Assistive Arm Exoskeleton

A Major Qualifying Project
Submitted to the Faculty of
Worcester polytechnic Institute
In partial fulfillment of the requirements for the
Degree in Bachelor of Science
in
Electrical and Computer Engineering
And
Robotics Engineering
By

_____
Eric Carkin

_____
Parker Grant

And in
Mechanical Engineering
By

_____
Bryan Therrien

Date: 4/25/19
Project Advisors:

_____
Professor Marko Popovic, Advisor

_____
Professor Stephen Bitar, Co Advisor

_____
Professor Joseph Stabile, Co Advisor

Abstract

The Assistive Arm Exoskeleton aims to assist those in need of support when trying to perform daily tasks. The need for independence during the act of eating by those afflicted by Muscular Dystrophy was specifically targeted. In order to accomplish this, a motorized linkage system capable of supporting a person's arm proof of concept was developed. In implementation, the device would be attached to the user's battery powered wheelchair, and be available to them whenever needed. The system uses a collection of force sensing devices (collectively dubbed the "NUB") in order to detect user intent. With the information gathered by this sensor interface, the system can position the linkage, and subsequently the user's arm, anywhere within the work space, thus enabling the user to have independent control of their arm once again.

Acknowledgements

Without the help of certain individuals, the completion of this project would not have been possible. First we would like to thank WPI for funding and giving us the opportunity to complete this project. We would like to thank our primary advisor, Professor Marko Popovic for his guidance and allowing us use of his lab. We would like to thank the members of Popovic Lab for providing insight and support on our project. We would like to thank our co-advisors for the project Professor Stephen Bitar and Professor Joseph Stabile for providing help in their areas of expertise.

We would first like to thank Steve Forti, for allowing us to use his personal machine shop on the weekends and allowing us to use any parts we needed. His collection of bearings and stock metal made machining the parts necessary for the project much easier. We would also like to thank Kyle Richards for coming with us to the shop to help us use the equipment there. When we were not using Steve Forti's shop, we were in Washburn. Ian Anderson and James Loiselle were very helpful in using the CNC machines available on campus. Finally, special thanks to Brett Yoder for his consulting on both mechanical and electrical hurdles we faced during this project, as well as helping to maintain our access to 3d printing technology.

Executive Summary

With the aim of aiding those affected by muscular weakness that prevents the performance of activities of daily life, a support device was developed that would re-enable the user to move their arm and interact with the world around them. The end result was a 5 degree of freedom linkage that could be attached to a wheelchair and supports the user at the forearm. This linkage allows for motion about the X, Y, and Z axes, and allows for motion associated with the bending of the elbow and rotating of the arm, but not for the rotation of the forearm independently. The system as a whole is an active one, relying on powered motors and sensor readings for its control. This enables the user to have assisted control of their arm in all directions, instead of simply receiving a passive gravity compensation. In order to be usable by the target audience, the control of the device is performed via a highly sensitive force sensing device dubbed the “Nub”, due to its likeness to the mouse control nub found on the keyboard of some laptops. This force sensor was developed specifically to accomodate the needs of the system.

The mechanism can be broken down into 2 parts, a planar arm and tower. The planar arm consists of 3 joints and is primarily responsible for all translational movements. The first link is attached to a platform, which in implementation would then be attached to a wheelchair. This link is supported by a circular bearing to prevent sag and binding from the large load it needs to move. The next link is attached via a rotational joint to the first, and ends in a rotational joint that supports the tower. The first and second both have the ability to rotate 360 degrees, allowing for smooth and continuous movement by the user. The joint at the base of the tower is primary responsible for the rotation of the arm, but in some positions bends the elbow in instead. The tower gives the mechanism the height it needs to be mounted at waist level and still allow for the needed motion of the arm in the Z direction. Two joints forming a 5 bar are at the top of this tower, and are responsible for movement in the Z direction along with rotation of the arm/ bending of the elbow in certain conditions.

Electrically the system can be seen as a network of nodes with distributed tasks. Each node has a different function, from positional control of joints to reading sensor input. All of these nodes communicated with a master, which is responsible for relaying positional information to a computer and then broadcasting new set points to different nodes on the network. All node communications happen via a I2C bus, while computer communications happen via a Uart channel. All nodes consist of an independent microcontroller. Each node has one of 3 elements attached to it; an encoder for positional feedback, potentiometer for positional feedback, or HX711 ADCs for user input recognition. Nodes responsible for position control also interface with motor drivers to actuate the links they are responsible for. Through the use of regulators, the entire system can run off of a 12V battery, typical of the type found on powered wheelchairs.

In order to control the mechanism, each node responsible for the motion of a link runs a tuned PID loop, that uses the attached sensor element for positional feedback. The setpoints that these nodes are using to find an error value however are determined by the computer attached to the master. The entire control structure operates in a loop. The computer will broadcast a series of set points to the master, which will then pass those setpoints on to the

correct node. Each node will adjust its set point, and then respond with the position the link is currently in. The master then requests data from the user input node, which detects user intent via the Nub. The master then sends all this data back to the computer. Upon receiving, the computer performs forward kinematics to determine the position of the mechanism in world space. It then uses the user input forces to generate a positional vector that determines the new goal of the linkage end-effector. With this new goal, inverse kinematics are performed to determine the new set point of each individual joint. The setpoints are sent to the master, and the process repeats.

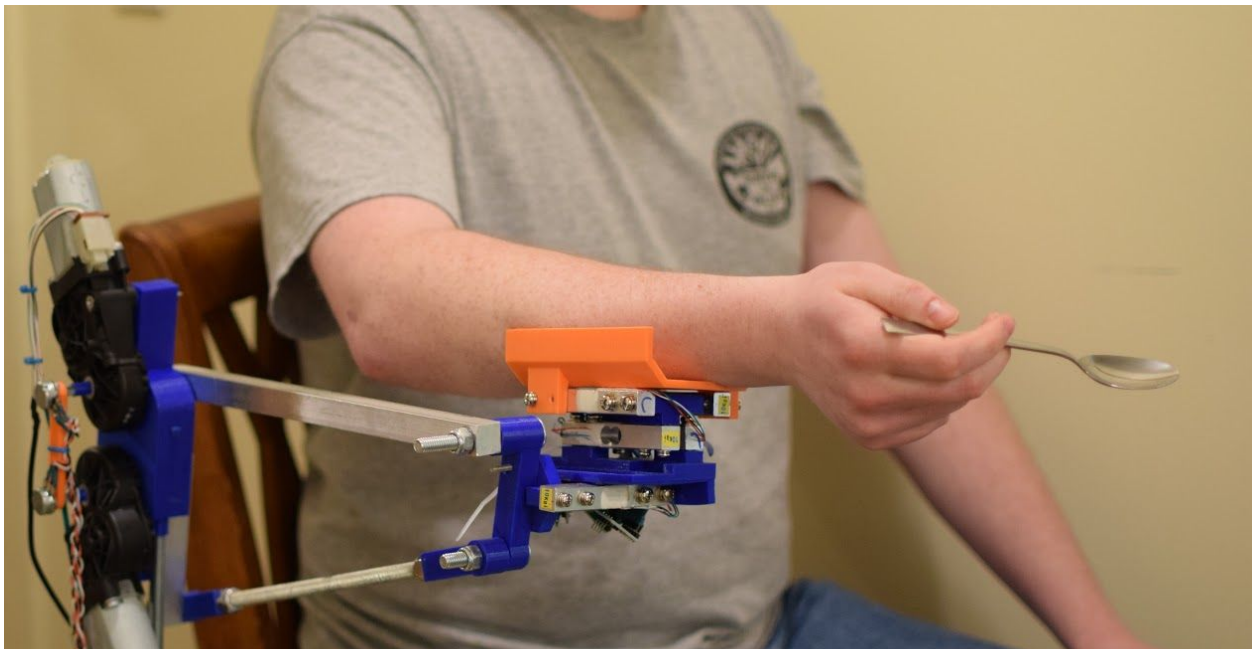


Figure 1: Resulting System in Use

Abstract 1

Acknowledgements 2

Executive Summary 3

Chapter 1: Background 8
1.1 The Need 8
1.2 The Target Group 8
1.3 State of the Art 8
1.4 The Motion of the Arm 10
1.5 Sensors 11
1.5.1 EMG Sensing 11
1.5.2 Force Sensing 11
1.5.3 Comparing Force Sensing and EMG 12

Chapter 2: Proposed Design 13
2.1 Project Goals 13
2.2 Project Objectives 13
2.3 Project Management and Tasks 14
2.4 Design Decisions 14
2.4.1 Mechanical Design 14
2.4.2 Electrical Architecture 17
2.4.3 Controls Method 19

Chapter 3: Implementation 22
3.1 Mechanical System 22
3.1.1 Stage 1: Unpowered Prototype 22
3.1.2 Stage 2: Working Prototype 23
3.1.3 Stage 3: Final Version 24
3.2 Electrical System 26
3.2.1 User Input Evaluation 26
3.2.2 Sensor Development 28
3.2.3 Initial Design evaluation 29
3.2.4 System Overhaul 30
3.2.5 Power Distribution 31
3.2.6 The Middle Joint 31
3.2.7 PCB Design 32
3.2.8 Troubleshooting 33
3.3 Control System 35
3.3.1 Actuator Control 35

3.3.2 Forward Kinematics 35
3.3.3 Inverse Kinematics 36
3.3.4 Interpreting user input 36

Chapter 4: Conclusions and Future Work 38
4.1 End Product 38
4.2 Next Steps 38
4.2.1 Mechanical Changes 38
4.2.2 Electrical Changes 38
4.2.3 Control Changes 39

Appendix 41
Electrical Schematics and PCB Layouts 41
B. Coordinate Frame Reference 47
C. Encoder_Node Code 48
D. Potentiometer_Node Code 50
E. Middle_Joint DAC 52
F. Middle_Joint Node 53
G. Nub Node 54
H. Control Script 56

References 66

Table of Figures

Figure 1: Resulting System in Use 4
Figure 2: Anatomical Body Position 10
Figure 3: Linkage Motion Diagram 15
Figure 4: Linkage Torque Estimation
16
Figure 5: Basic Proposed Design 17
Figure 6: Initial System Block Diagram 18
Figure 7: Pololu Motor 19
Figure 8: Labeled Joint Variables used for Kinematics 20
Figure 9: Proof of Concept Model 22
Figure 10: Flex Shaft Couplings 23
Figure 11: Initial Planar Assembly 24
Figure 12: Revised Link 2 24
Figure 13: Final System Implementation 25
Figure 14: Integrated Difference Between Bicep and Tricep Readings During Curls 26
Figure 15: Bicep Reading and Potentiometer Reading During Curls 27
Figure 16: User Input Prototype 28
Figure 17: Final User Input Sensor 36
Figure 18: Fully Assembled Dual Node Board 33
Figure 19: Final Electrical Architecture 34
Figure 20: Interrupt to DAC PCB Layout 41
Figure 21: Node and MSP432 Breakout PCB Layouts 42
Figure 22: Nub PCB Layout 43
Figure 23: Dual Node Schematic 44
Figure 24: Interrupt to DAC Schematic 45
Figure 25: Nub Schematic 46
Figure 26: Reference Coordinate Frame 47

Chapter 1: Background

1.1 The Need

As people lose muscle function in their limbs, they begin to lose the ability to perform everyday tasks and hobbies. What was once an easy task, such as eating or cleaning, becomes an extreme challenge that often requires help from loved ones or service people. This can be problematic, relying on others puts their life on hold, while also taking time away from those who help them. If people were able to be more independent for a longer period of time, they and those around them could experience a higher quality of life. 13.2 million adults that live indepently receive an average of 31.4 hours of assistance a week with activities of daily living(LaPlante, Harrington and Kang, 2002). Furthermore, 21.3% of people interviewed living under Medicare reported unmet needs for assistance with daily activities (Craig et al., 2015). With unmet need associated with increased death/ hospitalization rates, giving people the ability to be independent is essential to increasing the quality of life for society.

1.2 The Target Group

There are numerous groups of people who suffer from debilitation, ranging from the elderly to neuromuscular afflictions. Three major diseases/ events that can cause loss of function in the muscles are Muscular Dystrophy, ALS, and strokes. Muscular Dystrophy is a disease that causes severe muscle weakness and degeneration (MDA, 2018). It affects many major muscle groups, including those responsible for the control of limbs. Duchenne Muscular Dystrophy, the most severe form of muscular dystrophy, affects 1/7250 males in the US, and causes 90% of those afflicted to be wheelchair bound by age 24 (CDC, 2018). After having a stroke, many people have difficulty controlling their muscles and suffer from partial paralysis (National Stroke Association, 2015). There are around 800,000 reported cases of stroke victims in the United States each year (The Internet Stroke Center, n.d.). ALS is a neurological disorder that affects the nerves involved with muscle control (NINDS. 2018). Nerves are damaged, causing nervous system signals to incorrectly reach muscles or even not reach them at all. This can cause muscle twitching and degeneration. Fourteen to fifteen thousand Americans have this disorder and have difficulty controlling their limbs.

1.3 State of the Art

As of now, there are multiple solutions on the market that attempt to address the issue of independent living for people with limited muscle strength (Popovic, 2019). Many of these systems, while effective, stop short of providing an ideal solution and creates issue of their own.

Liftware Level is a product that tackles the issue of trying to eat with limited mobility by modifying the utensil the patient uses (Verily, n.d.). Their spoons use auto leveling technology so that no matter how the user moves their body, food will stay on the spoon. This solution however does not address the problem if the user does not have enough strength to actually move their arm to their mouth.

Neater is a company that focuses on helping the disabled in many different areas of life (DLF, 2018). They have numerous products designed to help the disabled, especially when it comes to eating. The Neater Arm Support is a linkage system that compensates for the weight of the user's arm. This is a passive system, with no feedback control- it allows what little strength the user has to be concentrated at moving an object- not their appendages. While meant for eating, the product claims that it may be able to help facilitated numerous daily actions other than eating. This system is somewhat awkward- the support structure attaches to the back of a wheelchair, but its movement increases the space the person takes up. In other words, the operating space of the wheelchair is increased in order to account for the moving linkages. Additionally, movement allowed by the system is awkward. Due to their little strength, the user has minimal control over their movements. Moving something to their mouth for example is very similar to "throwing" their hand at themselves.

The Neater Eater on the other hand is a spoon system mounted to the users plate (DLF, n.d.). This system can be configured to work a couple different ways, but achieves the same function. A spoon/ fork that can obtain food, and then bring it to mouth level. The system can be controlled by a hand lever system, or by electronic joystick. While using this seems somewhat more elegant than the Neater Arm Support, it is limited to helping the user eat. Additionally, it requires special set up- limiting where the system can be used easily.

The iArm by Exact Dynamics is similar to the Neater Eater in that it is a controllable robotic arm (Exact Dynamics, n.d.). The difference is this arm is mounted to you chair, and is much more flexible with what it can be used for. While this arm is far more versatile, its drawback is that you use a joystick to control it. Use of a joystick requires the user to have a free hand to dedicate to using it and have the ability to manipulate the controls with that hand.

The JAECO WREX arm exoskeleton is a passive linkage system that proved a vertical force upwards on the forearm (Jaeco Orthopedic, 2018). This force is meant to be similar to the downward acting force gravity has on the arm, countering it and allowing the user to move more freely. The system contains 5 degrees of freedom with 3 links rotating about the Z axis and 2 about the Y axis. The system has elastics providing force onto the two links that are rotating about the Z axis (assuming a standard coordinate system.) The links rotating about the Y axis do not have any assistive forces acting on them. The drawback of this system is that a balance needs to be achieved with the force of the elastics. As the arm is raised the force the elastics provide gets lower. If the initial force provided by the elastics is too high then the resting place for the user would be above the table which causes discomfort to the user.

The Stable Slide Self-Feeding device is a simple ramp that the user places their forearm on(Performance Health, 2018). The normal force provided by the ramp acts to reduce the Y direction force that the arm experiences due to gravity. It also provides a minimum point that the forearm can be lowered, which reduces the amount the user need to raise their arm to reach their mouth.

1.4 The Motion of the Arm

Standard convention for naming movements of the body requires the body to be in the anatomical position; standing upright, feet together, arms down and palms forward. From the position the body is divided into three planes. The transverse plane differentiates the body into top and bottom sections. The sagittal plane differentiates the body into left and right sections. The coronal plane separates the body into front and back sections. These terms can be used to more accurately describe the movements of the arm.

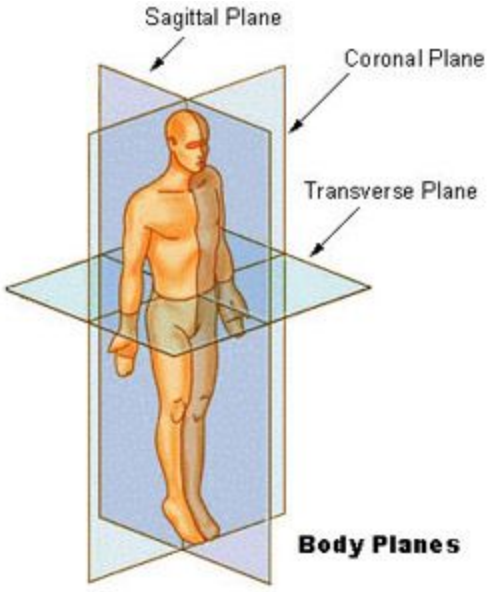


Figure 2: Anatomical Body Position(national Cancer Institute, n.d.)

The upper arm has rotation across all three planes. Movement parallel to the sagittal plane is called flexion for forward motion and extension for backwards motion.(Jones, 2018a) Movement parallel to the coronal plane is called abduction for upward motion and adduction for downward motion. Movement parallel to the Transverse plane is called internal roll when the elbow rolls away from the body and external roll when the elbow rolls towards the body. and external rotation describe the movement towards and away from the midline in the plane which separates the top and bottom sides of the body, sometimes referred to as the transverse plane. The shoulder can also translate along axes normal to the transverse and sagittal planes.

The forearm has rotation across only 2 planes. Flexion and extension of the forearm describes motion parallel to the sagittal plane similarly to the upper arm(Jones, 2018b). Movement parallel to the transverse plane is however described as supination for when the

palms are moving towards a forward facing position and pronation when the palms are rotating to face backwards.(Jones, 2017; Shahid, Goffin & Chaves, 2018)

The motion of the components of the arm described previously can now be used to describe the movement of the arm during eating. A study analysing typical people eating using a fork, spoon, and drinking from a glass found the following degrees of motion to be used; 5 degrees to 45 degrees shoulder flexion, 5 degrees to 35 degrees shoulder abduction, 5 degrees to 25 degrees shoulder internal rotation, 70 degrees to 130 degrees elbow flexion, from 40 degrees forearm pronation to 60 degrees forearm supination, from 10 degrees wrist flexion to 25 degrees wrist extension, and from 20 degrees wrist ulnar deviation to 5 degrees wrist radial deviation(Safee-Rad, Shwedyk, Quanbury and Cooper, 1990).

1.5 Sensors

It is imperative that the controls of this system are accurate, as it will be operating in close proximity with people and dealing with tasks that have little room for error. For this reason, the system will utilize sensor fusion- the use of multiple sensors to obtain usable feedback. Of these sensors, the intent is at least to use encoders on all motors, as the information provided by them is extremely versatile and can be used for determining the exact position of the arm in task space, how fast it is moving, and other factors without needing an immense amount of interpretation. However, encoders can only provide information about the current state of the arm, methods are needed to determine where the user wants to go.

1.5.1 EMG Sensing

One method of control would be through the use of EMG. EMG, or electromyography, are the nervous system signals used to control muscles (Raez, Hussain and Mohd-Yasin, 2006). These signals are used frequently for the diagnosis of neurological diseases, and dictate the motion of skeletal muscles. Through appropriate signal manipulation, such as filtering and Fourier analysis, distinct signals can be detected that indicate different actions for muscle groups. Through the use of electrodes, the EMG signals of a user could be used to control prosthetics, or in this case the motion of an arm support.

In *Implementation of EMG- and Force-Based Control Interfaces in Active Elbow Supports for Men With Duchenne Muscular Dystrophy: A Feasibility Study*, a single threshold control scheme was used, where the current measured signal was compared to a steady state signal collected prior to tests, and then scaled by the average maximum signal. This was done for both the bicep and tricep, and their difference was used to control an active elbow support.

1.5.2 Force Sensing

The other primary method of control considered is force monitoring. The idea is that the user can exert miniscule forces on the world around them. If these forces can be detected, they can be used to determine the direction and orientation the user is trying to move their arm to. *Implementation of EMG- and Force-Based Control Interfaces in Active Elbow Supports for Men*

With Duchenne Muscular Dystrophy: A Feasibility Study demonstrated that force monitoring is a viable method of active support control for Muscular Dystrophy patients. However, it is stressed that the voluntary forces produced by the user must be differentiated from other external forces for effective control. This can be complicated, but was completed by force estimations for their 1 DOF set up. For force measurement, the study utilized a single DOF force sensor; (LSB200 – 5lb, FUTEK Advanced Sensor Technology Inc., USA., a variant of load cell.

There are a few different technologies used to measure and detect forces (national Physical Laboratory, 2010). The most popular of these are load cells. Load cells are configurations of elastic materials and strain gauges. They offer reliable and robust methods of force detection. These sensors are readily available, and come in numerous designs depending on how a force is to be measured. When cell experiences a force, it deforms. This deformation causes strain gauges, typically made of some kind of electrical foil, to stretch or compress. These compression and strain forces cause a change in the resistance of the foil, which can be detected and used as a force indicator. These metal foils can be used in a similar way as a pressure sensor in the form of a small mountable pad.

A typical load cell has a gauge factor around 2, which is an indicator of how sensitive it is to strain (Al-Mutlaq, n.d.). When a force is applied to a load cell, it can be expected that a result on the order of millistain can be expected. With these factors, it can be expected that an applied force will result in a resistance change on the order of milliohms. Due to this small resistance change, amplification is needed in order to get a reliable voltage output. Differential ADC's, such as the HX711 can be used in combination with wheatstone bridge configurations, to achieve this amplification and reliable signal interpretation. A combination of this ADC and a 5kg load cell would only cost around 10 dollars.

1.5.3 Comparing Force Sensing and EMG

Both primary control structures of force monitoring or EMG are viable, according to *Implementation of EMG- and Force-Based Control Interfaces in Active Elbow Supports for Men With Duchenne Muscular Dystrophy: A Feasibility Study***.** In this study, the use of EMG and force monitoring methods were investigated to control active elbow supports for men with DMD. The results were quite positive for both force control and EMG. Even with the fact that DMD degrades EMG signals considerably over time, it was found that usable signals were still able to be extracted for control uses. On average, force control methods were faster and more accurate, but were more tiring than EMG. Additionally, force control requires more estimations to determine what's a user applied force and what is an external force.

Load cells are far cheaper than EMG sensors. It costs about 10 dollars for a load cell and an amplifier on Amazon. Basic EMG sensors on the other hand cost around 40 dollars, and don't include any high level processing or analysis (Adafruit Industries, n.d.). Additionally, sensors would be needed for each major muscle group- requiring many electrodes to be attached to the user. In order to implement the load cell approach, a single cell would be needed for each degree of freedom. Finally, as discussed earlier, EMG sensors require extensive analysis.

Chapter 2: Proposed Design

This section discusses the main goal and objectives of the project. It goes into detail about the general design for the exoskeleton along with the hardware and software used to implement it.

2.1 Project Goals

The Purpose of this project is to create a mechanism that assists people with the task of eating. To accomplish this the system must be able to be attached to the users arm, so that they feed themselves, instead of the machine feeding them. It must be able to move the users arm as well as a small payload so that it can carry the food up to the mouth. The sensor system must be easy and intuitive to use. The target users do not have much arm strength, but have fine motor control, therefore the sensor system should be able to utilize their motor control without requiring much force to be applied.

2.2 Project Objectives

- No dangerous elements exposed to user/nearby people
- The system is able to be securely mounted to stationary object (i.e. desk/chair)
- The system is intended for users sitting in an upright position
- System reacts dynamically to user input with a maximum response time of 150ms
- The system will be able to support and lift a weight of 10 lbs (the weight of the average human arm is 9 lbs(Plagenhoef) plus weight of 16 oz glass of water is 1 lb(perlman)) attached to point of contact with the arm
- The maximum speed of the end effector is at least 0.18 m/s (roughly 4 second from plate to mouth)
- Jerk limit of the system is 100 m/s^3 (Breteler, Meulenbroek & Gielen, 2002)
- Acceleration limit is 10m/s^2 (Breteler, 2002)
- System can reach three positions/ orientations, named mouth position, plate position, and home*
    - Plate: Coordinate:[-5”,9”,2”] center, with a 5” radius(X & Y) and 4” height(Z) cylinder, Rotation:[-10,0,30] degrees
    - Mouth Position: Coordinate:[-5”, -3”, 14”] center, sphere 1.5” radius, Rotation: [0,20,90] degrees
    - Home Position: Coordinate: [0,0,0], Rotation: [0,0,0]
    - Please see appendix for pictures of position/ how coordinate frame is to be interpreted.
- Evaluate the viability of EMG and force sensors for user feedback

2.3 Project Management and Tasks

To complete this project, the work was divided up into three parts based on the specialty of the members. The three sections were 3.1 Mechanical System, 3.2 Electrical System, and 3.3 Control System.

The deadlines for the various parts of the project were broad, only consisting of major goals to be completed at term ends. For the mechanical system the deadlines were completing the initial design of the system by the end of A term, a physical system by the end of B term, and a fully functional system by the end of C term, leaving D term for tuning specific issues and allowing the other two parts of the project to work with a fully functional mechanical system. For the electrical system the deadlines were structured by subsystems. The sensor input was to be developed by the end of B term, motor and positional control system was to be developed by the end of C term, leaving network development/ troubleshooting for D term. The forward and inverse kinematics were to be worked out by the End of B term, while keeping in mind the mathematics were subject to change if the mechanical system underwent any substantial edits. PID tuning would occur as soon as the electronics were ready, and the total control loop needed to be finished by mid D term.

2.4 Design Decisions

2.4.1 Mechanical Design

For the mechanical system design it was determined that there were two main methods to consider. The mechanism could be either attached to the user or to a structure near the user. Each method has its advantages and disadvantages. First, the method of attaching the mechanism to the user will be discussed.

To obtain required motion for eating a mechanism attached to the user would have to actuate a system that rigidly connected to both the users forearm, upper arm, and shoulder/torso. The forearm has to be able to move relative to the upper arm and the upper arm has to be able to move relative to the torso. For full range of motion the shoulder connection would need three degrees of freedom, being able to rotate about all three axes. The forearm connection would need two degrees of freedom, elbow flexion, and forearm pronation. The mechanism would need to be somewhat lightweight, as the user be supporting the weight of the mechanism with their body. It would need to be also be thin so that it could be attached along the arm without interfering with movement of arm against other surfaces such as tables or the torso.

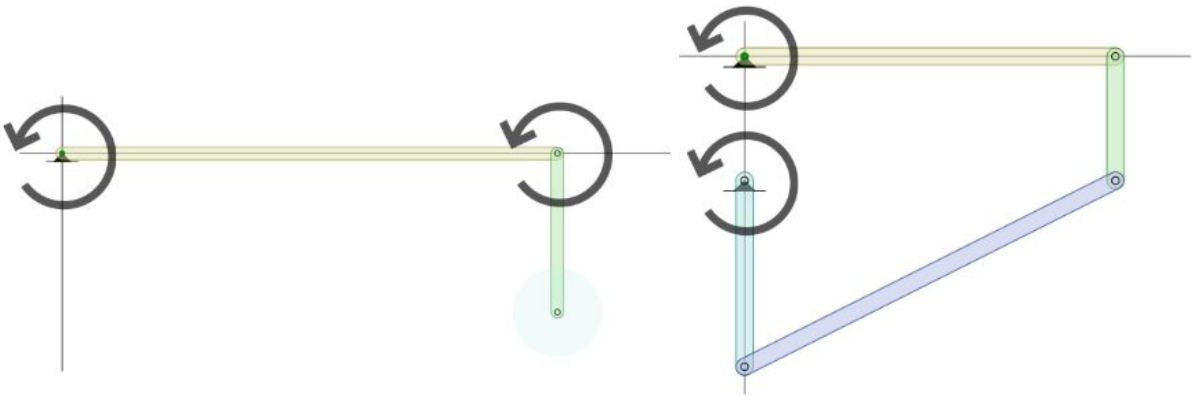
The primary difficulty observed was the difficulty of imitating the shoulder joint. Requiring three degrees of freedom around one point while also requiring it to be lightweight and small in size would be rather difficult to accomplish. Also having the mechanism close to the body requires small lever arms for the joints to be moved about. Small lever arms require bigger

forces making the necessary motors larger. This causes a tradeoff between bulk and weight that would be hard to overcome for a user friendly mechanism.

The second method would be a linkage that attaches to the arm and is mounted to another structure, such as a wheelchair or a table. Since this mechanism is not attached to the user the weight is not nearly as much of a concern, as the structure would be supporting it, not the user. The size of the mechanism still does matter, it needs to be small enough that it does not interfere with the table, user, or any other surroundings during use. Another concern with this type mechanism is that it is less discrete than the user mounted system especially with increased size.

The actual mechanisms that could be created for this method vary as well. It was determined there were multiple options to consider. The mechanism was split up into two main sub mechanisms, one (dubbed the tower) to move the arm in the z direction, as well as rotate about the y axis. The other sub mechanism is the planar mechanism to move the tower in the x-y plane and rotate it about the z axis. Two concepts were developed for each of the sub mechanisms.

The tower is a tall structural component that connects the lower portion of the mechanism to the linkage on the top. There are two configurations for the linkage as shown in figure 3. The user's forearm is attached to the green section and the actuated joints are marked with a circle. Configuration 1, shown by Figure 3a is a 3 link linkage. Configuration 2, shown by figure 3b is a 5 bar linkage. The benefit of configuration 2 is that both of the actuated joints are located on the tower, making it easier to transmit the torque required for motion. Having two attachment points to the tower also makes the system more rigid. The main benefit of configuration 1 is that since it has only one link coming from the tower, it is a more discrete system. Configuration 2 was used for the linkage on the tower.



a b

Figure 3: Linkage Motion Diagram

For the planar mechanism, their are also two configurations that were considered. The first one is similar to configuration 1 of the tower shown in figure 3a. The difference for the planar mechanism would be that both of the links would have the same length. This would create a circle in the x-y plane with radius double the length of the links. The mechanism would be able to move the tower to any position within that circle, and be able to move in any direction. Configuration 2 is a system similar to a crane at an arcade. The tower would be mounted on a threaded rod. The threaded rod would be turned, sliding the tower axially along the rod depending on which way the rod was turned. There would then be two perpendicular threaded rods on either end of the first one. They would be connected the same way giving the tower movement in both the x and y plane. Both of these configurations require a separate component to adress rotation about the z axis. Configuration 1 was the configuration pursued. It was used because it was a smaller system than configuration 2, allowing it to take up less space and be mounted closer to the user. It also requires less motors, needing 3 instead of 4. The only advantage of configuration 2 is that the controls aspect would be simpler.

A model of the mechanism (figure 5) was created in Creo to determine how much torque would be required for each actuated joint. Figure 4 shows the torque required for a movement of the five joints. The figure shows three torques near 0 and two significant torques. The two significant torques are at the joints in the tower. Adding elastic elements to the links could adjust the torque requirement to around 10 Nm for the tower joints. Based off of these torque requirements motors can be found, the actual selection will be covered in the next section of the report.

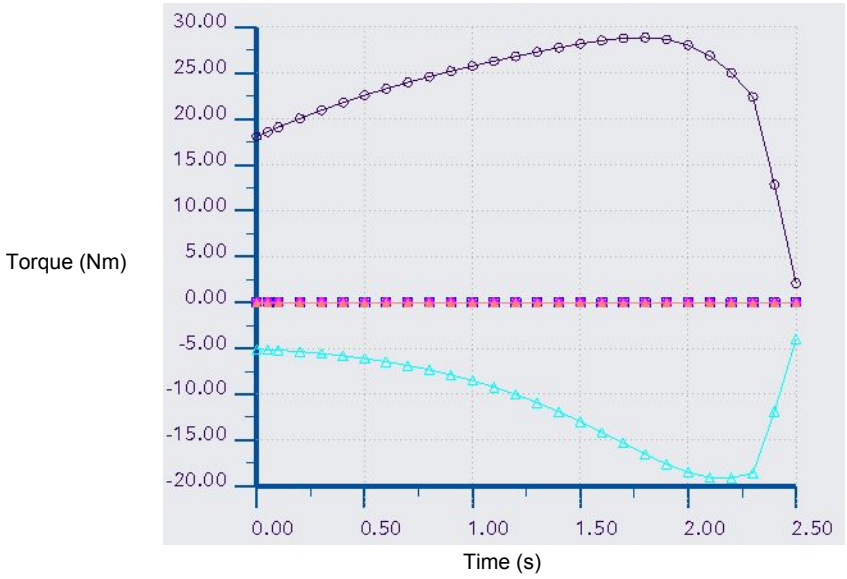


Figure 4: Linkage Torque Estimation

To transmit the torque from the motors to the joints, options were explored other than direct drive for the tower motors. Having the motors lower on the tower or in the base of the mechanism would create a more discrete system. The torque transmission scheme is based on

the use of flexible drive shaft. Flexible drive shaft would allow the motors to be in the base of the mechanism. For the joints in the planar linkage, direct drive would be acceptable.

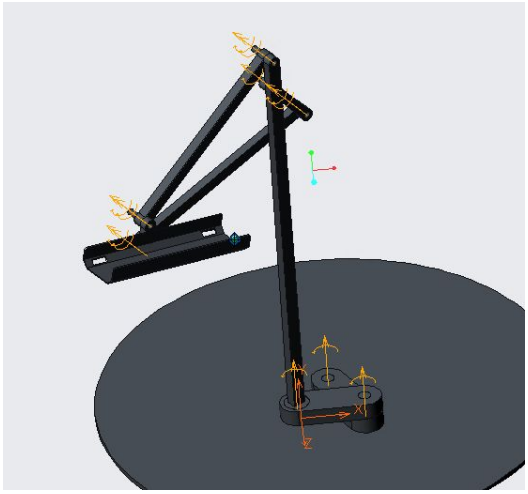


Figure 5: Basic Proposed Design

2.4.2 Electrical Architecture

The core piece of the initial electrical system was a real-time operating system on a MSP432 controlling all major functions. This was used over other microcontrollers due to familiarity from classes and the availability of the board. This controller would be responsible for PWM outputs to maintain motor positions, reading of the sensor input, and communicating with an off board computer to determine new set points for each motor position. The MSP432 would communicate with a seperate computer via a Uart connection, due to its speed and wide scale availability.

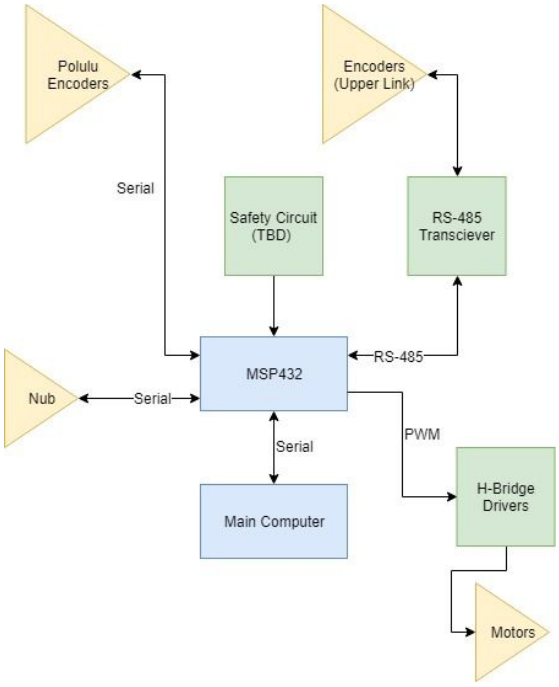


Figure 6: Initial System Block Diagram

Two possible sensory inputs for controller the mechanism were proposed at the beginning of this project; EMG and force feedback. The first term would be spent evaluating these two options and then selecting the most viable one. Due to the experimentation needed to determine the user sensor input method, a solid design for reading this input was not proposed at the beginning of the project. However, due to the high configurability of the MSP432, the expectation was that any needed method could be integrated into the existing design. As for feedback elements, rotary encoders were to be used for all joints due to their ease of use, high accuracy, and possible factory integration with selected motors. This of course would require the use of interrupts on the MSP432 to keep track of the position of each motor.

Pololu motors are used for the planar arm joints. These motors both satisfied the torque requirements of the joints and could be ordered with an integrated encoder. These motors have a 64 CPR encoder on them. However, when taking into account the gearbox on the output of the motor, the encoder actually creates a resolution of 6400 counts per revolution. The upper joints however required a high torque. For this reason, bosch seat motors from Andy Mark were used. The hall sensor on these motors however would only allow steps of 1 degree, which seemed too coarse for the desired application. For that reason, CUI Inc rotary encoders were intended to be used with the seat motors due to their fast shipment time, high resolution, and serial interface.

Figure 7: Pololu Motor

With the end product likely being mounted to a motorized wheelchair, the electrical system will need to be able to work off of a 12V battery. Due to the high capacity of the type of battery used in the target user's typical wheelchair type, power consumption was not considered a high priority. A 5v regulator would be needed to power the MSP432 launchpad off of this battery. All motor selections could operate in a 12V range, so the only additionally component would be motor drivers that operated in the 12V range and could receive a 3.3V PWM signal from the MSP432. According to their respective datasheets, the pololu motors would not draw more than 5A during stall, while the seat motors would stay under 12A. A suitable driver was found on Amazon by the company Drok. These drivers had 2 channels on one board each capable of handling 6A. The channels could be tied in parallel for the seat motors. While this does not leave much room for error, larger drivers costed considerably more. That, couple with the fact that these motors should not reach stall current under normal conditions, it was decided that this driver would suffice for this project.

2.4.3 Controls Method

As discussed earlier, the initial control design relied on a MSP432 running a real time operating system. This operating system would need tasks to cover PWM signal and PID loops, user input reading, joint sensor reading, and communication with the off board computer for new set points. Of these tasks, the communication task would have the highest priority make sure full messages are transferred between the two devices. The PWM signal task would then have the second highest priority task to make sure the PID loops can be accurately tuned, joint sensor reading would then be third to ensure the joint positions are accurate, and user input

would then fall in last. In order to meet the project goals, this RTOS would need to be set up such that user input is read at least every 150ms. The offboard computer was responsible for the mathematically intensive tasks to reduce the load on the microcontroller. This primarily involved the kinematics and the user input processing.

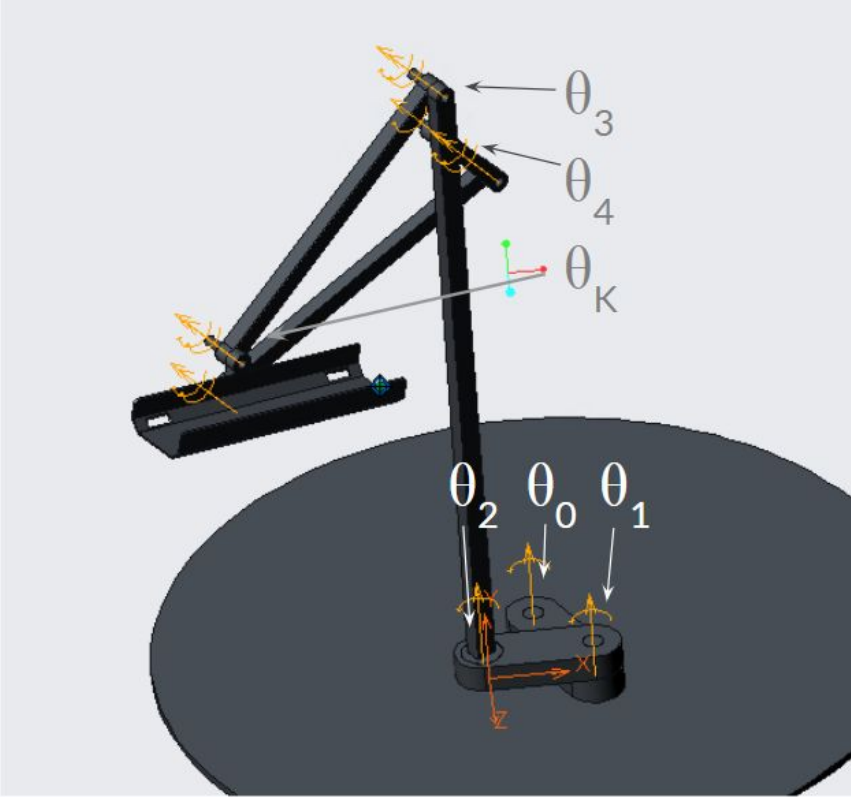


Figure 8: Labeled Joint Variables used for Kinematics

The forward kinematics function is responsible for determining the world space position that results from a given set of joint angles. The robot has direct control over five joint angles (θ_0 θ_1 θ_2 θ_3 θ_4) as shown in figure 8. This allows the robot to directly manipulate five degrees of freedom in the worldspace. The controllable axes are x, y, z, θ_Y(pitch) and θ_Z(yaw), while θ_X(roll) is fixed to 0. The mechanism can be split into to two sections that make the kinematics easier to understand and calculate. These sections are the 5 bar linkage on the tower assembly, and the planar base assembly. The 5 bar linkage on the tower is controlled using θ_3 and θ_4, and is the part of the mechanism responsible for controlling motion in the z and θ_Y. The base plane section is controlled using θ_0, θ_1 and θ_2, and is the part of the mechanism responsible for controlling motion in the x, y, and θ_Z.

Originally, the forward kinematics were implemented using DH parameters and homogeneous transformation matrices. The kinematics for the planar subsection were trivial to solve for, but the 5 bar linkage introduced much more complexity to the kinematics calculations for the other section. This is because the 5 bar linkage is a closed kinematic loop, which means that angles in this loop are constrained by relations between joint lengths and other angles in

this loop. To aid the calculations the model was simplified so it could be represented as a standard serial manipulator with an open kinematic chain, so that this model could be used to generate out DH parameters. To accomplish this, the 5 bar was simplified down to a planar 2-link arm with joint angles of θ_3 and θ_K. While the robot has the ability to directly control the value of θ_3 on the robot, it cannot do the same for the joint that corresponds to θ_K. θ_K is instead dependant on θ_3 and θ_4, and used as a symbol to help with calculations.

For inverse kinematics, the first attempt used a jacobian matrix. The Jacobian matrix describes the relationship between instantaneous changes in the joint-space and world space variables. The inverse of the jacobian matrix is actually what is needed to determine the inverse kinematics for the robot. On this robot, there are 5 controllable variables in both the joint-space and world space, so the jacobian matrix for the robot was square.

The jacobian matrix is a tool that can estimate the change in world space variables for a given change in joint space variables. Similarly, the Inverse jacobian can be used to determine the change in joint space variables for a change in world space variables. The jacobian matrix is different for different poses of the robot, and so the current joint angles of the robot must be known in order to calculate it. This is not a problem for this implementation since there is always positional data available for all of the joints. The Jacobian is also a local approximation, so it gives efficient and accurate results for small movements, but gets progressively less accurate for larger movements. To correct for this when trying to do large motions, the inverse jacobian can be used repeatedly until the results converge towards the correct answer. To make this repeated process converge, the resulting joint angles from the previous attempt are used as the “initial position” to calculate the new inverse jacobian, and the inverse kinematics are done once again. The joint angles that result from this can then be put through the forward kinematics function and compared to the target world space values. If these two values differ too greatly, the process can be repeated as many times as is necessary to get a solution that is close enough.

The nub sensor provides 5 force values that are used to determine user intent and move the robot accordingly. These force values require some processing in order to be used intuitively for user input. There are 2 pairs of parallel load cells and one single load cell. Each parallel load cell pair is capable of reading a force and a torque. Summing the values of each load cell in a pair will give the force along an axis, and the difference will yield the torque along another axis. This calculated force is along an axis parallel to the axes in which the load cells in the pair measure their forces. The torque vector calculated from a load cell pair is along the axis perpendicular to the plane formed by the individual load cell force axes. The load cell that is not part of a pair directly measures the force along a third axis. Using this technique the X, Y, and Z components of the force the user applies can be found, as well as the pitch and yaw components of the torques they apply.

These vectors can be combined into single force and torque vectors. The assumption is made that these vectors indicate the direction the user would like to move. These vectors are relative to the reference frame of the nub, so if the sensor is mounted on the end effector of the robot, the robot’s forward kinematics are needed to transform the force and torque vectors into the global reference frame. Once the the user’s desired direction of travel in the global reference frame is known, the robot can be controlled to to move accordingly.

Chapter 3: Implementation

3.1 Mechanical System

Creation of the mechanical system is split into three stages; unpowered prototype, working prototype, and the final version. The first stage, creating the unpowered prototype, would be a proof of concept that the mechanism will be able to achieve all the positions necessary, as well as getting a feel for the specific issues that need to be addressed when creating the first working model. The second stage was creating a working prototype that could actuate all the required joints. The final stage is the final working model. 3D printing and threaded rods were utilized to make the stages. From that mechanism, it can be determined which components need to be manufactured by other means, primarily machining.

3.1.1 Stage 1: Unpowered Prototype

The completed stage 1 mechanism is shown below in figure 9. The model is relatively easy to make, using only threaded rod, 3D printed material, and bolts. There were no difficulties in creating this mechanism other than learning the tolerances required for 3D printed interfacing. With this prototype built, the motions were tested, showing that the mechanism would be able to achieve the full range of motion required. From this stage the mechanism can be updated and converted to the stage 2 mechanism.

Figure 9: Proof of Concept Model

3.1.2 Stage 2: Working Prototype

The first task for creating the working prototype was to develop adapters so that the flexible drive shaft could connect to the motor and also to the joint. The three parts that were printed for this application are shown in figure 10. The parts (from left to right) are the pin, the connecter, and the shaft. The end of the flex drive shaft is put into the pin and clamped in using screws. The pin is connected to the links that are to be actuated. The connecter clamps the two drive shafts together. With the two flex shafts connected to each other, the tubing wont spin and therefore requires no other attachment to the tower other than directly to the pin. The shaft part is simply an adapter so that the motor can interface with the other end of the flex shaft. The power test for the flex drive shaft failed. The shaft internals broke before any of the plastic 3D printed components.

Figure 10: Flex Shaft Couplings

Three methods to fix the flex drive shaft failure were discussed by the team. For the flex shaft to work, the torque being transmitted needs to be reduced. This can be accomplished by gearing down the torque before it is sent through the flex shaft and then gearing it up again on the tower. This method seemed to counteract the benefits of using the flex shaft in the first place. It would require bulking up the system and putting a lot of parts at the top of the tower, two things that the flex shaft intended to not do. The next option is to develop a system with wires or belt and mount the motors near the bottom of the tower. The final option would be to simply direct drive the motors at the top of the tower. The links at the top of the tower ended up being directly driven to reduce complexity and get back on schedule. The reason a system was not developed with wire or belt is because there were no readily implementable alternatives that were less bulky than direct driving the motors, and having the motors at the bottom of the tower would take away from the area that could be used to attach electronics.

The direct drive was easy to set up. When actuating the linkage was tested, the plastic deformed in the pin. Because of this the pin was machined out of aluminum. After machining the pin out of the stronger material, the test was successful. The motors were able to actuate the links and support the weight of an arm. After some testing the plastic in the link itself began to deform.The link was machined out of metal for stage 3.

Next the planar linkage is developed. The first iteration is shown in figure 11. The joint that connects the tower to the planar linkage has a donut shape. The reason for this shape is because the use of gears would have introduced other issues and there is no room to attach the motor on the bottom of the link. The motor connects to a cube with a matching interface to the D shaped shaft. The cube has a slot interface on the link. This method of attaching the shaft to the

motor is the same for all three attachments on the planar sub mechanism. The reason for this attachment is it allows the cube to be replaced if it deforms at all. The cube is much smaller and faster part to reproduce than any other part on the mechanism. The other two joints are fairly simple. The motor is attached to a cover for the link and the shaft goes through the cover onto the link that the cover is being rotated relative to.

Figure 11: Initial Planar Assembly

The planar linkage was completed and actuated. The links rotate easily before the tower was attached. Once the tower mechanism was placed on the end of the planar linkage the linkage stopped working. The weight of the tower caused too much friction on the joints, the only joint that was still able to turn was the one directly under the tower.

It was next realized that the mechanism needed a slip ring located directly under the tower joint. The wires for the second motor would rotate around the tower during operation and get tangled. This was not a solution that could be fixed without a slip ring or using wireless communication. A slip ring was purchased and link 2 was modified to accommodate the new component. The cube method of attaching the motor to the link was no longer an option with the slip ring, as the part needed to pass through the center of the ring and didn't have enough space for the cube. The interface with the shaft had to be placed directly into the part. This is a concern because if the interface deforms the entire link would have to be reprinted, and this is the largest print on the assembly. Figure 12 shows the new link 2 that accommodates the slip ring. With this change, stage 2 is complete and stage 3 can be started.

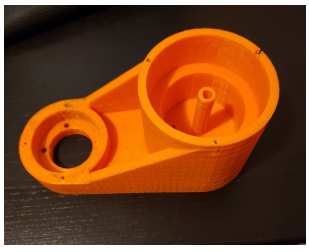


Figure 12: Revised Link 2

3.1.3 Stage 3: Final Version

With the working plastic prototype it was determined which parts needed to be converted to metal. It was decided that all of the planar sub mechanism was to be machined, as well as

the two links in the tower linkage that the motors actuate. The two links from the tower linkage were reproduced with metal bars with no changes to the design. The design of the planar linkage changed slightly. With the metal parts, ball bearings were included at each of the joints to reduce friction that was slowing down the plastic version. A lazy-susan ball bearing was also added at the end of link 1. This lazy-susan removes all deflection from that link. The motor attachment points were converted from the cube interface to set screw locking. Using set screws instead makes it easier to take apart and eliminates the possibility of deformation breaking the interface. The final version of the mechanism is shown in figure 13.

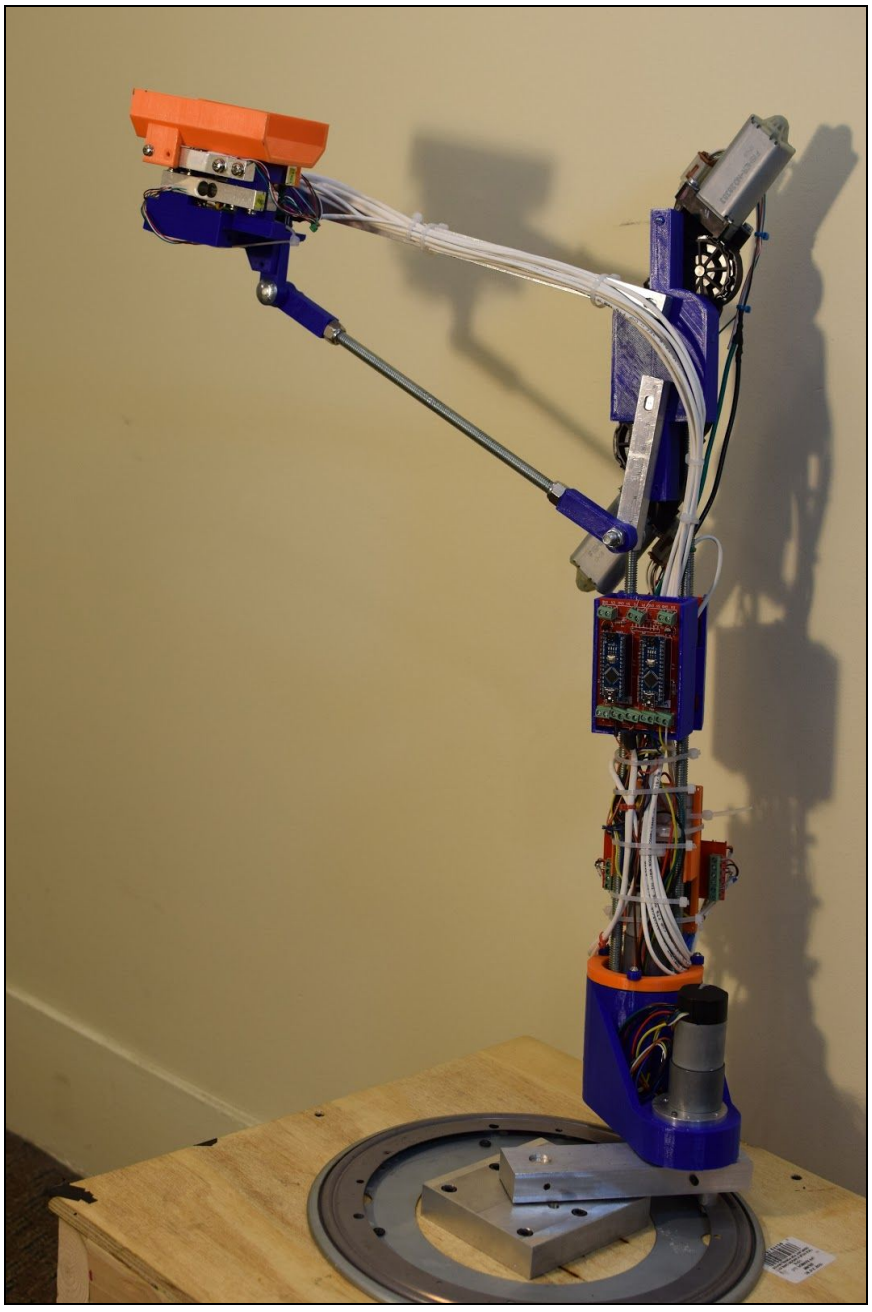


Figure 13: Final System Implementation

3.2 Electrical System

3.2.1 User Input Evaluation

The first step in the development of the electrical system was the evaluation of the EMG and force control approaches. The research started with EMG, and two emg sensors were found from Myoware that could be used to read signals from the surface of the skin. In order to use this sensor input, it needed to be confirmed that they could be used to create accurate position control. To evaluate this, one sensor was placed on a test user's bicep, while the other was placed on the tricep. The user then curled their arm as if they were using a dumbell. The sensor output was recorded and then integrated over time. The best results can be seen below.

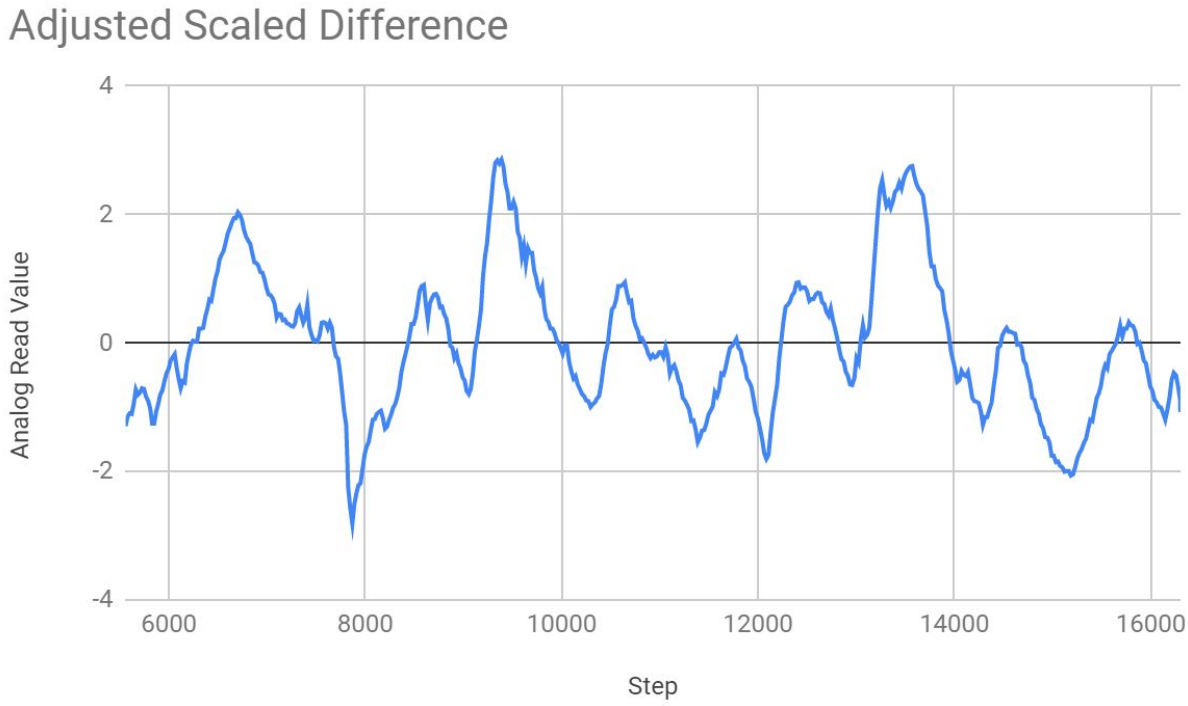


Figure 14: Integrated Difference Between Bicep and Tricep Readings During Curls

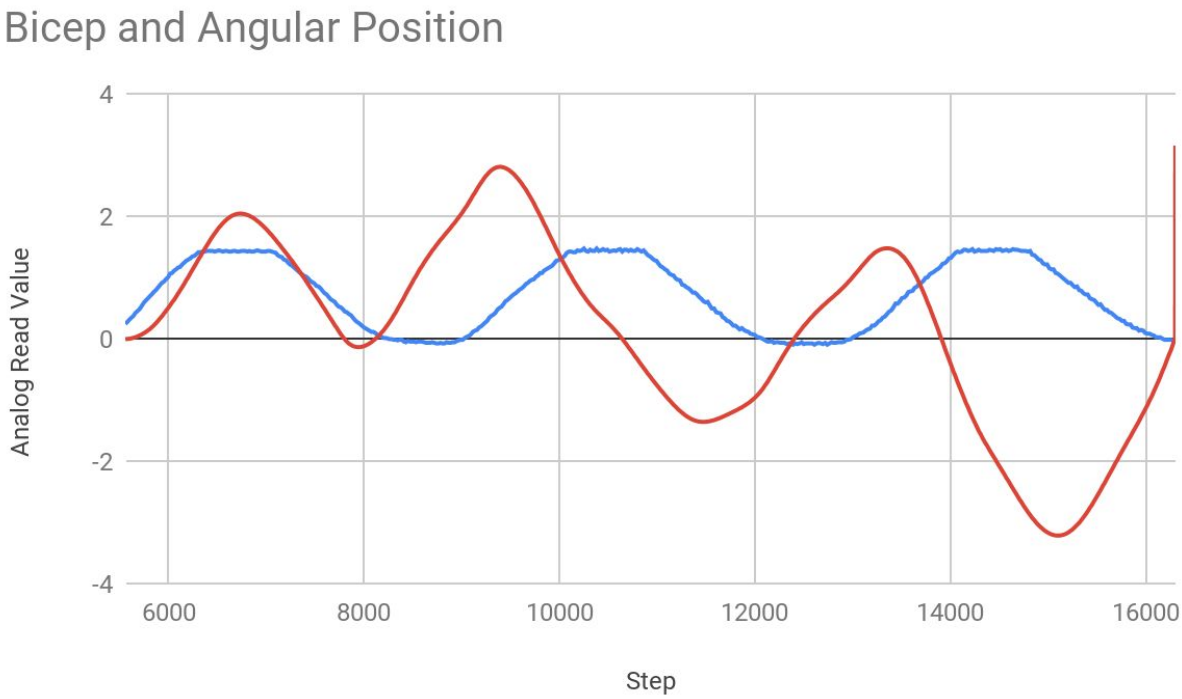


Figure 15: Bicep Reading (red) and Potentiometer Reading (Blue) During Curls

The most difficult aspect of EMG is the actual placement of the sensors. Special sticky pads are attached to the users skin on the muscle that is to be monitored. Placement is absolutely imperative to get correct or else the signal would be completely unusable. Out of 20 attempts, only 5 produced usable results, likely due to poor placement. Furthermore, these pads were uncomfortable to wear for an extended period of time, and actually could cause the skin to bruise if left on for too long. For these reasons alone it was decided that EMG was not usable. The use of the sensor was simply too cumbersome and difficult to set up. Even with that, the sensor was extremely prone to noise and drift. One test revealed that touching a cell phone caused measurements to spike, and all attempts showed drift that occurs very rapidly.

Next, the use of force feedback for user input was to be evaluated. The easiest known way to measure forces was through the use of a load cell. Familiarity of using HX711 ADC amplifies with load cells from previous classes pushed the guided the purchase of a load cell and amplifier to see how sensitive they were. It was found that the cells would detect forces perpendicular to the cell in one plane. Pushing down on a cell would create a positive reading, while pulling up would create a negative one. There was very little hysteresis in the sensor, if any, and plenty of software libraries existed to use as reference to interface with the HX711. With a 10kg cell, it was found that a change in sensor readings could be caused with the mass of a penny. From this

evaluation, it was apparent that load cells were a suitable option, and moved to implement a user input system with them.

3.2.2 Sensor Development

In order to control the 5 dof linkage, forces in the X, Y, and Z directions needed to be monitored, along with moments about the Z and Y axes. The problem then presents itself: how to distinguish a force from a moment. The best apparent approach was to used coupled load cells about the Z and Y axis. By placing each cell on opposite sides of the force application, The two cells can be compared. If they both have opposite magnitudes, then a force is being applied. If the moments are the same then a moment is being applied.

Two moments and 3 forces would therefore require a total of 5 load cells and HX711 amplifiers. Initially, a tall tower was designed the hold the cells and demonstrate how the mechanism would work (figure 16). This frame was 3d printed and a n arduino was used to measure force readings from the load cells. With very little effort, the system was able to measure how the user was interacting with the frame, whether it was a force or moment.

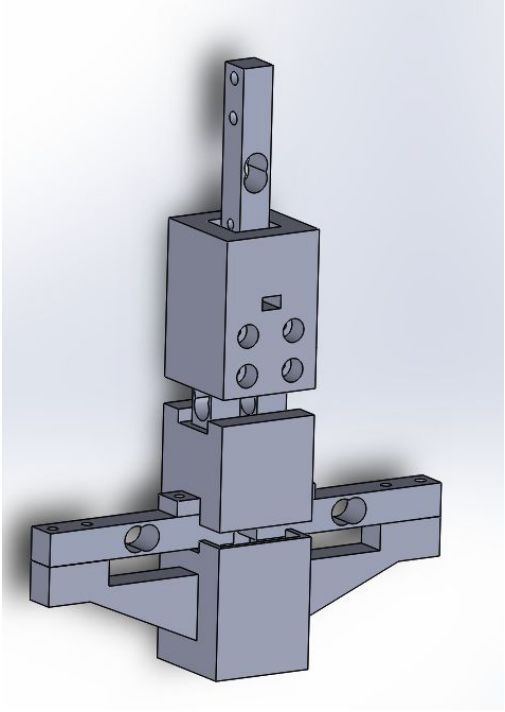


Figure 16: User Input Prototype

After confirming that this method of detecting user intent was feasible, the next step was designing a more compact structure for the sensor, along with developing a library for interfacing the MSP432 with the HX711s. The HX711s work similar to a shift register. Data from a read is clocked out with 24 pulses. An additional 1 to 3 pulses will then set the gain for the next read. For the largest possible gain, a total of 25 pulses are used when getting data from the HX711. Pulses are applied to the SCK input of the HX711, followed by a read on the DT pin.

These reads are then shifted and added, creating a 24 bit value representing the reading of the HX711. To read the load cells on the MSP432, 10 pins were configured for IO, 5 for DT and 5 for SCK. A read function sequentially pulsed the SCK pins and read the SCK pins. Each read was considered a binary bit, shifted to its correct weight, and added together. The function then returned 5 values representing the read of each cell. Further functions were added to the library to maintain a rolling average of readings for each cell to eliminate noise, along with an initial offset read to zero the load cells.

The sensor array was compacted down into a gauntlet, which was a suitable size and form for being mounted to the linkage. The gauntlet design was 3d printed, assembled, and tested with the newly developed library for the MSP432. This new system was then dubbed the “Nub”, due to its likeness in operation to the small nub found in the center of some laptop keyboards that a user can move the mouse with. This new device measured forces and moments as expected, however the plastic body of the gauntlet introduced a large amount of hysteresis. This is partly because PLA does not share the same modulus of elasticity as the aluminum load cell, as well as the fact that the screws would sometimes catch on the plastic pieces, causing the gauntlet itself to apply forces the cells once the user has stopped doing so. This hysteresis needs to be accounted for when reading the Nub, however it is manageable and valid readings are still able to be taken from it.

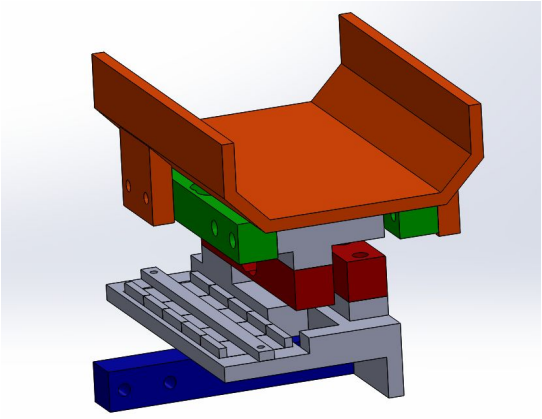


Figure 17: Final User Input Sensor

3.2.3 Initial Design evaluation

With the user input settled, it was time to develop electrical system dedicated to maintaining PWM/ motor control. The first motors purchased were the bosch seat motors, as they were needed to test top link actuation. For that reasons, the CUI encoders were also ordered. The MSP432 was configured to have 5 pwm outputs. A simple proportional controller was then written to control the two seat motors. The trouble began when trying to interface with the CUI Inc encoders, which had their own serial communication protocol.

These encoders used a RS485 communication structure, which uses a baud rate of 2Mbps. Using the Uart libraries for the MSP432, serial communications with a putty terminal at rates up to 115200 baud were able to be consistently achieved. At 2Mbps however, communications were quickly dropped or corrupted. Due to deadlines and increasing suspicion that the MSP432 could not reach such a high rate without some serious reconfiguration, the decision was made to change sensing methods. The top links have a rotation of less than 270 degrees, which meant that a potentiometer could easily be mounted to the top joints to track position. This also meant that the top links would not need to be homed on startup, and would decrease cpu load by preventing the need for more encoder interrupts. Two spare 5K ohm potentiometers were mounted to the links, and 2 ADC channels were configured to read the potentiometers on the MSP432.

From here, the development of the electronics was re-focused to achieve milestones that would help with the testing of the mechanical system. An easier way to apply torques to the upper links was wanted for stress testing. For this reason, the development of Uart communications was started so that the PWM signals could be varied more easily during testing. With TI provided Uart examples, it didn't take long to communicate with the MSP432 through a putty terminal. However, once the Uart channel was continuously used via a python script, it was noticed that after a few seconds communications would drop. After isolating pieces of the MSP432 code, it was found that the addition of the PWM drivers caused the communication problem. It was theorized that the hardware interrupts generated by the clocks for the PWM driver were occuring during Uart transfers, causing communications to become out of sync. With Uart and PWM both being an integral part of the design, redesigning the electrical architecture of the system was necessary.

3.2.4 System Overhaul

With the large number of tasks that needed to occur in the system, along with the need for fast response times, it was decided that the new system revision would need to allow for the ability to perform multiple tasks simultaneously. In order to achieve this, the electrical system was expanded into a network of nodes. A single node would take care of PID loop and PWM generation, along with positional feedback sensor reading. Each joint requiring a node would require at minimal 5 nodes and one master to be present on the network. With such a large number of peripherals, an I2C communication protocol was used to communicate between nodes, reducing the number of physical connections needed across the linkage as well as quick and easy data transfers.

A possible issue with this system approach is an increase in cost. Each node would require its own microcontroller, along with any other supplemental electronics to allow it to interact with the different sensors, motors, and communication channels. However, due to the light load required of each node small, low cost controllers could be used, such as an Arduino Nano. A nano is a small board that can be purchased for only 3 dollars online from certain vendors. Additionally, Arduino comes with a well documented library and example rich environment that will allow for fast paced development. This was key due to the unexpected setback of needing a complete overhaul of the electrical system. Since the MSP432 was readily

available, it was decided that it would be used as the communications master. Its responsibility would be to read the Nub, communicate with the pc via Uart, and send set points to the different nodes on the network via I2C.

Two separate types of nodes developed from this change, the difference being in the sensor used for feedback. One type of node used an adc to read the position of potentiometers, while the other used two interrupts to keep track of encoder positions. Other than this difference, the nodes are identical. Both consist of an Arduino Nano, interface with a Drok motor controller, and communicate via I2C to a master. Code was quickly developed for these nodes, and positional control of motors was demonstrated to work as expected.

3.2.5 Power Distribution

With a more complex network, more thought needed to be given to how power would be delivered to the different pieces of the project. The motor drivers took 12V to run the motors, so each driver would get a direct line to the battery. Along with powering the motor the drivers also had a regulated 5V output. These outputs could be used to power the nodes that already had to interface with each driver to control the motors. The HX711 drivers that read the load cells in the Nub needed 5v to operate. These drivers draw a little less than 1.5mA, so the MSP432 launchpad regulated 5v output could power them. The MSP432 itself would be powered via the USB cable used to communicate to it via Uart.

3.2.6 The Middle Joint

At this point, motors were being integrated into the mechanical system. This integration brought to the team's attention that the middle joint in the planar arm was completely isolated. There was no way to run wiring as both sides of the joint had the ability to rotate 360 degrees. In order to get wiring to this joint, a 6 channel slip ring was purchased. Of course, a problem with slip rings is that they produce a lot of electrical noise. For this reason, it was decided that it would be better to pass sensor and motor wiring through the slip ring rather than I2C communication lines.

This did not solve all of the issues however. The joint used an encoder to keep track of position, which required encoder pulses to be passed across the slip ring. Attempting to do this without any kind of filtering proved disastrous; the read position of the encoder would increase ticks by the thousands as the mechanism moved. A debouncing circuit consisting of a schmitt trigger and other filtering elements was constructed and attached to the output of the slip ring. This helped, but proved to be not enough. With a rapidly approaching deadline, the decision was made to go analog. By using an analog signal, permanent loss of position could not happen. A missed read in one instance would be available in the next, unlike a pulse which is gone forever once missed. Additionally, capacitive filters could be added to an analog signal to further assist in noise reduction.

The plan was to use an arduino on one side of the encoder to keep track of encoder pulses. This nano would produce a position, and then encode it into two analog channels via 2 dacs. This analog signals would be passed over the slip ring, read by the I2C network connected node via 2 adcs, and then bit shifted back together to be used for positional control.

For the initial attempt an R2R ladder network of resistors was constructed to be used as a dac, and 2 small low pass filters were placed on the analog lines after the slip ring. The noise proved to be too strong, as the position would fluctuate by about 10 degrees.

The original design had 2 analog lines which could be classified as a MSB and LSB line. Extremely small amounts of noise on the MSB line could cause huge changes in the final position, as the noise was essentially amplified by a 6 bit left shift. However, if the MSB bits were distributed in the higher ranges of both lines, then the noise would not affect the final result as strongly. So before interfacing with the DACs, the nano would split the MSB and LSB line values into two parts each. The MSB values and LSB values were then paired, keeping the most significant bits out of reach from noise. This turned out to be highly effective, and the new accuracy of this joint was about half of a degree. A later revision had the 2R ladder converted to 2 10 bit I2C DACs to better reduce noise and to clean up the system as a whole.

3.2.7 PCB Design

All development for the electronics had been done first on breadboards and then perfboard. While fine for initial development, these implementations do not do well on moving systems. Vibrations from the links moving, whether during operation or relocation of the system as a whole, would cause wiring to become loose and faulty. For this reason, all circuits on the device were assembled onto PCB boards so that they would remain intact, organized, and less susceptible to noise. The first PCB design was for the nodes. Both types of nodes were very similar, so the PCB was designed such that it could be configured for either type. Additionally, it was seen that some nodes would be physically very close together, so it was decided to have 1 PCB hold two nodes. With two nodes on one PCB, a plug in was created such that the motor drivers would plug directly into the board, eliminating wiring and centralizing the node's peripherals. Screw terminals were used to attach external devices to the board, and female headers for the controllers were used for quick and easy troubleshooting. Additionally, the Drok motor drivers came with the added feature of an enable pin. When grounded, this pin disconnects power from the motors. Terminals were added to this enable pin so they could be used as a safety device later.

The next PCB designed for the system was a MSP432 breakout board. The MSP432 to connect 10 IO pins to the Nub, as well as have enough I2C attach points to control all the nodes. The PCB design was attached to the Node PCB such that they could be ordered as one unit. Holes were designed between the boards so that they could be snapped apart. For schematics and board layouts, please see appendix a.

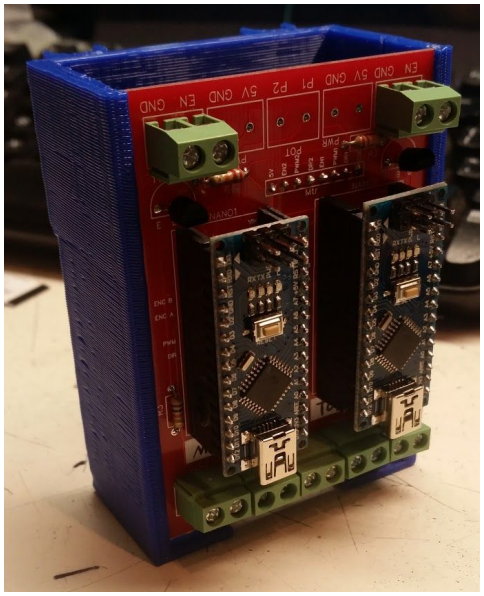


Figure 18: Fully Assembled Dual Node Board

3.2.8 Troubleshooting

Once fully assembled and testing began using kinematics to control the mechanism, it was noticed that the middle link seemed to jerk around when passing over 0 degrees (for example, going to 355 to 15 degrees). After examination of the DAC circuit, noise levels, and arduino code, it was decided that the reference voltage levels for the adc and dac must be different. A quick voltmeter test on Vref of the Arduino ADC and on Vcc of the DAC confirmed this suspicion. The next question was why this was the case. Further research revealed that the function of Vin on the Nano had been misinterpreted. It was thought that this was an unregulated voltage input for the microcontroller, and therefore a regulated 5v signal needed to be passed to it. In reality, Vin is a regulated input, and requires a minimum of 9v. With the Drok motor driver providing only 5v to Vin on the Nano, it was no wonder why the reference voltage was incorrect. The trace connecting 5v to Vin on the PCB was cut, and a wire was placed to jump 12v to vin. After this change the middle joint began behaving correctly. Each of the other nodes were carefully checked to see if this low voltage was having an adverse effect on them. Luckily, no other issues were detected related to the low voltage.

Once the system was deemed operable, testing of user input began. While the Nub was able to be used to move the linkage, there was what felt like a 2 second delay between an action and reaction. This was far too high to be considered even remotely acceptable. By placing print statements within the python script and MSP432 controller, it was found that a majority of the delay was coming from reading the Nub. In order for a read to be complete, each cell was read multiple times, averaged together, and then returned. On top of that the HX711 require a delay proportional to the rate at which the devices are read before the next read can be started. In order to speed this process up, the Nub was moved onto its own node, such that reads can happen concurrently with other processes. This, along with refinement of the python

script, dropped the delay to half a second. This change resulted in the final architecture shown below.

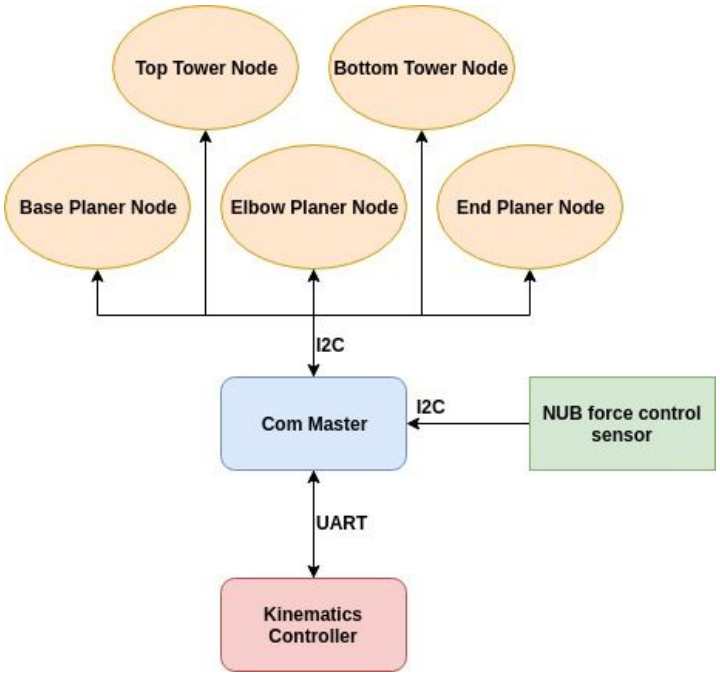


Figure 19: Final Electrical Architecture

Better user input exposed just how jerky the system really was. When user input was delayed, it was hard to see just how the linkage reacted to forces. With a quicker response time it became very clear that the linkage made extremely jerky movements. Mechanical causes were ruled out, as the links could be turned smoothly when not powered on. A closer look at the code on the motor control nodes revealed that 2 significant digits were being dropped when measurements were converted to degrees. Additionally, the communication protocol did not allocate enough space to send a larger message in one transfer. During this time another phenomena had been occuring where the MSP432 would seem to stop responding. This would prevent new setpoints being sent to the nodes and make the system freeze in place. On average, the system would remain live for 1 to 2 minutes before freezing. Examination of the MSP432 in debug mode showed that the execution would block when a node failed to respond on the I2C network. It was suspected that interrupts caused by encoder interrupts or the millis timer on the nodes would cause I2C to miss data. Solving this issue would require a lot of testing and research, as it was not known how the Arduino Wire I2C library functioned, and if it used interrupts of its own.

With one week left before the project deadline, it was decided that only one issue could realistically be addressed. With jerk being an actual project goal, this issue absolutely needed to be addressed, and a large overhaul sensor related code, I2C parameters and Uart protocol was started. With an increase of movement resolution, the mechanism moved much smoother and felt far more controllable.

On the day of the project presentation, a metal shard fell onto the MSP432, shorting it out and permanently damaging it. By the end of the day, the conversion the MSP432 code over

to a spare Arduino Mega had been completed. Since doing this, the I2C crashes have stopped. Likely, this is because the Arduino Wire I2C library is non-blocking, and corrupted messages are ignored. This allows the system to recover from a bad or missed transaction.

3.3 Control System

3.3.1 Actuator Control

Control is manifested over the joint angles of the robot using motors to actuate the joints. These motors have either encoders or potentiometers on them to provide positional feedback. This positional feedback allows the actual physical joint angles of the robot to be known at any point in time. Each joint can be commanded to an absolute setpoint and is positionally controlled using a PID loop. This PID loops function is to reduce the error between a positional setpoint and the current measured position of the joint to 0. To accomplish this it varies the strength and direction of the PWM signal that is sent to the motor. The PWM signal determines how hard a motor will try to drive a joint to move in a given direction.

The PID controller is highly dependant on its tuning to satisfactorily achieve its desired results. The proportional, integral, and derivative terms need to have their gain coefficients calibrated on a joint by joint basis. These also need to be calibrated based on changes to the mechanics, such as adding weight or changing link lengths. The P-term for each link was adjusted to the point where each motor would fight to hold its position when an external force was applied, and it would move close to the position it was commanded to move to. The I-term then takes care of correcting any steady state error caused by friction or gravity. The D-Term is used to prevent any overshoot and dampen any oscillations. In practice, the robot ended up needing very little damping from the d term since the mechanics of the system naturally damped the motion quite a bit.

3.3.2 Forward Kinematics

The forward kinematics was completed using DH parameters and transformation matrices as intended, but then opted to move away from this solution and find a geometric solution for several reasons. The trigonometric expression that θ_K stands in for tended to make the final transformation matrices very messy. The primary reason for the change is that cleaner, simpler formulas were wanted for when it came time to calculate the inverse kinematics. Finding the geometric solution started with the 5 bar section of the mechanism. The expression for θ_K is found once again by splitting the 5 bar into triangular sections and using several trig laws. Using θ_3 and θ_K (which is in terms of θ_3 and θ_4), the pitch and the Z position can be calculated. These two angles are also used to calculate the horizontal position component of the whole tower assembly, which is used to give the projection of the tower assembly onto the x-y plane. The two base links and the tower projection can then be treated as a 3 link planar arm, allowing the x, y, and yaw to be calculated using θ_0, θ_1 and θ_2, and the projection of the tower onto the X-Y plane.

3.3.3 Inverse Kinematics

For inverse kinematics, an inverted jacobian matrix was created. Several issues came up with its implementation. Mostly, there were problems with computational efficiency which primarily came from calculation of the jacobian matrix. One immediate issue with this method was the existence of singularities in the robot's workspace. These singularities are specific points where the jacobian matrix becomes singular, and therefore noninvertible. To work around this,these points would need to be identified, and then hardcoded in solutions for them would need to be added. Something similar for the areas surrounding singularities would also need to be implemented, since the jacobian becomes far less accurate and more distorted when approaching a singularity. Additionally, with the kinematic structure of the robot the calculation of the jacobian itself was actually rather expensive. This is mostly due to the complexity the 5 bar linkage adds. Changes in θ_3 or θ_4 can technically affect x, y, z, and pitch movement depending on the position of the robot, so calculating the jacobian matrix can require evaluating partial derivatives of up to 8 complicated trigonometric functions that describe these relations every time the jacobian is generated. Lastly the practice of iteratively repeating the inverse kinematics until the answer is found can be extremely inefficient, since it requires both the full forward kinematic calculations and inverse jacobian calculations to be completed multiple times.

Because of these issues, the controls were once again moved to a geometric solution. In the end this came with the additional benefits of making the mathematical implementation more easily readable and allowing some geometric tricks to be used to simplify the calculations. Like the forward kinematics, the inverse kinematics were divided into a vertical section and a ground plane section. For the 5 bar linkage, θ_3 and θ_4 can be found using the Z and pitch of the end effector. With the 5 bar linkage, there can technically be multiple inverse kinematic solutions for certain positions, so constraints were added to the problem that allow for only one solution. These constraints try to provide solutions that all exist on the same side of any grashof conditions, which helps prevent the linkage from moving through potentially damaging positions. Once the values for θ_3 and θ_4 are known, they can be used to calculate the projection of the tower assembly onto the X-Y plane. The X, Y, and yaw can then be used to determine the values of θ_0, θ_1 and θ_2. The lower planar assembly is then evaluated like a typical 3 link planar arm, where the the base links are the first two links and the tower projection is the third link. The two base links are the same length, so this allows isosceles triangle geometry to be used as a shortcut to reduce computational load. The 3 link planar arm will give 2 solutions ("elbow left" or "elbow right") for most positions in the workspace. To determine which of these solutions would be optimal, the inverse kinematics can take in the current joint angles of the robot and determine which solution requires less movement to achieve. To determine the closet solution, the absolute values of the differences between the current and proposed values for θ_0, θ_1 and θ_2 are summed, then the solution which has the smaller total distance to move is selected.

3.3.4 Interpreting user input

The user's intent needed to be inferred from the force data gathered by the nub, then acted upon by the robot. There were many control schemes which could have been used to

accomplish this, but a very straightforward and simple approach was opted for. The implementation is essentially a proportional positional controller. When the user applies a force, they are indicating that they want the robot to be in a position that differs from the current position, so the applied force can be interpreted as a metric of positional error for proportional feedback control. In this control scheme, the robot generates positional setpoints in the worldspace based on the current position of the robot summed with an additional displacement. This displacement is in the same direction as the force applied and has a proportional magnitude as well. In a control loop that runs at a consistent rate, a constant force will result in a constant “velocity” of the end effector, since it will take a consistent sized step each cycle. The actual velocity is technically not constant, because it is only sent positional setpoints, but with a high enough resolution it was found to be hardly noticeable.

Chapter 4: Conclusions and Future Work

4.1 End Product

The result of this work was a 5 degree of freedom actuated linkage that could be moved through the application of force to a detached gauntlet. This linkage is capable of supporting 5 lbs at its end effector, and is capable of being accurate in its movements toa half of a degree. While functional, interpretation of the sensing device is still somewhat problematic, and does not allow for very precise movements. Response time of the system is an estimated 750ms, and can be fully powered by a 12v battery. Currently, the system is mounted to a small wooden table for stability and has exposed electronics.

4.2 Next Steps

4.2.1 Mechanical Changes

The Mechanical system works close to how it was intended. There are 2 main issues that can be addressed to improve the system. The first issue is between the first and second link in the planar mechanism. The attachment point between the two is a shaft with a set screw. This leaves some deflection on the shaft and flex on the overall system. This can be fixed by using a larger shaft on the joint. The larger shaft will limit the deflection of the shaft allowing the links to be fastened closer together.

The second issue is the joint below the tower. The use of threaded rod on the tower causes the tower to be able to flex about the z axis. The design of the joint forcing the use of threaded rods. If the joint used a gear system, the motor would not have to be centered on the joint, and the joint could have a much sturdier structure supporting the tower. This would reduce the flexing about the z axis of the tower.

A general improvement could also be the elimination of set screws in the design. Currently, most mechanical backlash in the system is caused by these screws loosening up. Creating a more secure connection would go far in increasing the stability of the system.

4.2.2 Electrical Changes

While reliable, the electronics on this system can certainly be improved. The biggest hindrance the current architecture has is the restriction of speed. Faster microcontrollers could be used on the nodes for more accurate PID tuning and better response times. Reading the Nub is a very time intensive procedure that could benefit from a rework. While adding another controller seems like overkill, perhaps a faster or lower bit ADC in combination with a more efficient reading process would be more efficient.

Another change related to the Nub is its hysteresis problem. Currently, the body of the Nub is made out of 3d printed PLA plastic, while the load cells are made out of aluminum. The

differences in modulus of elasticity can cause erroneous measurments, and the plastic can deform in ways that change what forces the load cells experience under no load. Specifically, the plastic will shift and catch on the screws affixing it to the load cells, causing plastic deformation on the body, applying forces to the load cells.

The current solution to achieving accurate readings of the middle joint is cumbersome at best. A better debouncing technique can replace the current solution to achieve more accuracy and les noise. Furthermore, the PCBs for all the nodes can be revised such that all nanos receive 12v at vin instead of the sub-optimal 5v input.

A large change that can be made is the integration of a better safety system. All nodes have the ability to disable motor power, an input simply needs to be wired to the enable terminal. Additionally, the nano is wired via the PCB directly to the motor enable. This provides two different routes when trying to integrate a safety mechanism. A particularly interesting implementation would be to use current sensing on the motors to determine if they are applying an excess force to the environment. If so, that motor can be disabled to prevent damage.

The top links of the linkage can have improved accuracy with a better resolution adc. Right now, only a 10 bit read is used on the potentiometers, which diminishes the possible accuracy of those links.

4.2.3 Control Changes

There are also several alternatives to the proportional positional control that was used when it comes to handling the higher level user input interpretation. These each have their own strengths and weaknesses which merit further investigation. One such method of control is a PID position/velocity control hybrid, that commands nodes to reach both desired velocities and positions. Using some sort of weighted combination of the signals resulting from the positional and velocity feedback loops could result in more fine control over the motion of the mechanism. Another potential alternative is direct force amplification with active gravity compensation. This would essentially be a direct mapping between force applied by the user and the force the mechanism applies. For instance, if the user applied a 0.1lb force in a direction, then they could experience a 1lb force in that direction applied by the mechanism. In order for this to be implemented an efficient jacobian to relate end effector forces and joint torques would be needed. Lastly, the positional awareness of the robot could be used to help make inferences about what the user is trying to accomplish. If points of interest like the mouth and plate hove known coordinates in the global reference frame, then the robot can assume the user wants to move to either of these points and help guide their hand. This could be achieved by splitting the workspace up into a coordinate system, and having guiding force/velocity vectors corresponding to each region on the grid system. The absolute best solution is likely a hybrid solution that draws from some of these control schemes.

The communications and the kinematics are currently two of the largest contributors to round trip response time. One cheap way to quickly improve response time is to increase the baud rate of both the UART and the I^2C connections. This will automatically enable faster data transfer if the hardware supports it. Increasing these speeds can lead to an increased chance for corruption, so the actual amount these baud rates can be increased will need to be tested.

Another possible change is converting the format of the positional messages passed over UART. Currently, each joint angle is sent over the serial port as an ascii representation. This is incredibly useful for debugging and manually controlling the robot via serial terminal, but it is far less efficient than sending raw integer numbers. Each joint value requires six bytes to be transmitted using the ascii format, whereas they would only need bytes per value an unsigned integer format. This would also mean the comms controller would not need to spend time parsing and converting between ascii and integer formats. The kinematics code, like the communications code, has already gone through a decent amount of optimizations, but could still benefit from further refinement. A way to further optimize the code on the offboard computer is parallelization of processing tasks. Currently the code executes in a serialized manner, but this could be changed so that kinematics and communication can run simultaneously using multithreading. Splitting the kinematics portion itself into smaller parts that can run in parallel would also lead to an increase in speed.

Appendix

A. Electrical Schematics and PCB Layouts

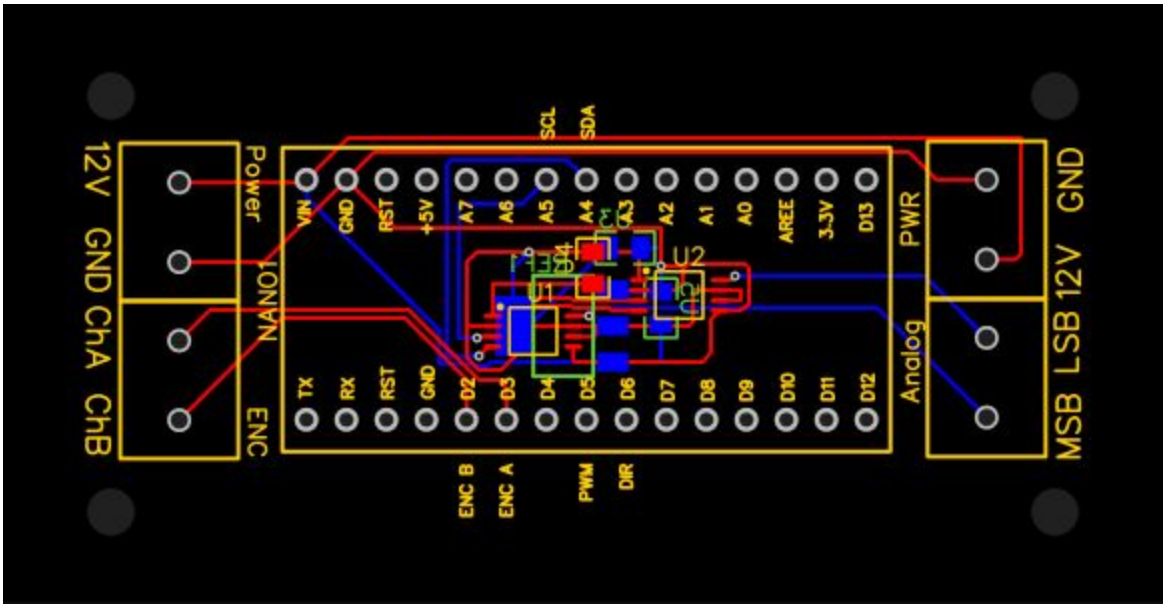


Figure 20: Interrupt to DAC PCB Layout

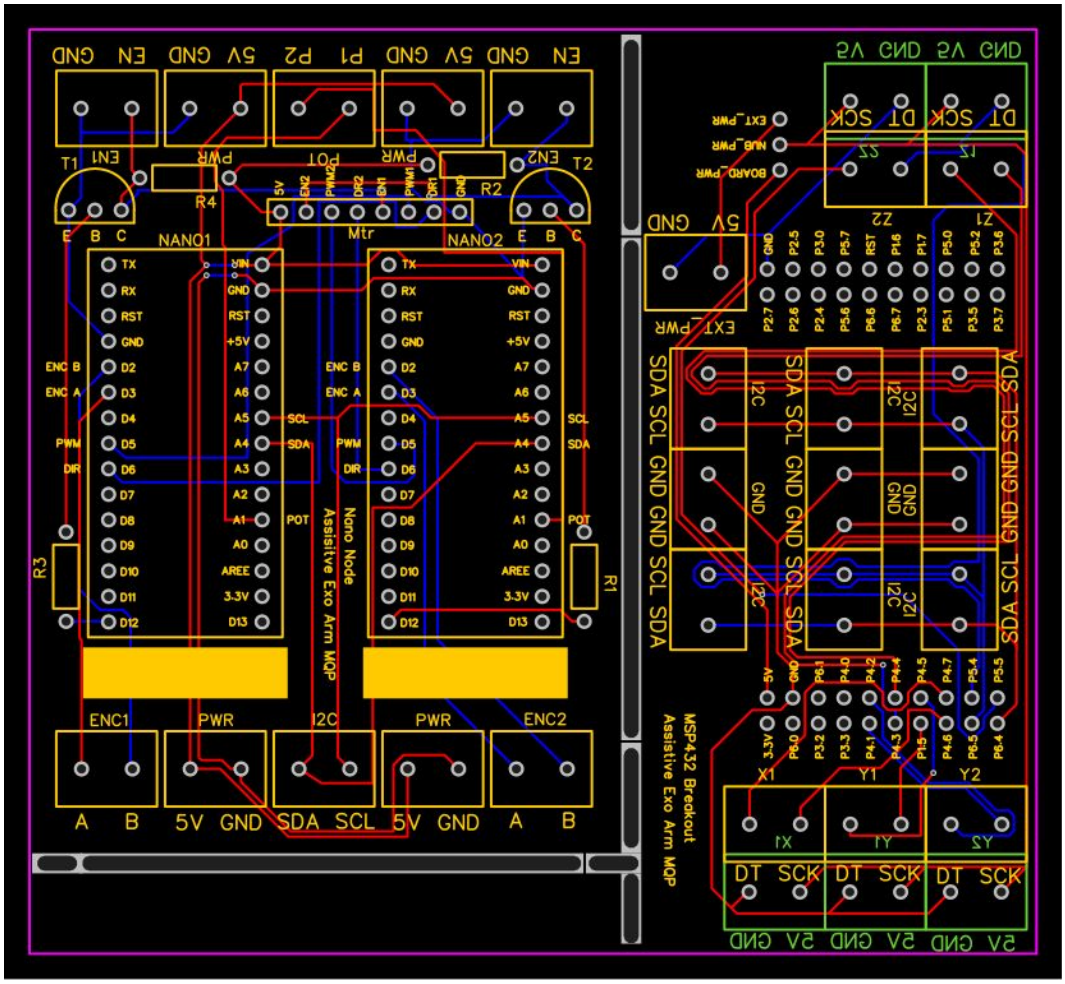


Figure 21: Node and MSP432 Breakout PCB Layouts

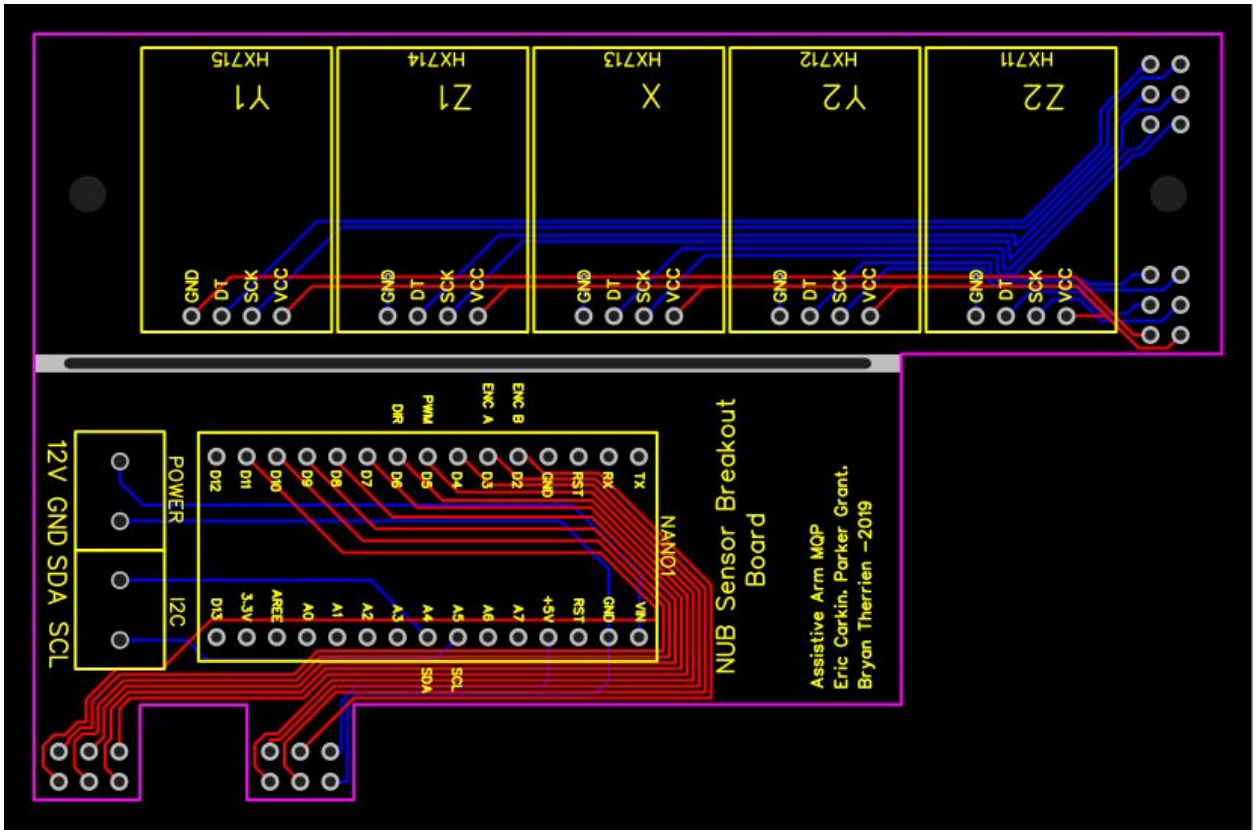


Figure 22: Nub PCB Layout

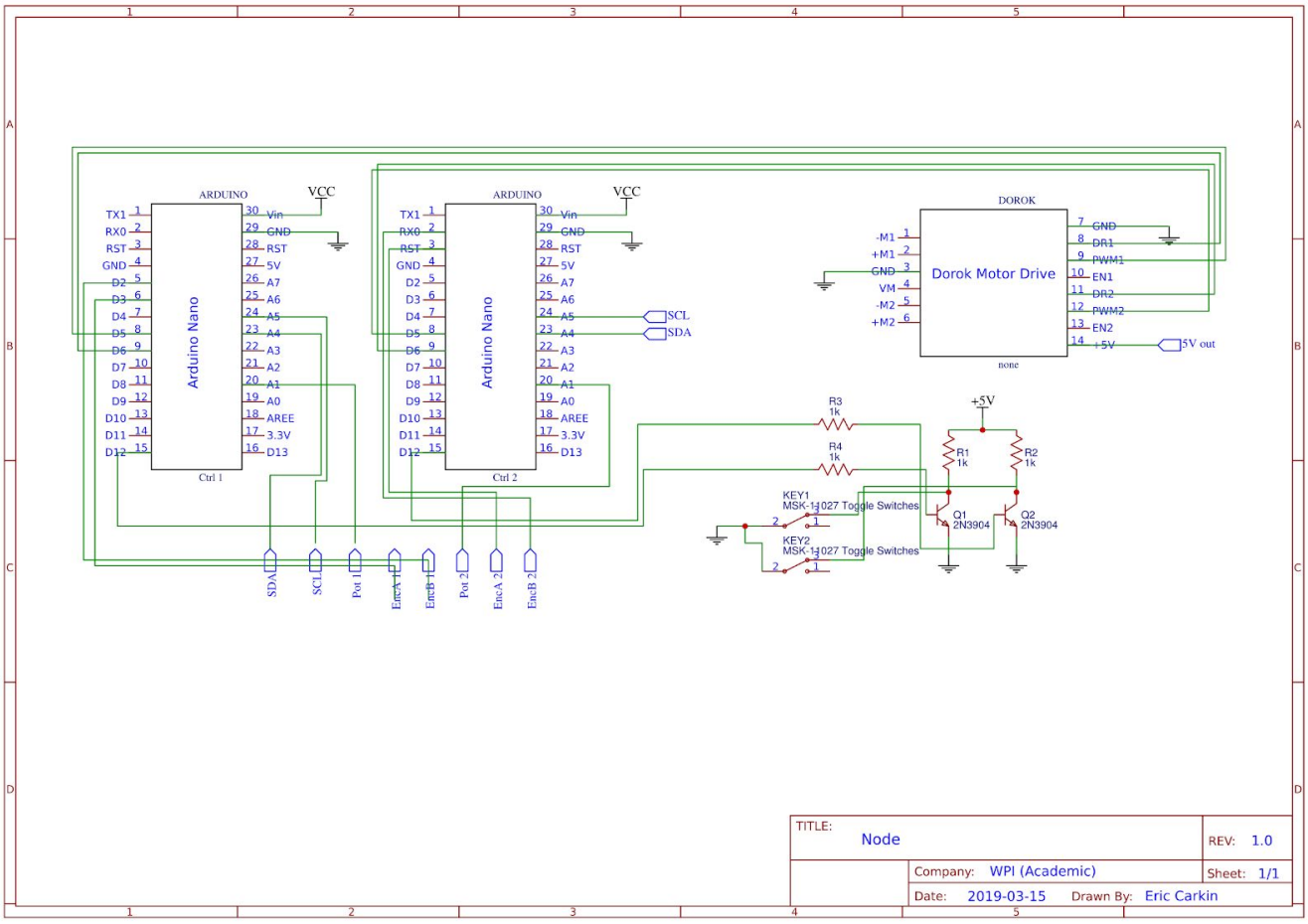


Figure 23: Dual Node Schematic

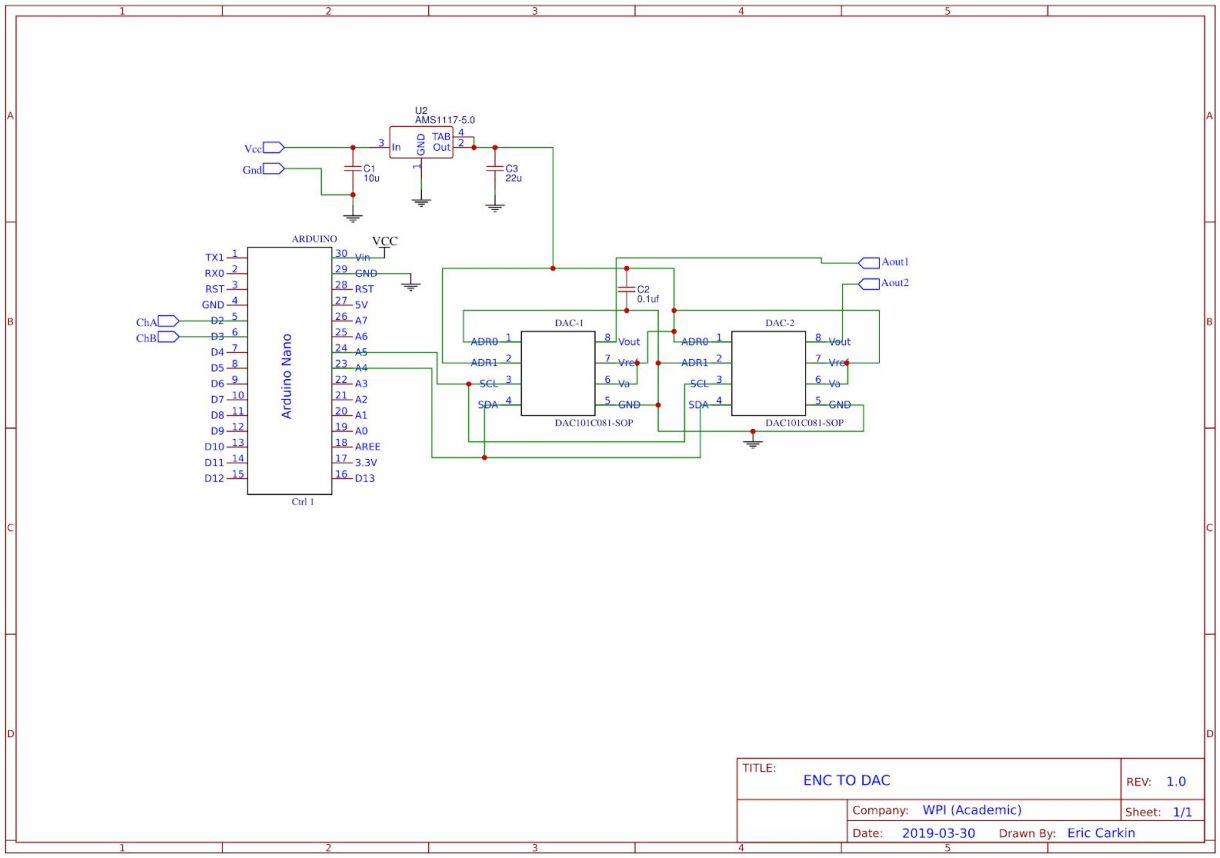


Figure 24: Interrupt to DAC Schematic

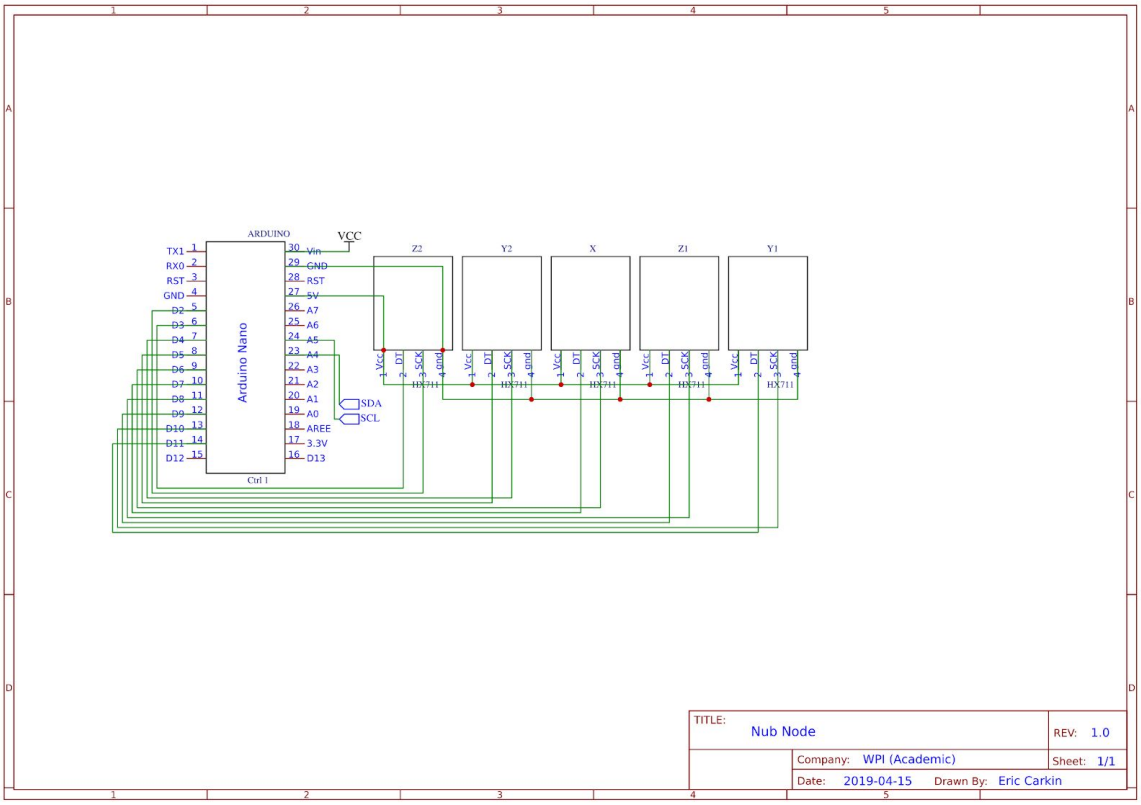


Figure 25: Nub Schematic

B. Coordinate Frame Reference

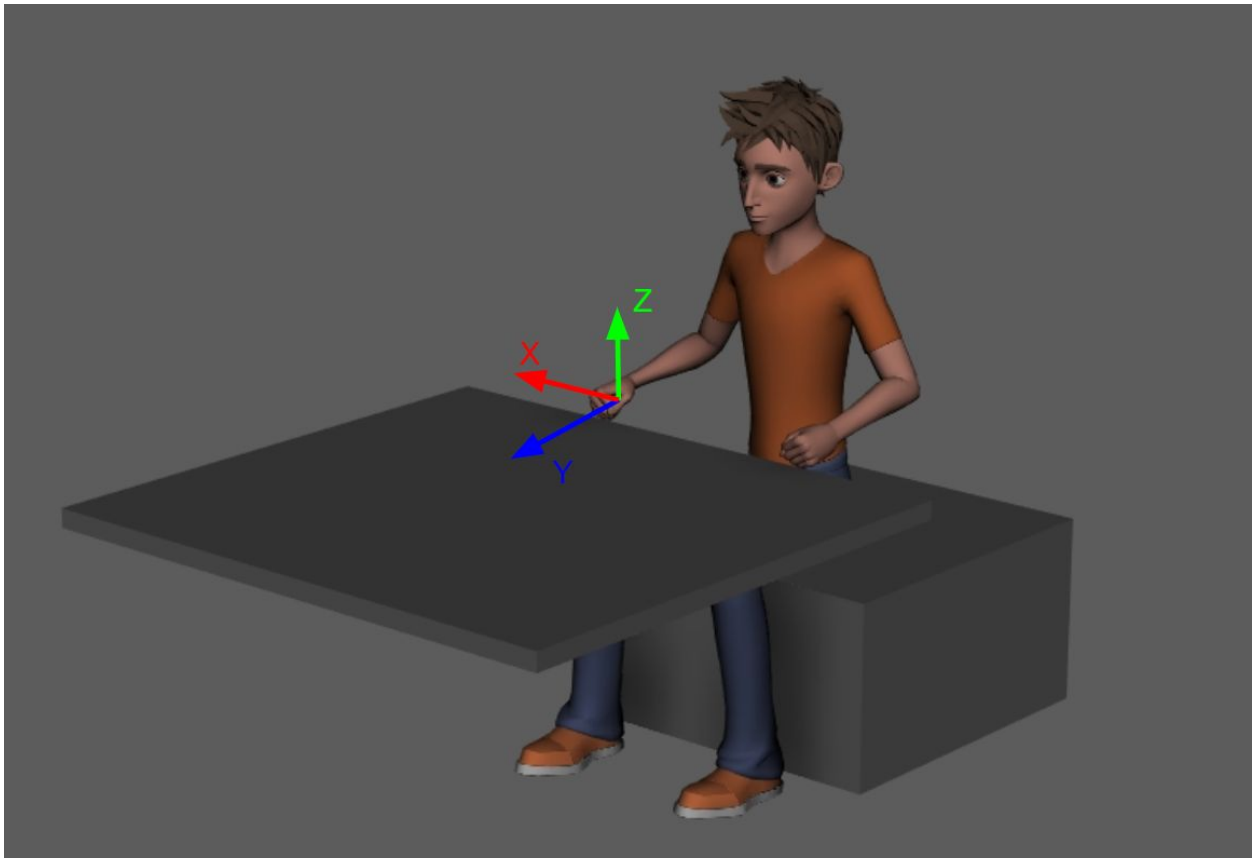


Figure 26: Reference Coordinate Frame

C. Encoder_Node Code

```
#include <Wire.h>
//#define SLAVE_ADDRESS    0x21  //TopTower
//#define SLAVE_ADDRESS    0x22  //BottomTower
#define SLAVE_ADDRESS    0x23  //base motor furthest
//#define SLAVE_ADDRESS    0x24  //base motor elbow
//#define SLAVE_ADDRESS    0x25  //base motor stationary

#define RECIEVED_SIZE    4
#define SENT_SIZE        20
#define PWM_PIN          5
#define DIR_PIN          6
#define POT_PIN          A1
#define ENC_PIN          3
#define ENC2_PIN         2
#define ENC_TICKS        6400
byte recievedSetPoint[RECIEVED_SIZE];
byte sentPosition[SENT_SIZE];
volatile long setPoint = 0;
volatile long currPosition = 0;
int error = 0;
volatile bool enc1 = true;
volatile bool enc2 = true;
volatile bool lastEnc = true;
volatile bool encDir = true;
volatile long ticks = 0;
float p = 0;
float i = 0;
float d = 0;
unsigned long prev_time = 0;
unsigned long curr_time = 0;
float curr_error = 0.0;
float prev_error = 0.0;
float integral_error = 0.0;
float deriv_error = 0.0;
unsigned long dt = 1;

void fixTicks(){
 ticks = (ticks+ENC_TICKS*2)%ENC_TICKS;
}

void encISR(){
 enc1 = digitalRead(ENC_PIN);
 enc2 = digitalRead(ENC2_PIN);
 if(!enc1 != !enc2) {
  ticks++;
 }
 else{
  ticks--;
 }
 fixTicks();
}

void encISR2(){
 enc1 = digitalRead(ENC_PIN);
 enc2 = digitalRead(ENC2_PIN);
 if(!enc1 != !enc2) {
  ticks--;
 }
 else{
  ticks++;
 }
 fixTicks();
}

void setup() {
 pinMode(ENC_PIN, INPUT);
 pinMode(ENC2_PIN, INPUT);


 enc1 = digitalRead(ENC_PIN);
 enc2 = digitalRead(ENC2_PIN);

 attachInterrupt(digitalPinToInterrupt(ENC2_PIN), encISR2, CHANGE);
```

```
 attachInterrupt(digitalPinToInterrupt(ENC_PIN), encISR, CHANGE);

 Wire.begin(SLAVE_ADDRESS);
 Wire.onRequest(requestEvent);
 Wire.onReceive(receiveEvent);
 pinMode(PWM_PIN, OUTPUT);
 pinMode(DIR_PIN, OUTPUT);

 // bottom stationary
 if(SLAVE_ADDRESS == 0x25){
 p = 0.4;
 i = 0.003;
 d = 0.001;
 }

 //bottom tower
 if(SLAVE_ADDRESS == 0x23){
 p = 0.251;
 i = 0.00084;
 d = 0.0010;
 }
 setPoint = 0;
}


int asdf = 0;
long duty;
void loop() {

 currPosition = ticks;

 error = (setPoint - currPosition);

 if (error>(ENC_TICKS/2)){
 error = error-ENC_TICKS;
 }
 else if (error<-(ENC_TICKS/2)){
 error = error+ENC_TICKS;
 }
 else{
 error = error;
 }

 curr_error = error;
 dt = (2) + 1;
 integral_error += (dt)*curr_error; //need to prevent overflow
 if(integral_error > 10000){
 integral_error = 10000;
 }

 if(integral_error < -10000){
 integral_error = -10000;
 }

 deriv_error = (curr_error - prev_error)/(dt);
 int  duty = (p*curr_error + i*integral_error + d*deriv_error);
 prev_time = curr_time;

 if(duty < 0){
 duty = duty*-1;
 digitalWrite(DIR_PIN, HIGH);
 }
 else{
 digitalWrite(DIR_PIN, LOW);
 duty = duty;
 }

 if(duty > 130){
 duty = 130;
 }
 analogWrite(PWM_PIN, duty);
}

void requestEvent(){
 get_byte_position();
 Wire.write(sentPosition, SENT_SIZE);
}
```

```
void receiveEvent(int bytesReceived){
 for(int i = 0; i < bytesReceived; i++)
 {
  if(i < RECIEVED_SIZE)
  {
   recievedSetPoint[i] = Wire.read();
  }
  else
  {
   Wire.read();
  }
 }
 setPoint = (((recievedSetPoint[1])<<8) + recievedSetPoint[0])&0xFFFF;
 setPoint = ENC_TICKS*((float)setPoint/36000);
 bool state = digitalRead(13);
 digitalWrite(13, !state);
}

void get_byte_position(){
 sentPosition[0] = (((int)((float)currPosition *36000/ENC_TICKS))& 255);
 sentPosition[1] = (((int)((float)currPosition *36000/ENC_TICKS))& (255<<8))>>8;
}
```

D. Potentiometer_Node Code

```
#include <Wire.h>
#define SLAVE_ADDRESS    0x21  //TopTower
//#define SLAVE_ADDRESS    0x22  //BottomTower
//#define SLAVE_ADDRESS    0x23
//#define SLAVE_ADDRESS    0x24
//#define SLAVE_ADDRESS    0x25
#define RECIEVED_SIZE    4
#define SENT_SIZE        20
//#define SENT_SIZE        4
#define PWM_PIN          5
#define DIR_PIN          6
#define POT_PIN          A1
#define ENC_PIN          8
byte recievedSetPoint[RECIEVED_SIZE];
byte sentPosition[SENT_SIZE];
volatile float setPoint = 0;
float currPosition = 0;
// position control var
float p = 0;
float i = 0;
float d = 0;
int Lower_Bound = 0;
int Upper_Bound = 0;
unsigned long prev_time = 0;
unsigned long curr_time = 0;
float curr_error = 0.0;
float prev_error = 0.0;
float integral_error = 0.0;
float deriv_error = 0.0;
unsigned long dt = 1;

void setup() {
 Wire.begin(SLAVE_ADDRESS);
 Wire.onRequest(requestEvent);
 Wire.onReceive(receiveEvent);
 pinMode(PWM_PIN, OUTPUT);
 pinMode(DIR_PIN, OUTPUT);
 currPosition = (map(analogRead(POT_PIN), 0, 1023, 0,807))/3;
 setPoint = currPosition;
 //top_tower
 if (SLAVE_ADDRESS == 0x21){
  p = 7;
  i = 0.003;
  d = 0.01;
  Lower_Bound = 115;
  Upper_Bound = 170;
 }
//bottom tower
 else if (SLAVE_ADDRESS == 0x22){
  p = 3;
  i = 0.004;
```

```
   d = 0;
   Lower_Bound = 35;
   Upper_Bound = 105;
  }
}

void loop() {
  currPosition = (map(analogRead(POT_PIN), 0, 1023, 0,807))/3;
  curr_error = setPoint - currPosition;
  dt = (2) + 1;
  integral_error += (dt)*curr_error; //need to prevent overflow
  if(integral_error > 10000){
   integral_error = 10000;
  }

 if(integral_error < -10000){
  integral_error = -10000;
 }
  deriv_error = (curr_error - prev_error)/(dt);
int  duty = (p*curr_error + i*integral_error + d*deriv_error);
  prev_time = curr_time;
   if(duty < 0){
   duty = duty*-1;
   digitalWrite(DIR_PIN, HIGH);
  }
  else{
   digitalWrite(DIR_PIN, LOW);
   duty = duty + 10;
  }
  if(duty > 90){
   duty = 90;
  }
  analogWrite(PWM_PIN, duty);
}

void requestEvent(){
  get_byte_position();
  Wire.write(sentPosition, SENT_SIZE);
}

void receiveEvent(int bytesReceived){
  for(int i = 0; i < bytesReceived; i++)
  {
    if(i < RECIEVED_SIZE)
    {
      recievedSetPoint[i] = Wire.read();
    }
    else
    {
      Wire.read();
    }
  }
  setPoint = (((recievedSetPoint[1])<<8) + recievedSetPoint[0])/100;
  if (setPoint > 360){
   setPoint = (map(analogRead(POT_PIN), 0, 1023, 0,807))/3;
  }
  if(setPoint < Lower_Bound){
   setPoint = Lower_Bound;
  }
  if(setPoint > Upper_Bound){
   setPoint = Upper_Bound;
  }
  bool state = digitalRead(13);
  digitalWrite(13, !state);
}

void get_byte_position(){
  sentPosition[0] = (((int)(currPosition*100)) & 255);
  sentPosition[1] = (((int)(currPosition*100)) & (255<<8))>>8;
}
```

E. Middle_Joint DAC

```
#include <Wire.h>
#define ENC_PIN         3
#define ENC2_PIN        2
#define ENC_TICKS       6400
#define MASTER_ADDRESS    0x30  //nano addr
#define LSB_DAC_ADDR      0x0A  //LSB addr
#define MSB_DAC_ADDR      0x4D  //MSB addr
volatile bool lastEnc = true;
volatile bool encDir = true;
volatile long ticks = 0;
volatile bool enc1 = true;
volatile bool enc2 = true;

void encISR(){
   enc1 = digitalRead(ENC_PIN);
 enc2 = digitalRead(ENC2_PIN);
 if(!enc1 != !enc2) {
   ticks++;
 }
 else{
   ticks--;
 }
 fixTicks();
}

void encISR2(){
 enc1 = digitalRead(ENC_PIN);
 enc2 = digitalRead(ENC2_PIN);
 if(!enc1 != !enc2) {
   ticks--;
 }
 else{
   ticks++;
 }
 fixTicks();
}
void fixTicks(){
 noInterrupts();
 ticks = (ticks+ENC_TICKS*2)%ENC_TICKS;
 interrupts();
}
void writeDAC() {
 Serial.println(ticks);
 int LSB_Val = (ticks & (0xF)) + ((ticks & 0x380)>>3);
 LSB_Val = LSB_Val << 3;
 int MSB_Val = (((ticks & 0x70)>>4) + ((ticks & 0x1C00)>>7))<<4;
 //data frame is 16 bits, 0000[data][data][00]
 byte MSB_Frame[2];
 MSB_Frame[0] = (MSB_Val & 0x3C0)>>6;
 MSB_Frame[1] = (MSB_Val & 0x3F)<<2;

 byte LSB_Frame[2];
 LSB_Frame[0] = (LSB_Val & 0x3C0)>>6;
 LSB_Frame[1] = (LSB_Val & 0x3F)<<2;
 Wire.beginTransmission(MSB_DAC_ADDR);
 Wire.write(MSB_Frame[0]);
 Wire.write(MSB_Frame[1]);
 Wire.endTransmission();
 Wire.beginTransmission(LSB_DAC_ADDR);
 Wire.write(LSB_Frame[0]);
 Wire.write(LSB_Frame[1]);
 Wire.endTransmission();
}
void setup() {
 pinMode(ENC_PIN, INPUT);
 pinMode(ENC2_PIN, INPUT);
 Wire.begin(MASTER_ADDRESS);
 enc1 = digitalRead(ENC_PIN);
 enc2 = digitalRead(ENC2_PIN);
 attachInterrupt(digitalPinToInterrupt(ENC2_PIN), encISR2, CHANGE);
 attachInterrupt(digitalPinToInterrupt(ENC_PIN), encISR, CHANGE);
```

```
 Serial.begin(9600);
}
void loop() {
 writeDAC();
}
```

F. Middle_Joint Node

```
#include <Wire.h>
//#define SLAVE_ADDRESS    0x21  //TopTower
//#define SLAVE_ADDRESS    0x22  //BottomTower
//#define SLAVE_ADDRESS    0x23  //base motor furthest
#define SLAVE_ADDRESS    0x24  //base motor elbow
//#define SLAVE_ADDRESS    0x25  //base motor stationary
#define RECIEVED_SIZE    4
#define SENT_SIZE        20
#define PWM_PIN          5
#define DIR_PIN          6
#define ENC_TICKS        6400
#define DAC_LSB_PIN      A1
#define DAC_MSB_PIN      A2
// planar elbow
float p = 0.2;
float i = 0.00015;
float d = 0.001;
int Lower_Bound = 35;
int Upper_Bound = 105;
long prev_time = 0;
long curr_time = 0;
int curr_error = 0.0;
int prev_error = 0.0;
float integral_error = 0.0;
float deriv_error = 0.0;
long dt = 1;
byte recievedSetPoint[RECIEVED_SIZE];
byte sentPosition[SENT_SIZE];
volatile long setPoint = 0;
volatile long currPosition = 0;
volatile bool enc1 = true;
volatile bool enc2 = true;
volatile bool lastEnc = true;
volatile bool encDir = true;
volatile long ticks = 0;

void readDAC(){
 long x1x1 = analogRead(DAC_LSB_PIN);
 long x1 = (x1x1 & 0x3F0);
 long x2x2 = analogRead(DAC_MSB_PIN)+2;
 long x2 = (x2x2 & (0x3F0));
 long Low_MSB = (x1 & 0x380) >> 7;
 long Low_LSB = (x1 & 0x78) >> 3;
 long High_MSB = (x2 & 0x380) >> 4;
 long High_LSB = (x2 & 0x70);
 long MSB = Low_MSB + High_MSB;
 long LSB = Low_LSB + High_LSB;
 ticks = LSB + (MSB << 7);
}


void setup() {
 Wire.begin(SLAVE_ADDRESS);
 Wire.onRequest(requestEvent);
 Wire.onReceive(receiveEvent);
 pinMode(PWM_PIN, OUTPUT);
 pinMode(DIR_PIN, OUTPUT);
 pinMode(DAC_LSB_PIN, INPUT);
 pinMode(DAC_MSB_PIN, INPUT);
 setPoint = 0;
}
int reportedPos = 0;
int asdf = 0;
long duty;

void loop() {
```

```
 readDAC();
 currPosition = ticks;
 int error = (setPoint - currPosition);
 if (error>ENC_TICKS/2){
  error = error-ENC_TICKS;
 }
 else if (error<-ENC_TICKS/2){
  error = error+ENC_TICKS;
 }
 else{
  error = error;
 }
 curr_error = error;
 dt = (2) + 1;
 integral_error += (dt)*curr_error; //need to prevent overflow
 if(integral_error > 100000){
  integral_error = 100000;
 }
 if(integral_error < -100000){
  integral_error = -100000;
 }
 deriv_error = (curr_error - prev_error)/(dt);
 int  duty = (p*curr_error + i*integral_error - d*deriv_error);
 prev_time = curr_time;
 if(duty < 0){
 duty = duty*-1;
 digitalWrite(DIR_PIN, HIGH);
 }
 else{
 digitalWrite(DIR_PIN, LOW);
 duty = duty;
 }
 if(duty > 180){
 duty = 180;
 }
 analogWrite(PWM_PIN, duty);
}

void requestEvent(){
 get_byte_position();
 Wire.write(sentPosition, SENT_SIZE);
}

void receiveEvent(int bytesReceived){
 for(int i = 0; i < bytesReceived; i++)
 {
  if(i < RECIEVED_SIZE)
  {
   recievedSetPoint[i] = Wire.read();
  }
  else
  {
   Wire.read();
  }
 }
 setPoint = ((recievedSetPoint[1]<<8) + recievedSetPoint[0])&0xFFFF;
 setPoint = ENC_TICKS*((float)setPoint/36000);
 bool state = digitalRead(13); // um ok
 digitalWrite(13, !state);
}

void get_byte_position(){
 sentPosition[0] = (((int)((float)currPosition *36000/ENC_TICKS))& 255);
 sentPosition[1] = (((int)((float)currPosition *36000/ENC_TICKS))& (255<<8))>>8; // lol clever
}
```

G. Nub Node

```
#include "HX711.h"
#include <Wire.h>
#define SLAVE_ADDRESS 0x50 //NUB
#define RECIEVED_SIZE    4
#define SENT_SIZE        20
```

```
// HX711 circuit wiring
const int z2_S = 2;
const int z2_D = 3;
const int y2_S = 4;
const int y2_D = 5;
const int x_S = 6;
const int x_D = 7;
const int z1_S = 8;
const int z1_D = 9;
const int y1_S = 10;
const int y1_D = 11;
const int avg_buffer_size = 5;
int buffer_index = 0;
long avg_buffer[5][avg_buffer_size];
long y1 = 0;
long y2 = 0;
long z1 = 0;
long z2 = 0;
long x = 0;
long y2_final = 0;
long y1_final = 0;
long z2_final = 0;
long z1_final = 0;
long x_final = 0;
byte sentReadings[20];
HX711 z1_cell;
HX711 z2_cell;
HX711 y1_cell;
HX711 y2_cell;
HX711 x_cell;

void setup() {
 Wire.begin(SLAVE_ADDRESS);
 Wire.onRequest(requestEvent);
// Wire.onReceive(receiveEvent);
 z2_cell.begin(z2_D, z2_S);
 z1_cell.begin(z1_D, z1_S);
 y2_cell.begin(y2_D, y2_S);
 y1_cell.begin(y1_D, y1_S);
 x_cell.begin(x_D, x_S);
 Serial.begin(9600);

}

void loop() {
 if (y2_cell.is_ready()) {
   y2 = y2_cell.read();
 }
 if (y1_cell.is_ready()) {
   y1 = y1_cell.read();
 }
   if (z2_cell.is_ready()) {
   z2 = z2_cell.read();
 }
   if (z1_cell.is_ready()) {
   z1 = z1_cell.read();
 }
   if (x_cell.is_ready()) {
   x = x_cell.read();
 }
 buffer_index++;
 buffer_index = buffer_index%avg_buffer_size;
 avg_buffer[0][buffer_index] = y2;
 avg_buffer[1][buffer_index] = z2;
 avg_buffer[2][buffer_index] = x;
 avg_buffer[3][buffer_index] = y1;
 avg_buffer[4][buffer_index] = z1;
 y2 = 0;
 y1 = 0;
 x = 0;
 z1 = 0;
 z2 = 0;

for(int i = 0; i<avg_buffer_size; i++){
 y2 += avg_buffer[0][i];
 z2 += avg_buffer[1][i];
 x += avg_buffer[2][i];
 y1 += avg_buffer[3][i];
```

```
 z1 += avg_buffer[4][i];
}

y2_final = y2/5;
y1_final = -(y1/5);
z2_final = z2/5;
z1_final = z1/5;
x_final = x/5;
delay(100);
}


void requestEvent(){
 get_readings();
 Serial.print(sentReadings[0]);
 Serial.print(sentReadings[1]);
 Serial.print(sentReadings[2]);
 Serial.println(sentReadings[3]);
 Wire.write(sentReadings, SENT_SIZE);
 bool state = digitalRead(13);
 digitalWrite(13, !state);
}

void get_readings(){
 sentReadings[3] = (char)(y2_final & 0xFF);
 sentReadings[2] = (char)((y2_final & (0xFF00))>>8);
 sentReadings[1] = (char)((y2_final & (0xFF0000))>>16);
 sentReadings[0] = (char)((y2_final & (0XFF000000))>>24);
 sentReadings[7] = (char)(z2_final & (255));
 sentReadings[6] = (char)((z2_final & (0xFF00))>>8);
 sentReadings[5] = (char)((z2_final & (0xFF0000))>>16);
 sentReadings[4] = (char)((z2_final & (0xFF000000))>>24);
 sentReadings[11] = (char)(x_final & (255));
 sentReadings[10] = (char)((x_final & (0xFF00))>>8);
 sentReadings[9] = (char)((x_final & (0xFF0000))>>16);
 sentReadings[8] = (char)((x_final & (0xFF000000))>>24);
 sentReadings[15] = (char)(y1_final & (255));
 sentReadings[14] = (char)((y1_final & (0xFF00))>>8);
 sentReadings[13] = (char)((y1_final & (0xFF0000))>>16);
 sentReadings[12] = (char)((y1_final & (0xFF000000))>>24);
 sentReadings[19] = (char)((z1_final & (255)));
 sentReadings[18] = (char)((z1_final & (0xFF00))>>8);
 sentReadings[17] = (char)((z1_final & (0xFF0000))>>16);
 sentReadings[16] = (char)((z1_final & (0xFF000000))>>24);
}
```

H. Control Script

```
import math
from serial.tools.list_ports import comports
import serial
import time
import math
import matplotlib
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

class Comms:

    def __init__(self):
        self.serialPort = serial.Serial()
        self.availableDevices = []
        self.availableDeviceDescriptions = []
        self.availablePorts = []
        for i in comports():
            self.availablePorts.append(i)
            self.availableDevices.append(i.device)
            self.availableDeviceDescriptions.append(i.description)
        # [self.serialPort.setPort(i.device) for i in self.availablePorts if i.description == "1234567876543234567"]
        # [self.serialPort.setPort(i.device) for i in self.availablePorts if i.device == "COM12" or i.device == "COM14"]
        for i in self.availablePorts:
            if i.device == "COM12" or i.device == "COM13" or i.device == "COM14" or i.device == "COM16":
                # self.serialPort.setPort(i.device)
                print(i)
```

```
            self.serialPort = serial.Serial(i.device)
        self.serialPort.baudrate = 115200
        # self.robot = Robot

    def registerRobot(self, robot):
        print("regstart")
        self.robot = robot
        self.robot.comms = self
        # self.serialPort.open()
        # self.serialPort.write(b'123')
        time.sleep(1)
        self.serialPort.flushInput()
        time.sleep(1)

        print("regend")

    def parseLine(self):
        if self.serialPort.is_open:

            self.serialPort.timeout = None
            # self.serialPort.write("000-000+000+000+000".encode("ASCII", "ignore"))



            v1 = str(abs(int(self.robot.jointTargets[0]*100))).zfill(5)
            v2 = str(abs(int(self.robot.jointTargets[1]*100))).zfill(5)
            v3 = str(abs(int(self.robot.jointTargets[2]*100))).zfill(5)
            v4 = str(abs(int(self.robot.jointTargets[3]*100))).zfill(5)
            v5 = str(abs(int(self.robot.jointTargets[4]*100))).zfill(5)

            if int(v2)>17000:
                v2 = "17000"
                print("v2 capped at 170")

            if int(v2)<11500:
                v2 = "11500"
                print("v2 capped at 115")

            if int(v1)<3500:
                v1 = "03500"
                print("v2 capped at 035")
            if int(v1)>10500:
                v1 = "10500"
                print("v2 capped at 105")
            outline = (" " + v1 + " " + v2 + " " + v3 + " " + v4 + " " + v5).encode("ASCII", "ignore")
            self.serialPort.write(outline)
            print(outline)



            # message = self.serialPort.read(5*6+20)
            message = self.serialPort.read(50)
            print(message)

            # print(str(outline)+" "+str(message))
            joints = [0]*5
            for i in range(5):
                joints[i] = int(message[i*6:i*6+6].decode("ASCII"))

            # nub = [0]*5
            # for i in range(5):
            #     nub[i] = (message[i*4-19]<<16) + (message[i*4-18]<<8) + message[i*4-17]
            #     if message[i * 4 - 20]==0 :
            #         nub[i] = nub[i]
            #     else:
            #         nub[i] = -nub[i]

            # nub = [0]*5
            # for i in range(5):
            #     nub[i] = message[(i*4-20):(i*4-16)]
            #
            # nub = [int.from_bytes(b, byteorder='big', signed=True) for b in nub]

            nub = [0]*5
            nub[0] = message[30:34]
            nub[1] = message[34:38]
            nub[2] = message[38:42]
            nub[3] = message[42:46]
```

```
            nub[4] = message[46:50]
            # print(nub[4])
            nub = [int.from_bytes(b, byteorder='big', signed=True) for b in nub]

            # print(joints)
            # print(nub)

            self.robot.joints = joints
            self.robot.nub = nub

    def printPortDebug(self):
        self.serialPort = serial.Serial()
        self.availablePorts = comports()

        for i in self.availablePorts:
            print("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~")
            print("Prod: "+str(i.product))
            print("Dev: "+str(i.device))
            print("Desc: "+str(i.description))
            print("Name: "+str(i.name))
            print("Location: "+str(i.location))
            print("Int: "+str(i.interface))
            print("Man: "+str(i.manufacturer))
            print("Prod: "+str(i.product))
        # self.availablePortNames = [i.name for i in self.availablePorts]
        # self.availablePortLocations = [i.location for i in self.availablePorts]
        # self.availablePortDescs = [i.description for i in self.availablePorts]
        # self.availablePortDevs = [i.device for i in self.availablePorts]
        # self.availablePortProds = [i.product for i in self.availablePorts]


class Robot:
    # ####################################
    # ### OG
    # ####################################
    # __TL = 13.
    # __BL = 14.
    # __GL = 2.5
    # __FL = 2.
    # __LL = 4.
    # __T = 21.
    # __HH = 1.
    # __HL = 2.
    # __L0 = 4.
    # __L1 = 4.

    ####################################
    ### 3-30-2019
    ####################################
    # __HL = 0.2   #?
    # __HH = 1.   #?

    __TL = 11.25
    __BL = 12.375
    __GL = 2.48
    __FL = 2.5
    __LL = 3.375
    __T  = 22.  #?
    __HH = 0.0   #?
    __HL = 0.0   #?
    __L0 = 4.   #?
    __L1 = 4.   #?

    def __init__(self):
        self.nub = [0]*5
        self.joints = [50, 150, 000, 000, 000]
        self.jointTargets = [50, 150, 000, 000, 000]
        self.lastTarget = self.joints.copy()

        self.comms = Comms()
        self.comms.registerRobot(self)
        self.calVals = self.nub.copy()
        self.calibrate()
        # self.calibrateR()

        self.lastgood = self.jointTargets.copy()
        time.sleep(1)
        self.calibrate()
```

```
    # self.calibrateR()
    self.ts = time.time()

    self.free1 = 0
    self.free2 = 0
    self.free3 = 0

def fwdKin(self, TH0, TH1, TH2, TH3, TH4):
        #     function[X, Y, Z, THX, THY, THZ] = Forwardplswork(TH0, TH1, TH2, TH3, TH4, self.HH, self.HL)
        #
        #         self.TL = 13;
        #         self.BL = 14;
        #         self.GL = 2;
        #         self.FL = 2;
        #         self.LL = 4;
        #         self.t = 21;
        #
        #         PPY = -GL - self.LL * math.cos(TH4);
        #         PPX = self.LL * math.sin(TH4);
        #
        #         PQY = self.TL * math.sin(TH3);
        #         PQX = self.TL * math.cos(TH3);

    PPY = -self.__GL - self.__LL * math.cos(TH4)
    PPX = self.__LL * math.sin(TH4)

    PQY = self.__TL * math.sin(TH3)
    PQX = self.__TL * math.cos(TH3)


        #
        #         P = sqrt(PPY ** 2 + PPX ** 2);
        #         Q = sqrt((PPY - PQY) ** 2 + (PPX - PQX) ** 2);
        #
        #         ALPHA = acos((Q ** 2 + self.TL ** 2 - P ** 2) / (2 * Q * self.TL));
        #         BETA = acos((Q ** 2 + self.FL ** 2 - self.BL ** 2) / (2 * Q * self.FL));
        #
        #         THK = ALPHA + BETA - pi / 2;
        #

    P = math.sqrt(PPY ** 2 + PPX ** 2)
    Q = math.sqrt((PPY - PQY) ** 2 + (PPX - PQX) ** 2)

    ALPHA = math.acos((Q ** 2 + self.__TL ** 2 - P ** 2) / (2 * Q * self.__TL))
    BETA = math.acos((Q ** 2 + self.__FL ** 2 - self.__BL ** 2) / (2 * Q * self.__FL))

    THK = ALPHA + BETA - math.pi / 2



    # ######################################################################
    # % -THY = TH3 + THK
    # % Z = self.t + 12 * math.sin(TH3) - self.HH * math.cos(TH3 + THK) + self.HL * math.sin(TH3 + THK)
    # ######################################################################

    Z = self.__T + self.__TL * math.sin(TH3) - self.__HH * math.cos(TH3 + THK) + self.__HL * math.sin(TH3 + THK)
    THY = -(TH3 + THK)

    dt = self.__TL * math.cos(TH3) + self.__HH * math.sin(-THY) + self.__HL * math.cos(-THY)

    THZ = TH0 + TH1 + TH2

    X = self.__L0 * math.cos(TH0) + self.__L1 * math.cos(TH0 + TH1) + dt * math.cos(THZ)
    Y = self.__L0 * math.sin(TH0) + self.__L1 * math.sin(TH0 + TH1) + dt * math.sin(THZ)

    THX = 0

    return X, Y, Z, THX, THY, THZ

def invKin(self,X, Y, Z, THX, THY, THZ):

    TH3 = math.asin((self.__T - Z - self.__HH * math.cos(-THY) + self.__HL * math.sin(-THY)) / (-self.__TL))
    THK = - THY - TH3
    dt = self.__TL * math.cos(TH3) + self.__HH * math.sin(-THY) + self.__HL * math.cos(-THY)
    P2X = X - dt * math.cos(THZ)
    P2Y = Y - dt * math.sin(THZ)
    THR = math.atan2(P2Y, P2X)
    R = math.sqrt(P2X ** 2 + P2Y ** 2)
```

```
    GAMMA = math.acos((self.__L0 ** 2 + R ** 2 - self.__L1 ** 2) / (2 * self.__L0 * R))
    # % RIGHT
    TH0 = THR - GAMMA
    TH1 = 2 * GAMMA
    # % % LEFT
    # % TH0 = THR + GAMMA;
    # % TH1 = -2 * GAMMA;
    TH2 = THZ - TH0 - TH1
    PKX = self.__TL * math.cos(TH3) + self.__FL * math.sin(TH3 + THK)
    PKY = self.__TL * math.sin(TH3) - self.__FL * math.cos(TH3 + THK)

    F = math.sqrt(PKX ** 2 + (PKY) ** 2)
    K = math.sqrt(PKX ** 2 + (PKY + self.__GL) ** 2)

    ALPHA = math.acos((self.__GL ** 2 + K ** 2 - F ** 2) / (self.__GL * 2 * K))
    BETA = math.acos((K ** 2 + self.__LL ** 2 - self.__BL ** 2) / (K * self.__LL * 2))
    TH4 = math.pi - ALPHA - BETA

    return TH0, TH1, TH2, TH3, TH4

def testKin(self):
    TH0, TH1, TH2, TH3, TH4 = 0, math.pi/2, 0, 30*math.pi/180, 40*math.pi/180
    X, Y, Z, THX, THY, THZ = self.fwdKin(TH0, TH1, TH2, TH3, TH4)
    TH02, TH12, TH22, TH32, TH42 = self.invKin( X, Y, Z, THX, THY, THZ)
    X2, Y2, Z2, THX2, THY2, THZ2 = self.fwdKin(TH02, TH12, TH22, TH32, TH42)

    print(TH0, TH1, TH2, TH3, TH4)
    print(TH02, TH12, TH22, TH32, TH42)

    print(X, Y, Z, THX, THY, THZ)
    print(X2, Y2, Z2, THX2, THY2, THZ2)

    # self.drawf(TH0, TH1, TH2, TH3, TH4)

def fwdKinPTS(self, TH0, TH1, TH2, TH3, TH4):
    PPY = -self.__GL - self.__LL * math.cos(TH4)
    PPX = self.__LL * math.sin(TH4)
    PQY = self.__TL * math.sin(TH3)
    PQX = self.__TL * math.cos(TH3)

    P = math.sqrt(PPY ** 2 + PPX ** 2)
    Q = math.sqrt((PPY - PQY) ** 2 + (PPX - PQX) ** 2)

    ALPHA = math.acos((Q ** 2 + self.__TL ** 2 - P ** 2) / (2 * Q * self.__TL))
    BETA = math.acos((Q ** 2 + self.__FL ** 2 - self.__BL ** 2) / (2 * Q * self.__FL))

    THK = ALPHA + BETA - math.pi / 2

    XPTS = []
    XPTS.append(0)
    XPTS.append(XPTS[0] + self.__L0 * math.cos(TH0))
    XPTS.append(XPTS[1] + self.__L1 * math.cos(TH0 + TH1))
    XPTS.append(XPTS[2] + 0)
    XPTS.append(XPTS[3] + self.__TL * math.cos(TH0 + TH1 + TH2) * math.cos(TH3))
    XPTS.append(XPTS[4] + self.__HH * math.cos(TH0 + TH1 + TH2) * math.sin(TH3 + THK))
    XPTS.append(XPTS[5] + self.__HL * math.cos(TH0 + TH1 + TH2) * math.cos(TH3 + THK))

    YPTS = []
    YPTS.append(0)
    YPTS.append(YPTS[0] + self.__L0 * math.sin(TH0))
    YPTS.append(YPTS[1] + self.__L1 * math.sin(TH0 + TH1))
    YPTS.append(YPTS[2] + 0)
    YPTS.append(YPTS[3] + self.__TL * math.sin(TH0 + TH1 + TH2) * math.cos(TH3))
    YPTS.append(YPTS[4] + self.__HH * math.sin(TH0 + TH1 + TH2) * math.sin(TH3 + THK))
    YPTS.append(YPTS[5] + self.__HL * math.sin(TH0 + TH1 + TH2) * math.cos(TH3 + THK))

    ZPTS = []
    ZPTS.append(0)
    ZPTS.append(ZPTS[0] + 0)
    ZPTS.append(ZPTS[1] + 0)
    ZPTS.append(ZPTS[2] + self.__T)
    ZPTS.append(ZPTS[3] + self.__TL * math.sin(TH3))
    ZPTS.append(ZPTS[4] - self.__HH * math.cos(TH3 + THK))
    ZPTS.append(ZPTS[5] + self.__HL * math.sin(TH3 + THK))
    return [XPTS, YPTS, ZPTS]

def drawf(self, TH0, TH1, TH2, TH3, TH4):
    self.fig = plt.figure()
```

```
    self.ax = self.fig.add_subplot(111, projection='3d', )

    [xs, ys, zs] = self.fwdKinPTS(TH0, TH1, TH2, TH3, TH4)
    self.ax.axis('equal')

    self.ax.plot(xs, ys, zs)
    # print((xs, ys, zs))
    self.fig.show()

def calibrate(self):

    print("\n\n\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~")
    print("nub: " + str(self.nub) + "    Joints deg: " + str(self.joints) + "    Joints targ: " + str(
        self.jointTargets))
    print("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~")
    time.sleep(1)
    self.comms.parseLine()
    cal1 = self.nub.copy()
    time.sleep(1)
    print("\n\n\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~")
    print("nub: " + str(self.nub) + "    Joints deg: " + str(self.joints)+ "    Joints targ: " + str(self.jointTargets))
    print("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~")


    self.comms.parseLine()
    cal2 = self.nub.copy()
    time.sleep(1)
    print("\n\n\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~")
    print("nub: " + str(self.nub) + "    Joints deg: " + str(self.joints) + "    Joints targ: " + str(
        self.jointTargets))
    print("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~")
    time.sleep(1)
    self.comms.parseLine()
    cal3 = self.nub.copy()

    print("\n\n\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~")
    print("nub: " + str(self.nub) + "    Joints deg: " + str(self.joints) + "    Joints targ: " + str(
        self.jointTargets))
    print("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~")
    time.sleep(1)
    self.comms.parseLine()
    cal4 = self.nub.copy()

    print("\n\n\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~")
    print("nub: " + str(self.nub) + "    Joints deg: " + str(self.joints) + "    Joints targ: " + str(
        self.jointTargets))
    print("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~")
    time.sleep(1)
    self.comms.parseLine()
    cal5 = self.nub.copy()

    print("\n\n\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~")
    print("nub: " + str(self.nub) + "    Joints deg: " + str(self.joints) + "    Joints targ: " + str(
        self.jointTargets))
    print("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~")
    time.sleep(1)
    self.calVals = [(cal1[0] + cal2[0] + cal3[0] + cal4[0] + cal5[0]) / 5,
                    (cal1[1] + cal2[1] + cal3[1] + cal4[1] + cal5[1]) / 5,
                    (cal1[2] + cal2[2] + cal3[2] + cal4[2] + cal5[2]) / 5,
                    (cal1[3] + cal2[3] + cal3[3] + cal4[3] + cal5[3]) / 5,
                    (cal1[4] + cal2[4] + cal3[4] + cal4[4] + cal5[4]) / 5]




def demoZ(self):
    self.comms.parseLine()
    # print(self.joints)
    print(self.nub)
    calz1 = self.nub[1]-self.calVals[1]
    calz2 = self.nub[4]-self.calVals[4]
    torque = calz1-calz2
    force = calz1+calz2

    gainF = .0001
    gainT = .00001
```

```
    dZ = gainF*force
    dP = gainT*torque

    # self.world = self.fwdKin(self.joints[2],self.joints[3],self.joints[4],self.joints[1],self.joints[0])
    # self.worldTarg = self.world.copy()
    # self.jointTargets = self.invKin(self.worldTarg)

    self.jointTargets[1] = int(self.joints[1] -dP)
    self.jointTargets[0] = int(self.joints[0] -dZ)

    # POSITIVE DOWN
    # NUB ARRAY 2ND AND 5TH VALUES


    time.sleep(.1)

def inv_kin_closest(self,X, Y, Z, THX, THY, THZ, XTH0, XTH1, XTH2, XTH3, XTH4):

    TH3 = math.asin((self.__T - Z - self.__HH * math.cos(-THY) + self.__HL * math.sin(-THY)) / (-self.__TL))
    THK = - THY - TH3
    dt = self.__TL * math.cos(TH3) + self.__HH * math.sin(-THY) + self.__HL * math.cos(-THY)
    P2X = X - dt * math.cos(THZ)
    P2Y = Y - dt * math.sin(THZ)
    THR = math.atan2(P2Y, P2X)
    R = math.sqrt(P2X ** 2 + P2Y ** 2)
    GAMMA = math.acos((self.__L0 ** 2 + R ** 2 - self.__L1 ** 2) / (2 * self.__L0 * R))

    # RIGHT
    TH0R = THR - GAMMA
    TH1R = 2 * GAMMA

    # LEFT
    TH0L = THR + GAMMA
    TH1L = -2 * GAMMA

    TH2R = THZ - TH0R - TH1R
    TH2L = THZ - TH0L - TH1L

    leftError = (TH0L - XTH0) ** 2 + (TH1L - XTH1) ** 2 + (TH2L - XTH2) ** 2
    rightError = (TH0R - XTH0) ** 2 + (TH1R - XTH1) ** 2 + (TH2R - XTH2) ** 2

    if rightError > leftError:
        TH0 = TH0L
        TH1 = TH1L
        TH2 = TH2L
    else:
        TH0 = TH0R
        TH1 = TH1R
        TH2 = TH2R

    PKX = self.__TL * math.cos(TH3) + self.__FL * math.sin(TH3 + THK)
    PKY = self.__TL * math.sin(TH3) - self.__FL * math.cos(TH3 + THK)

    F = math.sqrt(PKX ** 2 + (PKY) ** 2)
    K = math.sqrt(PKX ** 2 + (PKY + self.__GL) ** 2)

    ALPHA = math.acos((self.__GL ** 2 + K ** 2 - F ** 2) / (self.__GL * 2 * K))
    BETA = math.acos((K ** 2 + self.__LL ** 2 - self.__BL ** 2) / (K * self.__LL * 2))
    TH4 = math.pi - ALPHA - BETA

    return TH0, TH1, TH2, TH3, TH4


def fwd_planar_partial_kin(self, TH0, TH1):
    P2X = self.__L0 * math.cos(TH0) + self.__L1 * math.cos(TH0 + TH1)
    P2Y = self.__L0 * math.sin(TH0) + self.__L1 * math.sin(TH0 + TH1)

    return P2X, P2Y

def inv_planar_partial_kin_closest(self,P2X, P2Y, XTH0, XTH1):

    THR = math.atan2(P2Y, P2X)
    R = math.sqrt((P2X ** 2) + (P2Y ** 2))
    GAMMA = math.acos((self.__L0 ** 2 + R ** 2 - self.__L1 ** 2) / (2 * self.__L0 * R))

    # RIGHT
    TH0R = THR - GAMMA
```

```
    TH1R = 2 * GAMMA

    # LEFT
    TH0L = THR + GAMMA
    TH1L = -2 * GAMMA

    # leftError = ((TH0L - XTH0 + (2*math.pi)) % (2*math.pi)) ** 2 + ((TH1L - XTH1 + (2*math.pi)) % (2*math.pi)) ** 2
    # rightError = ((TH0R - XTH0 + (2*math.pi)) % (2*math.pi)) ** 2 + ((TH1R - XTH1 + (2*math.pi)) % (2*math.pi)) ** 2

    # leftError = abs((TH0L - XTH0 + (2*math.pi)) % (2*math.pi)) + abs((TH1L - XTH1 + (2*math.pi)) % (2*math.pi))
    # rightError = abs((TH0R - XTH0 + (2*math.pi)) % (2*math.pi)) + abs((TH1R - XTH1 + (2*math.pi)) % (2*math.pi))
    leftError = abs((TH0L - XTH0 + (2*math.pi)) % (2*math.pi))
    if leftError>math.pi:
        leftError = math.pi*2 - leftError
    rightError = abs((TH0R - XTH0 + (2*math.pi)) % (2*math.pi))
    if rightError>math.pi:
        rightError = math.pi*2 - rightError

    # print("TH0L, TH0R, XTH0: " +str([TH0L, TH0R, XTH0])+" TH1L, TH1R, XTH1: "+str( [TH1L, TH1R, XTH1])+" P2X, P2Y: "+str([P2X, P2Y]))

    # if rightError > leftError:
    if True:
        # print("arm left!    LE: " +str(leftError)+" RE: "+str(rightError))
        TH0 = TH0L
        TH1 = TH1L
        # TH2 = TH2L
    else:
        # print("arm right!    LE: " +str(leftError)+" RE: "+str(rightError))
        TH0 = TH0R
        TH1 = TH1R
        # TH2 = TH2R

    valid = R < (self.__L0+self.__L1-0.1)

    return (TH0 + (2*math.pi))%(2*math.pi), (TH1 + (2*math.pi))%(2*math.pi), valid

def lastgood327(self):
    # print("Returning to last good position")
    #
    # self.jointTargets = self.
    home = [85, 150, 0, 0, 0]
    print("moving a bit towards home")
    # self.jointTargets = self.lastgood
    calibrated = [(9*a_i+ b_i)/10 for a_i, b_i in zip(self.lastgood,home)]
#     MAKE THIS CLOSED ROTATION AVERAGE
#     THIS WILL BREAK THINGS AT SEAMS OF PLANAR MOTION IN ITS CURRENT STATE
    self.jointTargets = calibrated.copy()

def rotationalAverage(self,a,b,weight_a=1,weight_b=1):
    diff = ((a - b + 180 + 360 + 360) % 360) - 180
    angle = (360 + 360 + b + (diff * weight_a / (weight_a + weight_b))) % 360
    return angle

def lastgoodPLANAR(self):
    # print("Returning to last good position")
    #
    # self.jointTargets = self.
    home = [85, 150, 180, 180, 180]
    print("moving a bit towards home")
    # self.jointTargets = self.lastgood
    calibrated = [ self.rotationalAverage(a_i,b_i,9,1) for a_i, b_i in zip(self.lastgood,home)]
    self.jointTargets = calibrated.copy()

def simplifiedNubControlDemo(self):
    # time.sleep(.2)
    # time.sleep(1)
    # time.sleep(.1)

    #   0------------------------0
    #   |
    #   |
    #   |
    #   |      The Robot links are expected to start
    #   |      in this orientation for the demo
    #   |
    #   |
    #   |
    #   |
```

```
#   |
#   0

self.comms.parseLine()


calNub = [a_i - b_i for a_i, b_i in zip(self.nub,self.calVals)]
# calNub = [a_i - b_i for a_i, b_i in zip(self.nub,[0]*len(self.nub))]


"""nub = [y1, z1, x, y2, z2]"""
forces = [calNub[2],
          calNub[0] + calNub[3],
          calNub[1] + calNub[4],
          0,
          calNub[1] - calNub[4],
          calNub[0] - calNub[3]]
        # [X, Y, Z, THX, THY, THZ]
# forces = [0 if abs(force) < 20 else force for force in forces[0:3]] +[0 if abs(torque) < 10000 else torque for torque in forces[3:6]]

# gains = [0, 0.0000025, 0, 0, 0, 0]
# gains = [0, 0.000005, 0, 0, 0, 0]
# gains = [0, 0, 0, 0, 0, 0]
# gains = [0.000005, 0, 0, 0, 0, 0]

# gains = [0.000005, 0.000005, 0, 0, 0, 0]
# gains = [0.00000, 0.00000, 0, 0, 0, 0.000005]
# gains = [0.000005, 0.000005, 0, 0, 0, 0.000005]


# gains = [0.000005, 0.0000025, 0, 0, 0, 0.000005]
# gains = [0, 0.0000025, 0, 0, 0, 0.0000025]


# gains = [0.00000, 0.00000, 0.0000025, 0, 0.0000007, 0.00000]
# gains = [0.000005, 0.0000025, 0, 0, 0, 0.000005]
# gains = [0.000005, 0.0000025, 0, 0, 0, 0.000005]
# gains = [0.000005, 0.0000025, 0, 0, 0, 0.000005]

# gains = [0.000005, 0.0000025, 0.0000025, 0, 0.0000007, 0.000005]
#
# gains = [0.000004, 0.0000025, 0.0000035, 0, 0.0000007, 0.00001]
# gains = [0.000004, 0.0000025, 0.00000, 0, 0.000000, 0.0000]
# gains = [0.00000035, 0.00000025, 0.000004, 0, 0.0000003, 0.000005]
# gains = [0.0000035, 0.0000025, 0.000004, 0, 0.000007, 0.000005] # last good
# gains = [0.0000035, 0.0000025, 0.00000, 0, 0.0000, 0.000005]
# gains = [0.0000035, 0.0000025, 0.00000, 0, 0.0000, 0.000000]
# gains = [0.00000, 0.00000, 0.000025, 0, -0.000037, 0.000000]

# gains = [0.00000, 0.0000025, 0.00000, 0, 0.0000, 0.000000]
# ##############################################
# gains = [-0.0000008, -0.000001, 0.000008, 0, -0.0000005, 0.000005]
# gains = [0.00000035, 0.00000025, 0.00000, 0, 0.0000, 0.0000005]
# FULL BEST WORKING ######################################
# gains = [-0.0000008, -0.000001, 0.000008, 0, -0.0000005, 0.000005]
# gains = [-0.0000008, -0.000001, 0.000008, 0, -0.0000005, 0.000005]
# gains = [-0.0000008, -0.000001, 0.000008, 0, -0.0000005, 0.000005]
# gains = [-0.0000008, -0.000001, 0.000008, 0, -0.0000005, 0.000005]

# gains = [-0.000002, -0.000001, 0.000008, 0, -0.000000, 0.00000]
gains = [-0.0000025, -0.000001, 0.00000, 0, -0.000000, 0.00000]
# gains = [-0.00000, -0.00000, 0.00000, 0, -0.000000, 0.00000]
gains = [-0.000002, -0.000001, 0.000008, 0, -0.0000005, 0.000005]

# gains = [-0.000000, -0.00000, 0.00000, 0, -0.00000, 0.0000085]



deltas = [a_i * b_i for a_i, b_i in zip(gains,forces)]

# deltas = [(1.5 * abs(delta) / delta) if abs(delta) > 1.5 else delta for delta in deltas]
# deltas = [(.15 * abs(delta) / delta) if abs(delta) > .15 else delta for delta in deltas]
# deltas = [(.05 * abs(delta) / delta) if abs(delta) > .05 else delta for delta in deltas]
# deltas = [(.1 * abs(delta) / delta) if abs(delta) > .1 else delta for delta in deltas]
deltas = [(.2 * abs(delta) / delta) if abs(delta) > .2 else delta for delta in deltas[0:3]] +[(4 * abs(delta) / delta) if abs(delta) > 4 else delta for delta in deltas[3:6]]

# deltas = [.005,0,0,0,0,0]
```

```
try:

    # GROOM JOINTS

    # groomedJoints = self.joints.copy()
    groomedJoints = self.lastTarget.copy()
    jointOffsets = [85, 150, 0, -90, 0]

    groomedJoints[0] = (groomedJoints[0] - 85)
    groomedJoints[1] = (groomedJoints[1] - 150)
    groomedJoints[2] = groomedJoints[2]
    groomedJoints[3] = -((groomedJoints[3]+90)%360)
    # groomedJoints[4] = 0
    groomedJoints[4] = groomedJoints[4]


    groomedJoints = [deg * math.pi / 180 for deg in groomedJoints]
    TH0 = groomedJoints[2]
    TH1 = groomedJoints[3]
    try:
        P2X, P2Y = self.fwd_planar_partial_kin(groomedJoints[2],groomedJoints[3])
        P2X = P2X + deltas[0]
        P2Y = P2Y + deltas[1]
        TH0, TH1, valid = self.inv_planar_partial_kin_closest(P2X,P2Y,groomedJoints[2],groomedJoints[3])
    except Exception as e:
        print(e)
        valid = False
        print("planar kin dead")
    # print("TH2, deltas[5],TH2 + deltas[5]: "+str([TH2, deltas[5],TH2 + deltas[5]]))

    TH3 = groomedJoints[1]
    TH4 = groomedJoints[0]
    try:
        xxx, yyy, zzz, txxx, tyyy, tzzz = self.fwdKin(1.1*math.pi,1.1*math.pi,1.1*math.pi, groomedJoints[1], groomedJoints[0])
        # print("hi2.1")
        zzz = zzz-deltas[2]
        tyyy = tyyy+deltas[4]
        not_used, not_used, not_used, TH3, TH4 = self.invKin(xxx, yyy, zzz, txxx, tyyy, tzzz)
        # not_used, not_used, not_used, TH3, TH4 = self.invKin(xxx, yyy, 21, txxx, tyyy, tzzz)

        # print("hi2.3")

    except Exception as e:
        print(e)
        print("zeds ded")
    except:
        print("zeds ded")



    # jointTargs = [0, 0, TH0, TH1, 0]
    jointTargs = [TH4, TH3, TH0, TH1, 0]
    jointTargs = [xyxyxy * 180 / math.pi for xyxyxy in jointTargs]


    self.free1 = (self.free1 + deltas[5] + 360)

    jointTargs[0] = jointTargs[0]+85
    jointTargs[1] = jointTargs[1]+150
    jointTargs[2] = jointTargs[2]
    jointTargs[3] = (((-jointTargs[3])-90)+720)%360
    # jointTargs[4] = 0
    # jointTargs[4] = ((jointTargs[2] - jointTargs[3]) + 720) % 360
    jointTargs[4] = ((jointTargs[2] - jointTargs[3]) + self.free1 + 720) % 360
    # jointTargs[4] = TH2

    if jointTargs[0] < 35: jointTargs[0] = 35
    if jointTargs[0] > 105: jointTargs[0] = 105
    if jointTargs[1] < 115: jointTargs[1] = 115
    if jointTargs[1] > 170: jointTargs[1] = 170


    # print("attempted move: " + str([P2X,P2Y]) + " rela: " + str(rela) +" abs: " + str(abso))



    if valid:
        self.jointTargets = jointTargs.copy()
```

```
            else:
                print("invalid position attempted")
                self.jointTargets[4] = jointTargs[4]
            # print("hi4")
            fs = [ "%08.0f"%f for f in forces ]
            # print("forces: " + str(fs).strip("""") + "  joints: " + str(self.joints) + "  targs: " + str(jointTargs) + "  ds: " + str(deltas))
            # print("forces: " + str(fs).strip("""") + " nub: " + str([f'{n:15}' for n in calNub]) + "  joints: " + str(self.joints) + "  targs: " + str(
            #     jointTargs) + "  timestamp: " + str(time.time()-self.ts) + "  ds: " + str(deltas))

            # print("Y: "+fs[2])
            # print("nub: "+str([ "%08.0f"%f for f in calNub]))

            # print("nub: " + str([f'{n:15}' for n in calNub]))
            self.lastTarget = self.jointTargets.copy()


        except Exception as e:
            print("Kin Broken")
            print(e)
            print("joints: " + str(self.joints) + "  targs: " + str(self.jointTargets))

            # self.lastgoodPLANAR()



if __name__ == '__main__':
    r = Robot()
    r.jointTargets = [85, 150, 00, 0, 0]
    last_time = time.clock()
    try:
        while True:
            r.simplifiedNubControlDemo()
            print("Exec time: "+str(time.clock()-last_time)+"s")
            time.sleep(.1)
            last_time = time.clock()
    # except KeyboardInterrupt as e:
    finally:
        r.comms.serialPort.close()
        print("Session Closed")
```

References

Adafruit Industries. (n.d.). MyoWare Muscle Sensor. Retrieved October 16, 2018, from https://www.adafruit.com/product/2699?gclid=Cj0KCQjwgOzdBRDlARIsAJ6_HNl2u339xpMPLqyryKRe34e9JnNOB7hbUT3IT7HBSHq9uhf66ILqKNkaAuwvEALw_wcB

Advancer Technologies. (n.d.). 3-Lead Muscle / Electromyography Sensor for Microcontroller Applications. Retrieved October 5, 2018, from https://cdn-shop.adafruit.com/product-files/2699/AT-04-001.pdf

Advancer Technologies. (n.d.). MyoWare Muscle Sensor. Retrieved October 15, 2018, from http://www.advancertechnologies.com/p/myoware.html

Al-Mutlaq, S. (n.d.). Getting Started With Load Cells. Retrieved October 6, 2018, from https://learn.sparkfun.com/tutorials/getting-started-with-load-cells

Breteler, M., Meulenbroek, R., & Gielen, S. (2002). An Evaluation of the Minimum-Jerk and Minimum Torque-Change Principles at the Path, Trajectory, and Movement-Cost Levels. Retrieved October 1, 2018, from http://www.socsci.ru.nl/meulenbroek/Publications/Breteler Meulenbroek Gielen 2002.pdf

Centers for Disease Control and Prevention. (2018, April 10). Muscular Dystrophy. Retrieved September 28, 2018, from https://www.cdc.gov/ncbddd/musculardystrophy/data.html

Delbiaggio, N. (2017). *A comparison of facial recognition's algorithms.*(Unpublished master's thesis). Haaga-Helia. Retrieved October 1, 2018, from

https://www.theseus.fi/bitstream/handle/10024/132808/Delbiaggio_Nicolas.pdf?sequence=1

Delsys. (2018). Bagnoli Desktop EMG Systems. Retrieved October 8, 2018, from https://www.delsys.com/products/desktop-emg/bagnoli-desktop/

DLF. (n.d.). Neater Eater Robotic V6 - Living made easy. Retrieved October 18, 2018, from https://www.livingmadeeasy.org.uk/eating and drinking/electrically-operated-eating-systems-p/neater-eater-robotic-v6-0119673-1523-information.htm

DLF. (2018). Neater Arm Support - Living made easy. Retrieved October 18, 2018, from https://www.livingmadeeasy.org.uk/eating and drinking/manually-operated-eating-systems-p/neater-arm-support-0037091-1522-information.htm

Exact Dynamics. (n.d.). IArm. Retrieved September 25, 2018, from http://www.exactdynamics.nl/site/?page=iarm

He, S., Craig, B. A., Xu, H., Covinsky, K. E., Stallard, E., Thomas, J., . . . Sands, L. P. (2015, September). Unmet Need for ADL Assistance Is Associated With Mortality Among Older Adults With Mild Disability. Retrieved October 18, 2018, from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4841172/

HTC Sensor. (n.d.). TAL220 Parallel Beam Load Cell. Retrieved October 11, 2018, from https://cdn.sparkfun.com/datasheets/Sensors/ForceFlex/TAL220M4M5Update.pdf

Jaeco Orthopedic. (2018). WREX: Wilmington Robotic EXoskeleton Arm. Retrieved October 5, 2018, from http://jaecoorthopedic.com/products/products/WREX:-Wilmington-Robotic-EXoskeleton-Arm.html

Jones, O. (2017, December 27). The Radioulnar Joints. Retrieved October 1, 2018, from https://teachmeanatomy.info/upper-limb/joints/radioulnar-joints/

Jones, O. (2018a, July 16). The Shoulder Joint. Retrieved October 1, 2018, from https://teachmeanatomy.info/upper-limb/joints/shoulder/

Jones, O. (2018b, August 12). The Elbow Joint. Retrieved October 1, 2018, from http://teachmeanatomy.info/upper-limb/joints/elbow-joint/

Kazemi, V., & Sullivan, J. (2014). One millisecond face alignment with an ensemble of regression trees. *2014 IEEE Conference on Computer Vision and Pattern Recognition*. doi:10.1109/cvpr.2014.241

LaPlante, M. P., Harrington, C., & Kang, T. (2002, April). Estimating Paid and Unpaid Hours of Personal Assistance Services in Activities of Daily Living Provided to Adults Living at Home. Retrieved October 18, 2018, from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1430364/

Lobo-Prat, J., Kooren, P. N., Janssen, M. M., Keemink, A. Q., Veltink, P. H., Stienen, A. H., & Koopman, B. F. (2016, November). Implementation of EMG- and Force-Based Control Interfaces in Active Elbow Supports for Men With Duchenne Muscular Dystrophy: A Feasibility Study. Retrieved from https://ieeexplore-ieee-org.ezproxy.wpi.edu/stamp/stamp.jsp?tp=&arnumber=7410057&tag=1

Muscular Dystrophy Association. (2018, June 22). Duchenne Muscular Dystrophy (DMD). Retrieved September 28, 2018, from https://www.mda.org/disease/duchenne-muscular-dystrophy

National Cancer Institute. (n.d.). Anatomical Terminology. Retrieved September 17, 2018, from https://training.seer.cancer.gov/anatomy/body/terminology.html

National Institute of Neurological Disorders and Stroke, & National Institutes of Health. (2018, August 9). Amyotrophic Lateral Sclerosis (ALS) Fact Sheet. Retrieved September 28, 2018, from https://www.ninds.nih.gov/Disorders/Patient-Caregiver-Education/Fact-Sheets/Amyotrophic-Lateral-Sclerosis-ALS-Fact-Sheet

National Physical Laboratory. (2010, March 25). How many different types of force transducer are there? (FAQ - Force) : FAQs : Reference. Retrieved October 11, 2018, from http://www.npl.co.uk/reference/faqs/how-many-different-types-of-force-transducer-are-there-(faq-force)

National Stroke Association. (2015, July 08). Post-Stroke Conditions. Retrieved September 28, 2018, from http://www.stroke.org/we-can-help/survivors/stroke-recovery/post-stroke-conditions/physical

Noraxon. (2017). Surface EMG. Retrieved October 12, 2018, from https://www.noraxon.com/our-products/emg/

OYMotion Inc. (n.d.). Gesture Armband gForce 100 Manual V1.1. Retrieved October 12, 2018, from https://oymotion.github.io/assets/downloads/gForce100_manual_v1.1-eng.pdf

Perlman, H., & USGS. (n.d.). Water Density. Retrieved October 17, 2018, from https://water.usgs.gov/edu/density.html

Performance Health. (2018). Stable Slide Self-Feeding Support. Retrieved September 20, 2018, from https://www.performancehealth.com/stable-slide-feeding-support

Plagenhoef, S., Evans, F.G. and Abdelnour, T. (1983) Anatomical data for analyzing human motion. Research Quarterly for Exercise and Sport 54, 169-178. Retrieved October 17, 2018, from https://exrx.net/Kinesiology/Segments

Popovic, Marko B. "Feeding Systems, Assistive Robotic Arms, Robotic Nurses, Robotic Massage." Biomechatronics (2019): 419.

Raez, M., Hussain, M., & Mohd-Yasin, F. (2006). Techniques of EMG signal analysis: Detection, processing, classification and applications. Retrieved October 16, 2018, from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1455479/#B8

Reaz, M. B., Hussain, M. S., & Mohd-Yasin, F. (2006, March 23). Techniques of EMG signal analysis: Detection, processing, classification and applications. Retrieved October 14, 2018, from https://biologicalproceduresonline.biomedcentral.com/track/pdf/10.1251/bpo115

Safaee-Rad, R., Shwedyk, E., Quanbury, A. O., & Cooper, J. E. (1990, June). Normal functional range of motion of upper limb joints during performance of three feeding activities. Retrieved September 28, 2018, from https://www.ncbi.nlm.nih.gov/pubmed/2350221

Shahid, S., Goffin, J., & Chaves, C. (2018). Pronation and Supination. Retrieved October 1, 2018, from https://www.kenhub.com/en/library/anatomy/pronation-and-supination

The Internet Stroke Center. (n.d.). The Internet Stroke Center. Retrieved September 21, 2018, from http://www.strokecenter.org/patients/about-stroke/stroke-statistics/

Verily. (n.d.). Liftware Level™. Retrieved October 1, 2018, from https://www.liftware.com/level/p