, mainly that only 1 gesture was available with the EMG sensing. Another problem was sustained signals, which would cause rapid back and forth toggling. This also added an air of unpredictability to the sensors since it sampled faster than the range of perception. This stage served primarily as a proof of concept and an early warning for possible problems we could have encountered later on.

**Version 2: Full Usage**

The next stage in the program’s evolution was to implement the PWM module within the gesture software. The implementation was simple enough, but produced a bug in which the servo, which also used a timer based interrupt, would sometimes twitch. It was determined that the cause of this was the interrupts interrupting one another, lengthening the duty cycle of servo inadvertently. This was resolved by rolling the servo control ISR into a function called by the higher speed PWM timer interrupt. From there both modules functioned correctly.
Having the PWM working caused a different problem however, in the form of the current sense no longer operating correctly. This was due to the pulsing of the motor causing it to stall lightly every time it was activated. This signal required some analysis to be useful in our design. To begin with, debouncing was implemented: after passing a voltage threshold the system would wait a specified amount (slightly longer than the PWM period) and then check to see if the current sensor was still high. Secondly, hysteresis was applied. When the sensor was checked a second time, it became necessary to lower the threshold. Lastly, a safety timer was implemented after damaging the fingers a few times from driving them into hyperextension of over curling. This was simply a timer that activated when the motors did, which would deactivate said motor after a few seconds had passed to ensure that they would not move to the point of damaging themselves.

With the EMG it became necessary to use a new system to emphasize the variety of signals possible from the sensor. This was done by setting a few thresholds that would affect the motor duty cycle. In addition, when the EMG stopped being driven the hand was to stop. This change required a much larger structural change to the program itself.

It started with defining a new way of choosing gestures, with the goal of eliminating platform as an intermediary gesture. To do this we divided the eight gestures into five setup phases. The setup phase filled the space formerly taken by the platform gesture, except it was relevant to the motion related to it rather than being a blank slate. The gestures were sorted as such:

<table>
<thead>
<tr>
<th>Platform</th>
<th>Opposed</th>
<th>45</th>
<th>45 split</th>
<th>Closed</th>
</tr>
</thead>
<tbody>
<tr>
<td>platform</td>
<td>cylindrical</td>
<td>spherical</td>
<td>tripod</td>
<td>lateral</td>
</tr>
<tr>
<td>hook</td>
<td>tip</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>point</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The reason these were divided as such were so that when the EMG was activated the hand could simply close or open the active fingers.

The next step was devising a way that transitions between gestures could be made. We decided that whenever a new gesture was selected the hand would move to the setup of the new gesture, and it would be able to do this most efficiently if we kept track of each finger’s individual position. This was done by replacing the finger stop function with a function that checked the direction and marked that it was closed if it was curling before it stopped, or open if it was extending.

The gesture motion function was simple following this, which ever finger that hand not been modified by set up and were active in the gesture would either open or close. We kept a static variable in the top of this function that would toggle each time it was called in order to determine whether the hand was opening or closing. This function also had a brother function that served only to
turn the active fingers off while moving. This was necessary as beforehand, any
time the EMG was below threshold it was impossible to move the hand at all, so
the signal had to be purely associated within gesture activation.

The resultant program after implementing these changes allowed for a
greater depth of control using the EMG than we originally anticipated. Not only
could the user control the speed at which the fingers moved, but they could also
control when the fingers stopped which makes this a more delicate hand. We also
deided to removing current sensing while the fingers were moving due to EMG
control. This means that as long as a user is flexing the motors will continue to
spin so as long as the user wants to flex, the grip will tighten. This style of control
allows the hand to lift heavier objects than originally planned for.

**Version 2.5: EMG Automatic Calibration**

The next stage of the project was making it more immediately adaptable. The EMG sensor itself needed some means of being adjusted since different users
have different magnitudes of signal when they used the sensor. This was
implemented using a digital potentiometer in conjunction with an
instrumentation amplifier to produce digitally controlled gain (more information
about this system can be found on page 32). The automatic calibration was
performed using an algorithm with would check to see if the EMG system met some criteria, and if it failed to then it would adjust the potentiometer by 10 Ohms.

To determine what would be the best criteria, we performed an experiment where we asked several individuals of varied body types to test the system while changing the criteria. The original test used the signal of the user flexing with one logical condition (less than a reasonable strength signal for flexing), but this did not work as the signal fluctuates during flexion. The next test had the user relax while using a similar condition but with a lower threshold. This worked for some instances but also would sometimes over dampen the signal. Adding a second condition, a lower bound, fixed this issue and allowed all tested users to get a reliable scale for their EMG values.

**Version 3: Wrist Actuation**

The next system to be added was the wrist control, which is functionally similar to how the fingers operated. The major difference was that each motor in the wrist moved independently and had its own duty cycle. The wrist motors also used feedback to move, in conjunction with an IMU that is located in the hand itself. This provides positional information, which can be compared to a desired position and relate whether this needs the wrist motors to adjust to be closer to that ideal position. Data from the IMU is sampled continuously when the auto-balance functionality is activated. For the rotational motor, we used a proportional feedback so how much the position is offset correlates to how quickly the motors react. The flexion motor had difficulty lifting the weight of the hand unless it was fully powered, so it used a bang-bang controller: when the position was not in a certain range it would activate the motors until it was back within that range.

**Control Panel**

The final aspect of the control system was putting it together with a new form of input. We originally planned to use four pushbuttons but a quick survey of our control scheme revealed that many people thought it was too cryptic. Adding a fifth button cleared up most of the mystery with people, and allowed us to be more clear with the on screen instructions.

The new control system needed a means of organization, since there were several functionalities that needed to be included with five buttons to do so. This is implemented as having three different modes in the control panel, with matching on screen information. There is a gesture selection mode, wrist adjustment mode, and EMG calibration mode. The I/O of the control panel itself is treated as a state machine, using certain outputs during certain states (or modes), and recognizing that the inputs correlate to different functions depending on the state. Also the wrist adjustment mode contains two sub-states which allow the user to either adjust the rotation or flexion of the wrist.
Within the gesture mode there are two buttons to cycle through all the gestures, one to set the selected gesture, a mode navigation button, and a lock button. The lock functionality turns off the finger module, preventing them from moving. This is for people who want to hold something while moving the muscle the EMG sensor is attached to. The mode navigation button, if held down, will change the state to EMG calibration mode regardless of what the current mode is.

The wrist adjustment modes use two buttons to run the motors in either direction when pressed. There is also an auto-balance activation button, a button that switches between the wrist motors (which will also go to calibration mode when held), and a button that returns the user to gesture selection mode. The auto-balance function uses whatever the positional information is when it is set as the desired position for adjustment purposes.

The EMG calibration mode has an option to run the auto-calibration algorithm, but also the ability to set the specific sensor thresholds manually. This replaces the finger activation threshold with whatever the current EMG reading is, corresponding to whichever button is pressed. The last button returns the state to gesture selection mode.

EMG Data Acquisition System

The data acquisition system of EMG signal consists of three major component: the electrodes, the analog circuit, and the software algorithm running on PIC32. The three electrodes are attached to the triceps of an arm. The
positive, negative and ground electrodes are connected to the analog circuit through an audio jack. The circuit will differentiate, filter, and amplify the signal. Then smoothing algorithm will clean up noisy signal to produce more stable output. Calibration algorithm will change the gain of the analog circuit using a digital potentiometer.

The design of EMG signal conditioning circuit was done by following the practices from the past literature [11][12][13]: filtering out movement artifacts, rectifying signal, and using Nyquist filter. The design and analysis of EMG signal smoothing algorithm was done in Matlab. The raw EMG signal was obtained through NI 6009 DAQ and processed in Matlab with its data acquisition toolbox. The data retrieval process was designed to simulate what would be on microcontroller: sampling rate was set to 1 kHz which was feasible on microcontrollers, data smoothing function designed and written as it should be designed in C programming language, which would be used in our microcontroller. The criteria for evaluation include the implementation overhead incurred by having multiplicative arithmetic and RAM usage. Two algorithms were considered for smoothing algorithm: moving average filter and IIR butterworth filter. The parameter that can be changed for moving average was the size of buffer. The parameter for IIR filter include cut-off frequency and the order.

Fig. 17: Block Diagram of EMG Data Acquisition System
**Analog Circuit Design Specs**

An overview of the specs for the components in the analog circuit is shown below:

- **Operational amplifier:** GBW $\geq 5$ kHz, slew rate $\geq 0.02$ V/us, rail to rail output, low bias current $\approx 50$ pA
- **Instrumentation amplifier:** CMRR $\geq 100$dB,
- **High pass filters:** 25 Hz cutoff frequency, -60dB rolloff
- **Nyquist Low pass filter:** 500 Hz cutoff frequency, passive RC
- **Digital potentiometer:** resistance step size $\leq 50$ ohms, range: [100,1000]

**Op-Amp Specs**

Initially, the op-amp MCP6004 was used to test and build the circuit. Then it was determined to have higher supply rails for the op-amp for PIC24, which takes as high as 5V on analog pins, to take the full range of the analog pins. The specs for slew rate(SR) and gain bandwidth product(GBW) are shown below:

$$SR \geq 2\pi V_{pp} f_C = 2\pi (5V)(500Hz) = 2\pi (5V) \left( \frac{500}{s} \right) \frac{1s}{10^6us} = 0.01576 \frac{V}{us}$$

$$GBW \geq G f_C = (10)(500Hz) = 5kHz$$

The later change to PIC32, which has 3.3V analog pins, didn't alter the criterion much for selecting the op-amp.

$$SR \geq 2\pi V_{pp} f_C = 2\pi (3.3V)(500Hz) = 2\pi (3.3V) \left( \frac{500}{s} \right) \frac{1s}{10^6us} = 0.01036 \frac{V}{us}$$

**High Pass Filter and Movement Artifacts**

According to the literature, movement artifacts, the signal generated by the vibration and movement of wires, are below 20 Hz. This signal could falsely triggered the control system if not being filtered correctly. And it was tested with FFT that most of the movement artifacts are below 50 Hz.

All the filter designs were done using the formula provided in Introduction to Mechatronic Design by Carryer. Initially, a 4th order Chebyshev filter was built to test the effect of filtering movement artifacts. The result is shown in Fig. 19, where the yellow signal is the filtered signal, and the blue signal is the raw EMG signal. For the purpose of testing filtering effectiveness, the movement artifact was generated by tapping the electrode. This allows us to see the step response of the signal after the signal is filtered. The overshoot and ripple could make microcontroller harder to interpret the output of the circuit because it could be misinterpreted as a quick and strong pulse done voluntarily. So the filter was changed to Bessel for better retainment of the shape of the signal, which turned out to ease the ripple and overshoot effect.
Nyquist Filter and Sampling Rate

According the literatures, most of the important signals from EMG signals lie under 500 Hz, and suggested sampling is thus 1 kHz. Therefore the Nyquist filter is set to be 500 Hz. Based on experiments and FFT analysis, it was found the signal does lie within that range and to be more precise 80%+ lie within 250 Hz.

Smoothing Algorithm

A moving average algorithm was initially considered and tested in Matlab, their results are shown in Fig 20. As it can be seen in the graph, the higher the buffer size, the smoother the signal becomes; However, this means that the larger RAM size it requires the algorithm to store the past values, where buffer=128 means the size of buffer is 128, which is an array of 128 elements of type uint16_t, which is the smallest type that can be used for 10-bit ADC. And there’s delay associated with moving average algorithm, which is approximately

\[
\frac{\text{buffer size}}{\text{sampling rate}} \ [s].
\]

It was experimentally confirmed by measuring the beginning/end time of the analog signal and the beginning/end time of the digitally filtered signal. It was found that the delay for 256 size buffer is about 0.25s, and the delay for 516 size buffer is about 0.55s. The goal of the smoothing algorithm is to serve as protection against the sinusoidal nature of the EMG signal, so when it’s used for proportional control, the speed of motor won’t change suddenly as it can be seen in graph of buffer=128. The size of the buffer are set to be \(2^N\), to save computational time. When the sum is divided to obtain average, right-shifting can be used instead of numerical division. The delays are not desirable, but they can be mitigated by increasing the sampling rate as shown in the formula above. This is not better solution either because the faster sampling rate will increase the overhead of the entire system. Especially, we are using interrupt for our smoothing algorithm.
An IIR butterworth low pass filter was further investigated for smoothing EMG signal. With the help of `butter()` in Matlab, the coefficients for direct form I were calculated. The best smoothing effect was found to have the cut-off frequency of the IIR filter to be 1 Hz and the order is 2. Higher cut-off frequencies were tried but didn’t produce smoothed result, they looked like the buffer=128 graphs if the cut-off frequency is above 10 Hz. Higher orders weren’t investigated since it would increase the computational overhead on embedded system as more multiplicative operation would be applied. The result of applying the filter to the EMG signal is shown in Fig. 21. In contrast with moving average filter, which is a poor low pass filter with slow roll-off and poor stop-band attenuation, IIR filter has good performance in both space(RAM) and time(embedded computational overhead) complexity. To further optimize IIR filter, `fdatool` in Matlab was used to obtain coefficients of of direct form II in floats, which is the other available choice besides double. The floats coefficients have might be less precise but it has less decimal values. And other options, such as integer coefficients, produced by `fdatool` were not usable because the input gain produced was 0. Because of the symmetry of coefficients of poles, the IIR function is further optimized to only using two multiplication and three static double variables. The delay was analyzed using the same method as used for moving average filter, which is shown in Fig. 22. The delay was about 0.1s, which was later used with the control algorithm. This amount of delay turned out to be subtle. When we tested with motor control, it was very responsive to muscle flexes. We also measured the computational time of the IIR filter by turning a pin high when the algorithm starts, and turning the pin low when the algorithm stops. The measures show that it takes 50us for every function call, 50us / 1s = 0.005% of overall processing power. And we call it with a 1 kHz interrupt handler, so it uses 1000*0.005% = 5% of overall processing power. The code for 2nd order IIR butterworth in direct form I can be found in appendix B.
Fig. 20: Comparing IIR filters

Fig. 21: Step Response with IIR filter
Placement of the Electrodes

Fig. 22. Baseline Shift With Raised Arm

Fig. 23: Comparing EMG Signals of Triceps and Biceps
The placement of the electrodes was determined to be on triceps. During experiments, the baselines of the EMG signal were found to be increased when the entire arm is raised as shown in Fig. 17, where the arm began relaxed and being perpendicular to the ground, and then it raised by 45 degrees, and then 90 degrees. Each angle correspond to the average baseline 0.05, 0.19, 0.70, respectively. A comparison of EMG signals from biceps and triceps are shown in Fig 18. The signal for biceps were conditioned by the analog circuit while the triceps signal weren't because we had only one set of signal conditioning circuit at the time. However it can still be seen from the amplitudes of the raw signals, the triceps signals weren't influenced as much as biceps by raising the arm. Based on this result, we concluded that the EMG electrodes should be applied to the triceps in order for the patients to use it when his arm is raised. It will be much harder to determine the level of muscular contraction if the electrodes were applied to the biceps. Because we are using hysteresis as a means to control the motors of the hand, it is more appropriate to attach the electrodes where they are less influenced by the raised arm. However, this method with one EMG signal still the limitation that, the range of motion is limited to the front side of the person. When the arm is turned to the back of the person the effects are reversed, that the biceps don't generate much signal but the triceps do.

Digital Potentiometer

It was found experimentally that when gain resistor of instrumentation amplifier is of low resistance (<100 ohms), output disappears. Our hypothesis is that the negative terminals of the two impedance barrier op-amps of the three op-amp design of in-amp causes the difference amplifier to differentiate two signal of the same magnitudes. This can be resolved by increasing the gain resistor’s resistance. This means that calibration is needed to change the gain for in-amp, or no-output situation could happen. This requires changing resistance with potentiometer. We compared the mechanical and digital potentiometers, and the latter have more advantages than the former. First, digital potentiometer allows automatic calibration done by software, and the adjustment of the resistance is much more precise than mechanical ones. This can free users from doing calibration themselves, and still allow manual control through software if it is still desired. Second, digiPot is less prone to vibration and shock, where mechanical pot can have ±2% error [10]. And since we have haptic motor attached to the elbow piece, which is attached to the arm piece, the vibration will propagate to the PCB, and potentially affect a mechanical potentiometer and change the gain. Environmental events could results in changing resistance for mechanical pot, such as ±1% to ±10% from temperature cycling [10], and ±15% from humidity excursions [10]. The disadvantage for digiPot is that it has small current ratings, usually 5mA [10]; However, our application doesn’t require higher current ratings than that amount.

There are three ways to control digital potentiometer: up/down, I2C, SPI. We chose up/down interface because the way the preliminary design of calibration algorithm does not require changing resistance to certain value directly, only changing up and down is necessary.
The specs of digiPot include resistance range from $100 \, \Omega$ to $1k \, \Omega$, with a minimal step size of $50 \, \Omega$, and up/down interface. The digipot we obtained can change resistance every 2us.

**Automatic Calibration**

Different people have different strength signals, so the circuit won’t work at times when different people try. This could be caused by different skin impedance. And, according to research, athletic person’s muscle cells produce stronger signals. Therefore, digitally controlled gain is added to the circuit and to accomplish normalizing signal strength. So on startup, gain is adjusted based on EMG measurements. How the algorithm works is described below in pseudo code. In order to make it work we need to tune the parameter A and B. These two criteria for an appropriate gain is based on the user’s relaxed signal. Test subjects would first to be relaxed with their triceps and then we will do the auto calibration. Several individuals were asked to try using the EMG sensor while different calibration criteria was tested. In the end we found total process time is about 4 seconds.

**Pseudo code:**

1. Lower resistance to 10Ω
2. Take a sample every 1ms for 32 times
3. If the average of 32 samples greater than A and smaller than B, stop calibration
4. Else increase resistance by one step, go back to step 2
5. If increment resistance exceeds maximum number, stop calibration

**Circuit Design**

**Power Budget**

The amount of current consumed by each individual component in the system and the voltage rails required were analyzed to develop an estimate of average power consumption. This data would later be used to choose a battery with capacity to power the system for the desired length of time. This data would also dictate the requirements for the power supply design. The current draws listed below for each component are worst case estimates with the expected typical draw adjacent in parenthesis. Typical motor draw is assumed to be 0A because the motors are unpowered when holding a steady grip.
A reasonable estimate was established that the average person using a robotic prosthetic would need to adjust their hand position once every 5 minutes. These actions draw high currents but only over a few seconds at a time. It was estimated that 98% of the time the system will experience current draws closer to the typical calculated value of 205mA with max current draw occurring 2% of the time on average. Using this logic, it can be calculated that an overall average current draw for the complete system is around 300-400mA. Using a battery of greater than 2500mAh capacity should in turn supply the system for a minimum of 6.5 hours. This is acceptable for a common day of work. It is also important to note that prosthetics are typically not worn during periods of relaxation such as lounging or watching television so having a battery life that extends through all waking hours is not necessary. With hip mounted battery packs it would be quite easily to extend battery life without upsetting the weight balance and comfort of the arm.

**Battery/Power Supply**

When researching battery options it was important to our requirements to pick the lightest possible battery. Upon doing research it was found that lithium polymer batteries contain the highest charge density per weight. A Zippy compact LiPo battery pack was chosen at first with a 2700mAh capacity which would supply the system with power for a complete work day and minimize

<table>
<thead>
<tr>
<th>Voltage Rail</th>
<th>5v</th>
<th>-3v</th>
<th>3v</th>
<th>Battery(7.4v)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Finger Motors</td>
<td></td>
<td></td>
<td></td>
<td>5*0.8A(0)</td>
</tr>
<tr>
<td>Wrist/Arm Motors</td>
<td></td>
<td></td>
<td></td>
<td>2*1.5A(0)</td>
</tr>
<tr>
<td>H-Bridge ICs</td>
<td></td>
<td></td>
<td></td>
<td>4*3mA(0)</td>
</tr>
<tr>
<td>PIC MCU</td>
<td></td>
<td></td>
<td></td>
<td>200mA(25mA)</td>
</tr>
<tr>
<td>OLED Display</td>
<td></td>
<td></td>
<td></td>
<td>250mA(125mA)</td>
</tr>
<tr>
<td>Haptic Motor</td>
<td></td>
<td></td>
<td></td>
<td>80mA(0)</td>
</tr>
<tr>
<td>Servo</td>
<td>600mA(25mA)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EMG circuit</td>
<td></td>
<td>50mA(10mA)</td>
<td>50mA(10mA)</td>
<td></td>
</tr>
<tr>
<td>Gyro/Accel. Breakout</td>
<td></td>
<td></td>
<td></td>
<td>50mA(10mA)</td>
</tr>
<tr>
<td>Totals</td>
<td>600mA(25mA)</td>
<td>50mA(10mA)</td>
<td>630mA(170mA)</td>
<td>7A(0)</td>
</tr>
<tr>
<td>Total</td>
<td>8.28A(205mA typ.)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
overall weight. This battery, however had a profile that would not fit inside the forearm section so a Turnigy 2100mAh pack was used that had the correct dimensions to fit our design. This battery would still meet minimum specifications for battery life at a light 100g weight, shedding 45% from the weight of the previous MECH 1 battery.

The power supply required 3 regulated rails at 3v, -3v, and 5v respectively. 3v was chosen rather than 3.3v as this was within a safe range for the OLED display while also saving energy. The typical current draws on the 5v and 3v rails are of similar magnitudes. In order to maximize the efficiency of the design, switching regulation was used for both 5v and 3v rails. After some research the TPS54295 was chosen as the regulator IC. It features dual switching regulators than supply up to 2A continuous. This offered a very compact form factor for the power supply in a TSSOP16 package and its capacity offered flexibility for more components to be added to the design later. The schematic for the design can be seen below.

The design called for special components to be chosen. C3, C4, C8, and C9 are high quality ceramic capacitors with ESR of 2mohm or less to offer fast switching response and power delivery. The resistor dividers are composed of 0.1% tolerance parts to offer a precise output voltage. The 10nF on each SS pin dictates the slow startup time delay of the power supply used to protect from transient spikes. Time in ms of the startup is found with the given formula from the datasheet. \( T_{ms} = \frac{(C(nF) \times 0.765)}{8\mu A} \). This power supply offers 90% efficiency in the target current draw range of 200 to 600mA. Special care must be taken when routing this part of the board as the signal ground must connect to the power ground in one single location to control the impedance.

The negative power rail was developed using the LTC1983-3. This is a boost cap converter designed for 100mA maximum draw in a highly compact SOT23-6 package. This device requires only a single flyback capacitor to generate a -3v rail. A special low ESL capacitor was chosen to minimize the voltage ripple on this rail, as it would be used to power critical analog circuitry for the EMG control. The schematic for this circuit can be viewed below.
Motor Drivers

Each DC motor in the system would draw no more than 1.5A stall current so the H-Bridges were chosen to allow continuous current draws of at least 1.5A. The DRV8833 was ultimately chosen to drive all the required motors for a variety of reasons. It has a wide operating range from 2.7v to 10.8v which matches the range of the desired 6.6v LiPo pack. The package is a very compact TSSOP16 package containing 2 full H-Bridges. With a low RDS on of 360 milliohm on each switching FET the H-Bridges are very power efficient and can supply 2A peak currents despite their small profile. Care must be taken during soldering as the H-Bridges have an underside thermal pad to dissipate the necessary heat. The DRV8833 also has current sensing pins for each motor driver allowing easy implementation of motor current sensing. We aimed to use current sense as a control mechanism for the fingers so this feature was crucial. The equation found in the data sheet for the current limiting set by connecting a shunt resistor from this pin to ground is given as Ichop = 200mV/Rsense. 150 milliohm was chosen as the shunt resistor value so the current limit would be set at 1.33A. This would be high enough to never limit current to the DC motors in our design, but allow us to measure voltage drop across with the microcontroller to determine the current draw of the motor. The schematics for the complete motor driving circuit can be seen below.

Fig. 25: -3v Boost Cap Converter Supply
EMG Analog Input

In order to simplify design and condense the number of components used in the circuit, the calculations for the filter design are done to ensure only single standard resistor values are needed in as many locations as possible. Signals are input on a standard 3.5mm stereo jack as this provides the 3 connections needed and is cheap and easy to acquire with shielded cabling. Additional buffers were added not present in the prototype to ensure proper high impedance output to the microcontroller. The final schematic for the EMG acquisition circuit can be seen below.
Soft Power Switch

To avoid the use of a large traditional toggle power switch which must bear the full current load of the system, a circuit was devised to allow the use of an ordinary tactile switch of any size to activate the device by switching a power PFET. This offers more flexibility in user control options and also allows the microcontroller to latch the PFET and turn itself off as an added power saving feature. A circuit was developed from a preexisting Adafruit breakout but was modified to meet the higher current demands that our system would require. The circuit uses Schmitt trigger NAND gates to add hysteresis to the switch and avoid accidental power toggles. An appropriate PFET was chosen which could supply the maximum possible 8.28A draw. The circuit can be seen below. After some time developing and testing, the soft power switch was never fully reliable so to maintain a steady progression this feature was abandoned and replaced with a typical rocker power switch between the battery and the circuit.
**Haptic Motor Control**

The haptic motor is a small 3v vibration motor similar to those in a cell phone. The circuit used in our design is similar to that used by the previous MECH 1 team using a 280mA N-Channel MOSFET to drive the small motor. The base resistance, however, needed to be lowered because during experimentation it was found that there was not enough base current to drive the motor in our design. The final haptic drive circuit can be seen below.

---

![Fig. 28: Soft-on Power Switch Circuit](image)

**Fig. 28: Soft-on Power Switch Circuit**

**Fig. 29: Haptic Motor Drive**
PCB Development

Hand Connector Board and Wire Harness

The hand and wrist contain a total of 7 motors, one servo, and SPI interface for the positional sensor. Wiring all the connections inside the hand would be difficult and could result in messy tangles while the arm is in motion. For this reason, a small PCB was designed to fit inside the hand with terminal block connections for each motor and headers to connect the positional sensor and servo. This allows for a cable to cleanly and easily be unplugged from the back of the hand unit when performing maintenance. The secondary function of the PCB is to ensure that the positional sensor breakout is always in a proper orientation for accurate measurements.

![Hand Connector PCB]

The wiring harness contains a total of 24 connections that interface the hand and wrist to the console. In order to not inhibit the range of motion the cable is designed to be as flexible as possible. This meant using braided 26 gauge wire with soft silicone insulation. To keep all the wires neat and close together, the harness is wrapped in a flexible nylon mesh sleeve that is heat-shrunk at each end.

Console PCBs

In order to fit the electronic circuit into the dimensions required to fit the arm, a stack of two boards was designed. The bottom board houses the power supplies, analog EMG input and H-bridge motor drivers. It also features the wire harness connection that is routed to the hand. Extra attention to vertical height had to be taken when choosing connectors and capacitors especially for the
bottom PCB to ensure the two would fit together correctly. The top board holds the microcontroller and user I/O including an OLED display and 5 tactile switches. An ICSP header is placed near the edge of the board for easy access during programming. Both PCBs are designed with a width to exactly match the OLED. This allows for the overall smallest form factor and allows for easy alignment of the buttons with the GUI. Connection between the two board were done with standard 0.1” header pins placed in a way to best secure the boards together and not allow incorrect orientation.
PCB trace widths were analyzed to ensure that there would be no unnecessary or dangerous heatup for the amount of current drawn. Originally the plan was to use 2 oz copper for the bottom PCB so that finer traces widths could be employed for the same power. After some further analysis and routing optimization this was ultimately not needed and using 1 oz copper cut the price of the PCB by nearly 50%. To further minimize cost of production, boards were limited to two layers. This also opens up the opportunity to homemade PCB production for rapid prototyping of future designs.

**Choice of PCB Design House and Open Hardware Considerations**

EAGLE CAD was chosen to design all the schematics and boards for the project. This was done as it is used extensively by the open source community and they offer a freeware version available to any beginner level user. By posting schematics and board files online in EAGLE format along with Gerber files, we aim to make our project available to build and improve upon to the greatest number of people. In order to make the PCB reasonable to hand solder and build at home, passive components were chosen no smaller than 0603 and only integrated chips with exposed leads were used. In the case of the motor drivers and switching regulator IC, a thermal pad must be soldered on the underside of the chip. In order to allow the use of a standard soldering iron, vias are placed under the chip to flow solder from the back of the board. Many hobbyists do not have access to a hot air station so this feature of the PCB is critical to making it accessible.

ITEAD Studio was chosen to fabricate the circuit boards. They are a highly affordable Chinese manufacturer with decent quality offering 10 boards of 5x10cm dimension for 20 dollars plus shipping. An added bonus is that the company
provides DRC and CAM files specifically for use with EAGLE, making checks of design rules and proper Gerber file generation simple for the inexperienced.

Graphical User Interface

**SSD1322 Firmware**

The firmware was originally written in Motorola 6800 parallel interface where 12 pins were used to do the communication, 8 data lines and 4 control lines (RW, E, DC, reset) and CS is grounded. We found that we were using up all the pins and might encounter issues with limited amount of pins, so we rewrote the driver using 4-wire SPI. As it turned out the performance with serial interface was 10+ times faster than the parallel interface. This could be clearly seen when we tried to change the text. The SPI implementation would change instantly while the parallel would be lagging. The maximum clock rate found by experiment was 20 MHz.

**Graphics**

Our complete GUI is shown in appendix C. The graphics, drawing texts and geometry shapes, were implemented with uGUI library [14]. This library requires the implementation of a function, putPixel(), that puts a color at a coordinate (x,y). This library supports drawing framed/solid rectangles and circles, and printing strings. It defines more than 10 different sizes of fonts. When this library is included, all the fonts are stored as array of values in flash memory. We used 4 sizes of fonts, so a huge amount of flash memory is used by these fonts. This and the IMU library lead us to use PIC32MX340F256H.

The OLED screen was capable of displaying 16 colors in grayscale, we were only using four of the 16 colors. The four-color mode was implemented by using an uint8_t 64x64 size array, instead of uint8_t 128x64 size array. This array was determined to be necessary because how the internal RAM of SSD1322 controller works. Every RAM address stores 4 pixels’ grayscale data. When color is to be written to a RAM address, it writes four pixels at the same time. Therefore, putPixel() needs to know the other three pixels’ values when it writes only one pixel, or else the other three pixel will be overwritten. This has to do with how the API of SSD1322 is written: after the RAM address is written to the controller, two calls to writeData() are required to write to one RAM address, where the first call is to write to the first two pixels and the latter is to write to the last two pixel in the same RAM address. When using 16-color mode, our graphic library uses 8K RAM for the array. When using 4-color mode, it uses 4K RAM.

The bottom rounded frames with texts are the button indicators for the physical buttons that are situated below the OLED on the arm. At the center, texts indicating the gestures can be changed when the left and right buttons are pressed. The battery on the top right can display values from 0 to 100% (no bar to bars filled the rounded frame.)

We used the 0.2 version originally as it was the latest version, during the early spring quarter v0.3 was released. We tried to used v0.3 for its more advanced functionalities such as the window with buttons, but it didn’t work for
OLED screen, it might be only working for color screens, as shown in some of author's example videos. As a result of that, we implemented our own window and event-driven framework.

**Event-driven Graphic Framework**

An event-driven graphics framework was implemented using some C techniques: X-macro, state machines, and object-oriented C syntax. It was developed to be easily manageable and scalable as it’s possible more graphic objects will be added to the screen, and to be reusable in the future on other platforms. Graphic objects have their own member functions and states. The functions are draw(), erase(), invert(), changeValue(). The states are the graphic state(on, off, inverted), and variables that keep track of if it’s drawn or inverted previously. An update function can be called after all the objects change their states, so it won’t take too much processing power between processes as it was to be updated once every loop. And the graphic objects will only be updated when their states change, meaning the system will not take time to draw it repeatedly unless it’s necessary.

As the end of the project, buttons were the only objects that were supported because it was not necessary to manage other objects such as battery indicator and text indicators dynamically. And only draw and invert functions were implemented for buttons. It was determined that the only situation that any object needs to be erased is when window changes, most of objects on the screen need to be erased except battery indicator. And there were two choices to accommodate this situation, the first way is to implement erase() and call it, the second way is to erase the entire screen by banging data through SSD1322 API. To write or erase a pixel it requires 9 calls to writeData() and writeCommand() which set the D/C pin, write to SPI buffer to set RAM address and data to be written. To clear the entire screen, a function OLED_clearScreen() is implemented to clear the internal RAM of SSD1322 and the array that stores pixels information, it induces 16393 calls to writeData() and writeCommand(). By comparison, 
\[ \frac{16393}{9} = 1821 \] pixels can be erased if putPixel() is used to write 0x00 color to the screen while OLED_clearScreen() is called. 1821 pixels represent 256x64[pixel]/1821[pixel] = 8% of pixels of the screen. If more than 8% of the screen is being used, it will be more efficient to just erase the entire screen rather than erase the individual object. An experiment was set up to count the pixels drawn in window modes, the gesture mode uses 12444 pixels, the calibration mode uses 13194 mode. They are 75.9% and 80.5% out of the 16384 pixels, 111996/16393=683% and 118746/16393=724% more efficient, respectively. This
shows that it’s more efficient to erase the entire screen.

Buttons were categorized into two types: the ones that hold and the ones that don’t hold. The FSM diagram is shown in figure X. When the former type of button is pressed, it flashes by inverting its color and back to the regular color. And the latter type inverts its color and holds until it’s pressed another time. This type of button is used for indicating if auto balancing or locking is on. Each of them is managed by its own internal state machine which check if it holds and the previous and the current pressed button status. This implementation allows user to have good visual feedbacks and the latter button type saves space on the screen by using the same graphic space for two purposes.

Inertial Measurement Unit

One of our objectives with the wrist actuators is to achieve auto-balancing, or have the hand being parallel and perpendicular to a flat surface, so the hand can hold things flat and anti-shaking. This means we need to get a reference with respect to the earth or to the surrounding. We have considered an array of distance sensors. This method won’t work well if the hand is away from surrounding objects. And we also considered motor encoders, but it would only give a reference frame about the wrist joint, not the ground. So we had to use other kind of reference. We chose IMU because it allows our hand to move according to the initial reference frame where it started. The information we need would be just the angle of rotation and flexion since these are the two DOFs our actuators are capable of.

Our final choice came down to Invensense’s IMU because, first, we didn’t have enough analog pins for two DOFs IMU where it needs at least 6 pins for
accelerometer and gyroscope, and if more precision is required we need 9 pins, so we need a serial interface, secondly, most of the SMD MEMS IMU packages, such as LGA, QFN, and BGA packages, with SPI or I2C serial interface don’t have any lead. And all the available SOIC packages are really expensive, they range from $40 to $80. This leads to a search for cheap breakout board with no-leads SMD and small PCB surface area. We also defined the specs. The sensitivity should be 360 degrees/s, also the voltage supply should be either 3 V or 5 V. finally we need to have at least two DOFs IMU.

One of the difficulties of using IMU is that the gyroscope values drift. The solutions we explored are Kalman Filter, Complementary Filter, and Invensense’s DMP (Digital Motion Processor). We went with DMP because an analysis shows that DMP is capable of doing signal filtering well enough to mitigate drift [15]. Invensense provides the motion driver that only supports MSP430 and I2C. With some trials and errors, we were able to port the motion driver using SPI, some more details are provided in their appnote on their website [16] and our port is accessible through our github page. However, we are not using the Motion Processor Library algorithm (MPL), which has more than what we need. According to the appnote [16], 68K of flash and 16K of RAM (No Optimization, 64K and 10K with optimization) are needed to fully port the Motion Driver, including DMP and MPL. And we were able to find readily available algorithm to convert quaternion data to angles from [17]. At the end of our software development, we used the total of 5.7K RAM and 13.5K flash memory. We’ve optimized the system by not using the MPL, just the Motion Driver.

To validate if the DMP is correctly measuring the angles of roll and pitch directions, we set up Matlab to track the outputs through serial port. The results are shown below:

The data was recorded for 5 minutes. The top plot shows the yaw direction, the mid plot shows pitch, and the bottom shows roll direction. This experiment shows that the IMU drifts only in the yaw direction about 1 degrees per 5 minutes, which was the axis we were not using for our application. For roll and pitch directions, their angle measurements fluctuated for ± 0.1 degrees, which is not significant in
our application. However, aside from the good result, DMP does not show angles larger 70 degrees in one direction of the pitch direction. This does not inhibit our usage of IMU and the quaternion algorithm, because our hand was also limited mechanically that it will only flex for ± 80 degrees. This was a design to be more anthropomorphic. The root cause was not further investigated but the quaternion conversion algorithm might be the cause.

The performance for the function that retrieves the quaternion data from DMP and converts them into angles in degrees was measured with oscilloscope. It takes about 1ms to 1.5ms for each function call. The DMP refresh rate can be set in MECH_IMU.c. And we found the best performance is when DMP refreshes at 10 Hz. It was found experimentally that the function takes longer time to process above or below 10 Hz. And if the DMP update rate is too high, the internal FIFO will overflows and causes the outputs not being able to be retrieved.

Future Works/Improvements

There are several potential improvements that we suggest for future version of the project. Case design with more accessibility to inner components like the battery would be a major improvement. Making maintenance as easy as possible with the use of only one hand is key in making the arm more user friendly. We also hope the hand can be modular so that different types of amputee can choose from what they need. And upper arm can be developed to coupled with existing system. To make the whole system more robust, the hand should also be water-proof. Also USB connection to the microcontroller for easy firmware updates and PC software integration would be excellent for making the product more polished and professional while also increasing ease of use. PC integration would allow for Internet based features such as sending medical information to a doctor and allows for the use of more complex gestures. This goes in hand with having a real time operating system, in order to have a more precise and complete means of control of the hand. Adding a second EMG sensor input or more would also allow for more complex means of control with more electrode pads, and perhaps integrate Myo Armband to allow gesture selection without using buttons. Wireless EMG circuit can also be implemented to free the usage of EMG cables. ESD protection should also be added to the jack if cable is still used. The OLED screen did not perform optimally in sunlight in terms of readability, this is something to be improved upon. More of the user input could be done without the use of a second hand and the need for buttons would be diminished. Use of more powerful wrist rotation motors with a greater power to weight ratio would improve the weight-lifting performance of the arm overall. Lastly, the use of proper high-side current sense amplifiers on each motor driver would greatly improve the characteristic of the motor sense outputs and improve that effectiveness as a means of hand control.

Conclusion
Our final project is a good representation and proof of concept showing what can be done with equipment available at a hobbyist level at fairly reasonable cost, ~$500. It is an optimization of last year’s project that reduces and redistributes the weight for comfort, while also expanding the hand’s form and function to be more expressive. In this way we have succeeded in what we set out to achieve with this project. Future developers should note that while this design is functional, it is also fragile and has vast potential for improvement. By starting a website and making the project completely open-source, we aim to pave the way for future designers of 3d-printed prosthetics. In a short amount of time, we created a functioning electronic and mechanical platform for this open-source mechanical arm. We look forward to seeing where the public can take this project in the near future. All code and schematics will be posted at this URL: http://mech2.mozello.com/
Acknowledgements

We would like to acknowledge our advisors: Paul Naud and Patrick Mantey, for helping us direct our project into a successful form. We would also like to extend thanks to professors Gabriel Elkaim, Stephen Petersen, and Mircea Teodorescu for helping us with our various inquiries throughout this project. Additionally, thank you to the Smartbar team, who lent us their bodies to test our EMG auto-calibration functionality. Finally, we would like to thank the university and CITRIS for providing us with both space and resources to work on this project.

References


[7] “115:1 Metal Gearmotor 15.5Dx30L mm.”


[16] App Note 3- Motion Driver 6.1 Porting Guide

[17] https://github.com/rpicopter/ArduinoMotionSensorExample
Appendix A

**Ultimaker 2 3D Printer Settings on Cura**

Hand and Arm Parts:
Appendix B: DSP

optimized 2nd order IIR butterworth filter code in C

```c
#define a2 -1.991114259
#define a3  0.9911535978
#define G  0.000009825917

uint16_t IIR_2ndOrder(double x) {
    static double w1, w2, w3; // initialize to 0

    double y = w1 + w2 + w2 + w3;
    w3 = w2;
    w2 = w1;
    w1 = x - a2 * w2 - a3*w3;

    return (uint16_t) (G * y);
}
```
Appendix C: GUI

Gesture Mode

Rotation Mode

Flexion Mode (AUTO balance is on)

Calibration Mode (showing the instruction on the left, and EMG value on the right)