Anatomically Accurate Motorized Shoulder Model

A Major Qualifying Project submitted to the Faculty of Worcester Polytechnic Institute in partial fulfillment of the requirements for the Degree of Bachelor of Engineering.

Submitted By:

Gray Rahm

Sean Merone

Authorship:

Gray Rahm

Sean Merone

Cameron Leffler

Date:

March 22nd, 2024

Project Advisor:

Professor Fiona Levey

Abstract

Despite its significance within the body, educational tools effectively modeling the intricacies of the shoulder joint are scarce. Seeking to address this void, our team continued the development of a life-sized model comprising the scapula, humerus, and a removable clavicle. Our model aims to more accurately replicate the nuanced scapulohumeral rhythm observed during abduction of the humerus, a process integral to understanding the shoulder joint. The scapulohumeral rhythm, orchestrating the coordinated movement between the scapula and humerus during various shoulder motions, particularly abduction, is paramount for maintaining shoulder joint stability and functionality. Our developed rig serves as an excellent foundation for future teams to supplement. A clear path forward is laid for any future teams to fully develop a valuable educational resource for students, healthcare professionals, and medical enthusiasts seeking a comprehensive understanding of the human shoulder.

Acknowledgements

Our team would like to thank several individuals for their efforts in aiding us in completing our project. We would like to thank Pradeep Radhakrishnan for providing us with a space where our team could thrive and construct the model. We would additionally like to thank Peter Hefti in providing us the tools we needed to build the final model. Finally, we'd like to thank our advisor, Fiona Levey, for her incredible guidance and support throughout this project.

Authorship

Our MQP project in creating an anatomically accurate motorized shoulder model was completed through the efforts of Gray Rahm, Cameron Leffler, and Sean Merone. Each member contributed towards the research behind the construction of the model and the materials necessary for the project. Construction of the model's cage, design, schematics, and circuitry was a team effort led by Gray Rahm. The code for the project was developed by Cameron Leffler. The final report submitted to the standards of the WPI MQP report guidelines was a team effort led by Sean Merone.

Cameron Leffler will continue to work on the project for the remainder of the academic year and will submit his MQP report at the conclusion of the academic year. He will be working on creating a 3D animation of the scapulohumeral rhythm using Blender.

Table of Contents

Table of Figures

Table of Tables

1.0 Introduction

The human shoulder stands as a marvel of anatomical complexity, functioning as a pivotal joint essential for a wide range of movements. Despite its significance, educational tools that effectively model the intricacies of the shoulder joint are scarce. In response to this gap, we have continued the mechanization of a life-sized model encompassing the scapula, humerus, and clavicle. This creation aims to replicate the nuanced motion observed during abduction, a process integral to understanding the scapulohumeral rhythm.

The scapulohumeral rhythm refers to the coordinated movement between the scapula (the shoulder blade) and the humerus (the upper-arm bone) during various shoulder motions, particularly abduction. This rhythm plays a crucial role in maintaining the stability and functionality of the shoulder joint. Our mechanization of the model seeks to provide a tangible and visually instructive representation of this intricate interplay, offering a valuable educational resource for students, healthcare professionals, and anyone interested in comprehending the complexities of the human shoulder.

By combining anatomical accuracy with a focus on motion dynamics, our model offers a hands-on approach to learning about the scapulohumeral rhythm. Through its open and spacious design, users can gain a tangible understanding of the spatial relationships and articulations involved in shoulder abduction. This educational tool not only serves as a valuable asset in academic settings but also contributes to a broader appreciation of human anatomy and biomechanics.

In the subsequent sections, we will delve into the key components of our contribution to the shoulder model's development through accurate relative motion, visual representation, exploring the anatomical principles it embodies, and elucidating the educational benefits it presents. From biomechanical insights to practical applications, this life-sized shoulder model is poised to enhance the understanding and appreciation of one of the body's most intricate and essential joints.

2.0 Background

2.1 Scapulohumeral Relationship

The movement of the shoulder consists of two primary bones, those being the the scapula and the humerus, with the clavicle factoring in largely as support and a connection to the sternum in the center of the ribcage. Understanding the relationship of movement between these two bones is key towards accurately modeling and displaying how the shoulder joint functions. The joints connecting these bones are equally as integral to this system, with the two most prominent in the system being the glenohumeral joint, which serves to connect the scapula and the humerus, and the acromioclavicular joint, which connects the scapula and the clavicle.

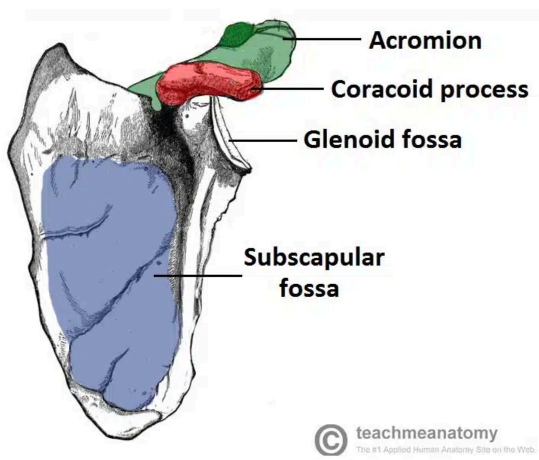

Figure 1: Depiction of the Scapula and its varying regions (Teach Me Anatomy, 2022)

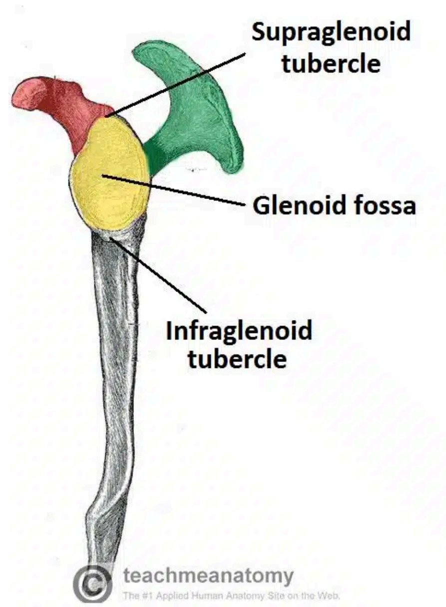

Figure 2: Depiction of the Scapula Ridge and its varying joints (Teach Me Anatomy, 2022)

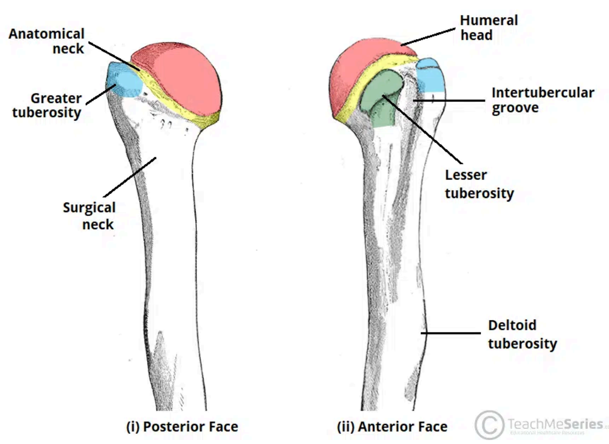

Figure 3: Depiction of the Humerus and its varying regions (Teach Me Anatomy, 2022)

Upon abduction of the humerus, the scapula transitions and rotates to accommodate the movement in several stages, however, where the humerus exists and moves on a two dimensional plane assuming there is no antepulsion or retropulsion, the scapula shifts and rotates in a three dimensional manner, with each stage of movement being met with a different facet of rotation. Nikita Igoshin's 2022 Independent Study Project (ISP) focused heavily on these different stages of movement. Igoshin's ISP based its findings on the work from "A Biomedical Analysis of Scapular Rotation During Arm Abduction in the Scapular Plane" written by Stephen D. Bagg, MD, MSc and William J, Forrest MD, MSc. with their results displayed below in Table 1.

Range	Average Arm Rotation (AA) : Scapular Rotation (SR)
0-20.8°	1:0
20.8-81.8°	4.29:1
81.8-139.1°	1.71:1
139.1-170°	4.49:1
20.8-170°	2.25:1

Table 1: Abduction Ratio (Average Arm Rotation:Scapular Rotation) (Bagg, 2016)

The results displayed indicate the ratio of movement between the humerus and the scapula, with the first range being especially of note as until the humerus meets a 20.8° angle with the horizontal axis, there is negligible movement in the scapula. As abduction of the humerus continues, transitional movement of the scapula and rotational movement of the scapula differentiate. The angle of the scapula was found to follow one of three patterns in its rotational movement with respect to the angle of the humerus through the analysis of a large number of subjects. The patterns are depicted below in figures 4, 5, and 6.

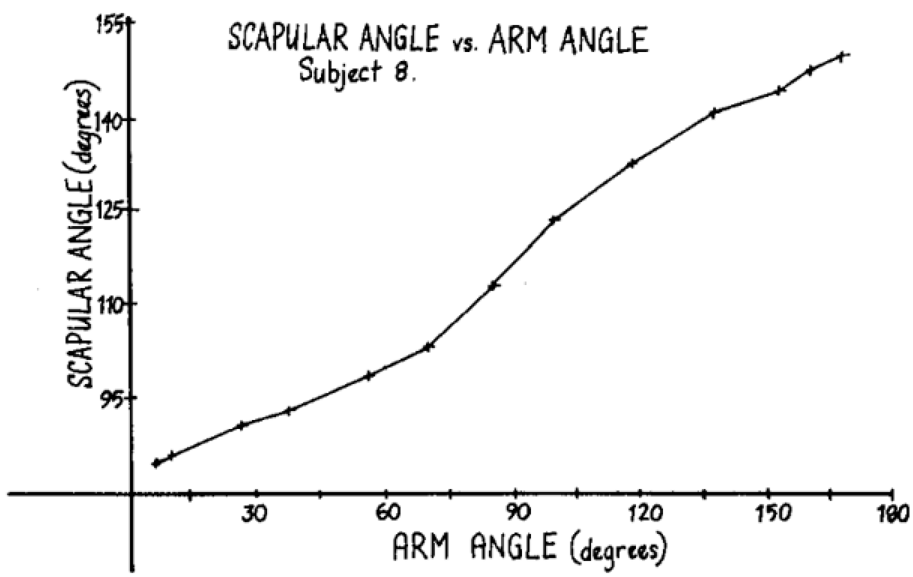

Figure 4: The type A pattern of scapulohumeral rhythm. (Bagg, 2016)

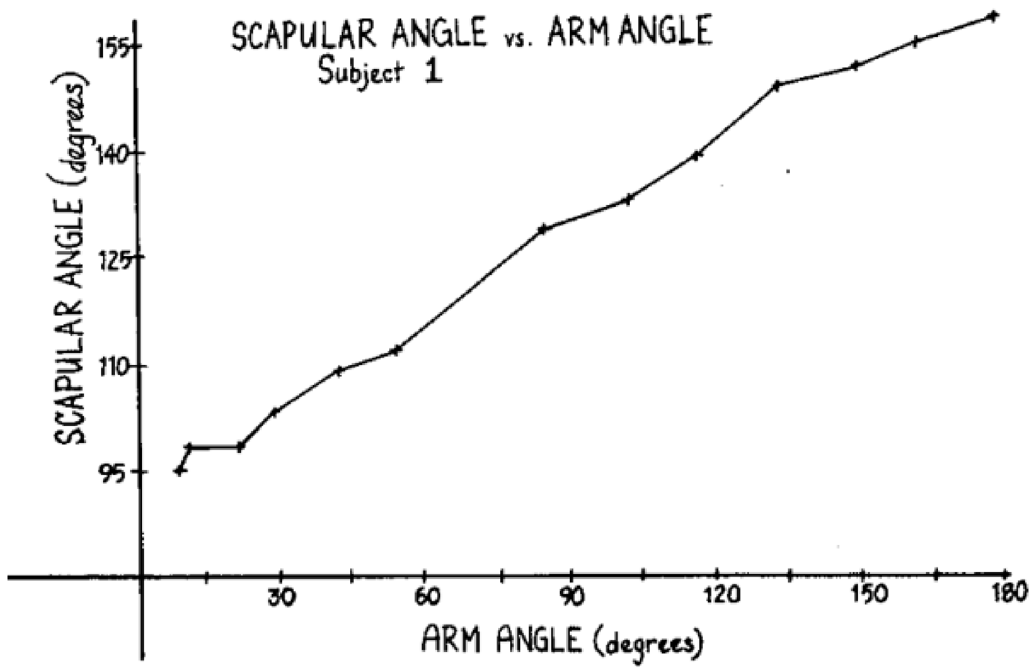

Figure 5: The type B pattern of scapulohumeral rhythm. (Bagg, 2016)

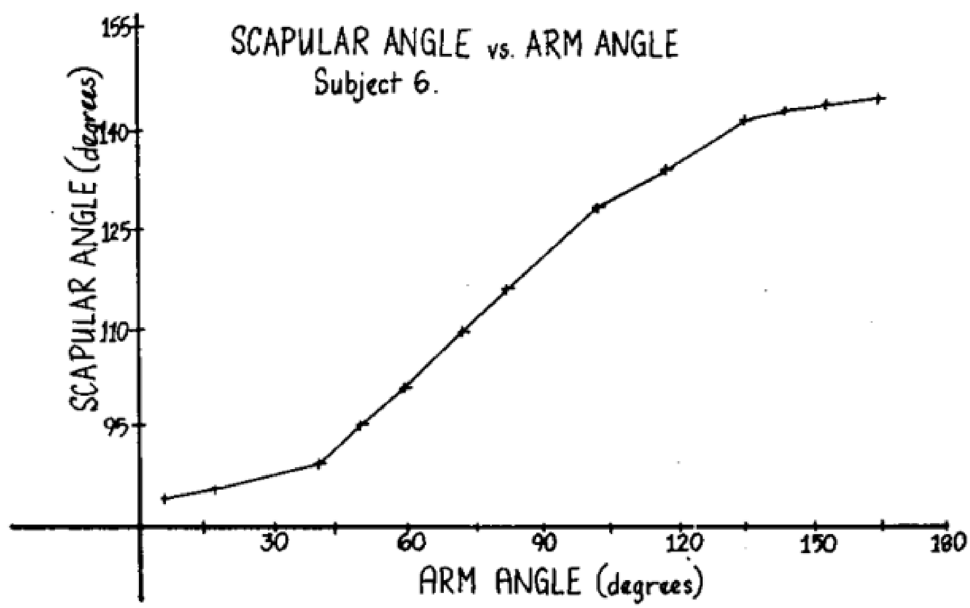

Figure 6: The type C pattern of scapulohumeral rhythm. (Bagg, 2016)

The beginning position of the scapula and the point at which the scapula begins to rotate in accordance with the movement of the arm differ greatly between the three patterns, with patterns A and C beginning at a starting angle of around 85 degrees to the vertical axis with pattern B beginning at 95 degrees to the vertical axis. The rate at which the scapula rotates and shifts, while not uniform between the patterns, is largely comparable at the later stages of rotation, with an arm angle of 60 to 80 degrees being the point where the three patterns begin to line up. These results indicate that while the scapulohumeral rhythm differs between individual subjects, it can be approximated and averaged to produce tangible and uniform results. Understanding this, the team opted to use the ratios found by Nikita Igoshin for the scapulohumeral rhythm as the foundation for the project.

2.2 Previous Work on the Model

This project is a continuation of two previous MQP efforts. Two prior teams similarly conducted their own work and tackled the project with different concepts. The 2021-2022 team was largely centered around the development of the bones and structure that the model would rest on and the construction of a rig to demonstrate scapulohumeral rhythm during abduction from 0 to 60 degrees. The group of students procured 3D models of the bones of the shoulder, the scapula, the humerus, and the clavicle as well as the rib cage; to which they added blocks for mounting. The team then 3D printed the models out of PLA (Polylactic acid; a common 3D printing filament). They then mounted the rib cage on to plywood and created a simple rig focused on replicating abductional movement of the shoulder through nylon threads and motors, with partial success.

The 2022-2023 team continued construction of the model, this time focusing on the addition of ligaments and altering motor attachment points to properly replicate the components of an anatomical shoulder. Their team intended to select materials that accurately replicated the function of shoulder tendons, ligaments, and muscles, utilizing a number of different materials to replicate the movement of the humerus.

Our team similarly continued the construction of a rig to replicate movements, utilizing the models produced by the 2021-2022 team and the ideas of an extraneous point of connection from the 2022-2023 team. This was the foundation that our team worked from, building and improving upon their previous designs and accounting for facets and complications the previous teams did not have the time to consider.

2.2.1 2021-2022 MQP: Anatomically Accurate Motorized Shoulder Model with Scapula Movement

The 2021-2022 team gave a clear indication of their design process throughout their project. The shoulder joint, including the scapula, humerus, clavicle, and rib cage would be mounted on a supporting board with motors providing the force needed to conduct movement of the humerus and scapula. Their primary designs were as follows: a secondary rod design and a pulley system trailing the length of the humerus.

1	Base Plate
2	Primary Rod
3	Secondary Rod
4	Humerus
5	Clavicle
6	Scapula
7	Glenohumeral Joint
8	Sternoclavicular Joint
9	Scapulothoracic Joint

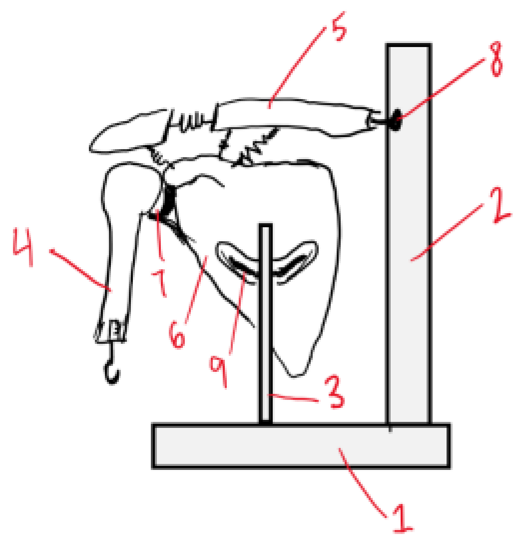

Figure 7: Sketch of the Secondary Rod Preliminary Design (Deane et al., 2022)

The above figure displays the shoulder joint in its entirety alongside the secondary rod system, with each part labeled within the table on the left. The Secondary Rod was intended to function as another source of support as well as a guiding hand for scapular rotation. It would achieve these goals by connecting to the scapula through a ball and joint socket in a slot displayed below in figure 8 and figure 9. The slot is curved and hollow so as to allow the scapula to rotate during translation throughout the humeral abduction process. During the rotation and translation of the scapula, the ball is free to slide through the hollow interior of the slot from the center to the edges to accurately mimic the movement of the scapula within the shoulder joint. Similarly during humeral flexion, the ball is free to return to its initial position.

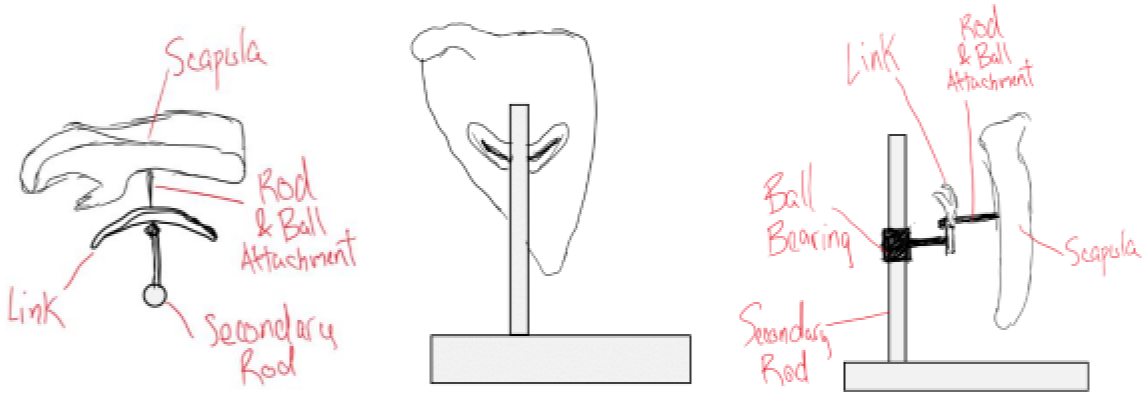

Figure 8: Sketches of top, Front and side view of the Scapulothoracic Joint utilized within the Secondary Rod Design (Deane et al., 2022)

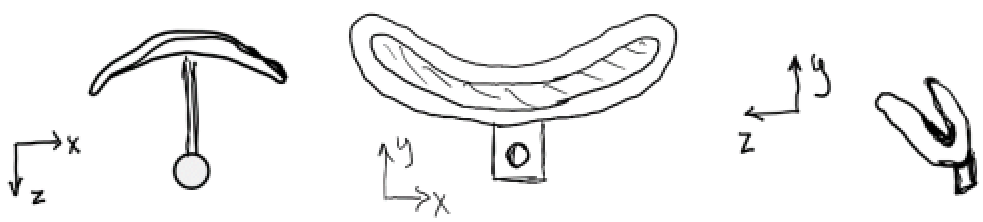

Figure 9: Sketches of top, front, and side view of the unique slot feature representing the Scapulothoracic joint (Deane et al., 2022)

The secondary rod idea was ultimately scrapped by the 2021-2022 team due to the additional parts that would be required to provide the force necessary to replicate the motion of the scapula and humerus. These additional parts physically intersected the secondary rod, preventing the motion from completion. However, this idea of another point of connection is something our team sought to continue with our project. The 2021-2022 team's second primary design was the development of a pulley system that trailed the length of the humerus and scapula as described in section 2.2.1.3.

2.2.1.1 3D Printed Bones

The 2021-2022 team purchased files of the bones that they then 3D printed (Figure 10 and Figure 11). In order to fasten the ribcage to the plywood, a 0.75in x 0.75in rectangular segment was added to each rib at varying lengths to meet at one plane. These fastenings are shown in Figure 12 and Figure 13. The points of these connections are displayed below in Table 2 and Figure 14.

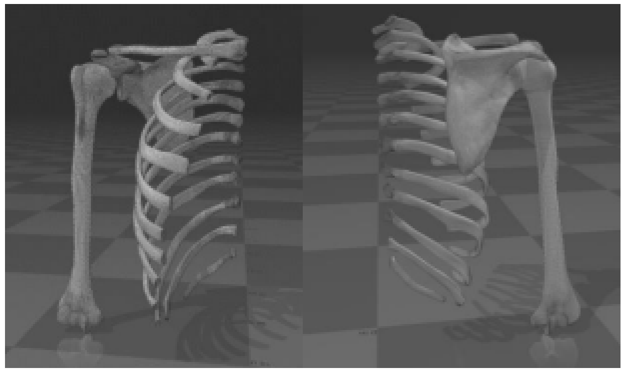

Figure 10: Front and back view of model true to scale (Biomedical Modeling Inc.)

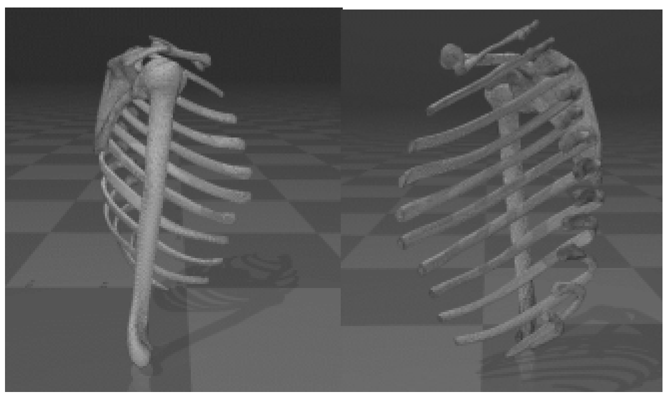

Figure 11: Left and right view of the model true to scale (Biomedical Modeling Inc.)

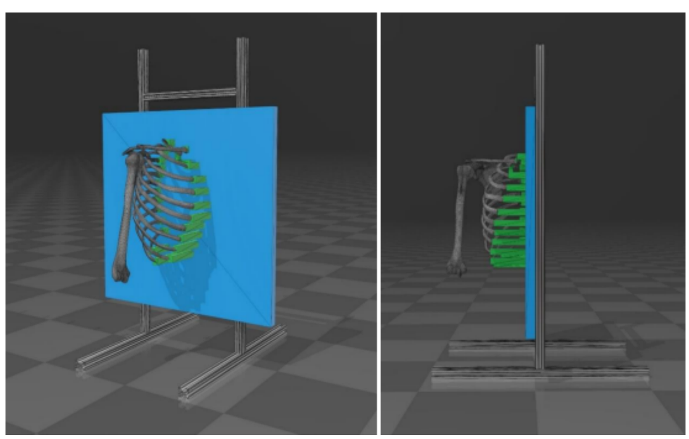

Figure 12: Front and Side Views of Full Shoulder Rig Assembly without Soft Tissue (Deane et al., 2022)

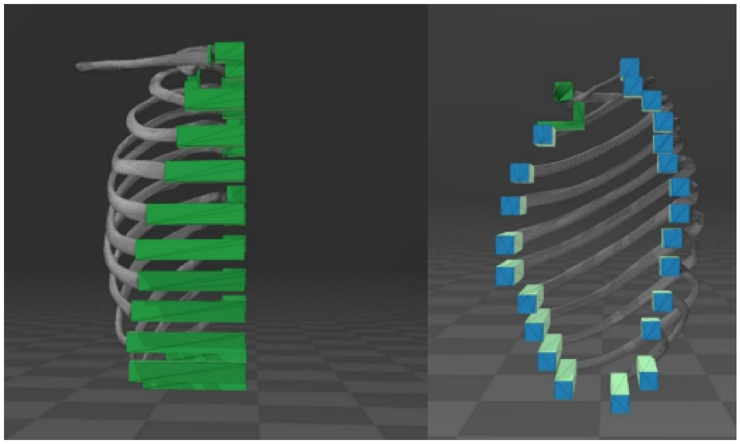

Figure 13: Front and Right-side View of Rib Connections (Deane et al., 2022)

Y	Z	Y	Z
38.8411	277.6354	160.753	105.5732
15.2039	238.4864	168.1988	142.0047
5.6789	204.5192	170.3961	171.297
0	164.5711	177.8404	196.7875
0	132.4797	177.8404	223.5691
22.1709	103.6558	165.7629	248.0114
29.3163	74.9135	160.8711	268.1104
42.4981	44.6453	160.8711	290.4167
59.4609	14.6593	149.1488	312.0448
117.1742	0	130.0988	325.9589
145.7492	24.1843	126.6992	345.0089
149.1488	76.648	59.4608	322.4804

Table 2: Rib Connection Points (Deane et al., 2022)

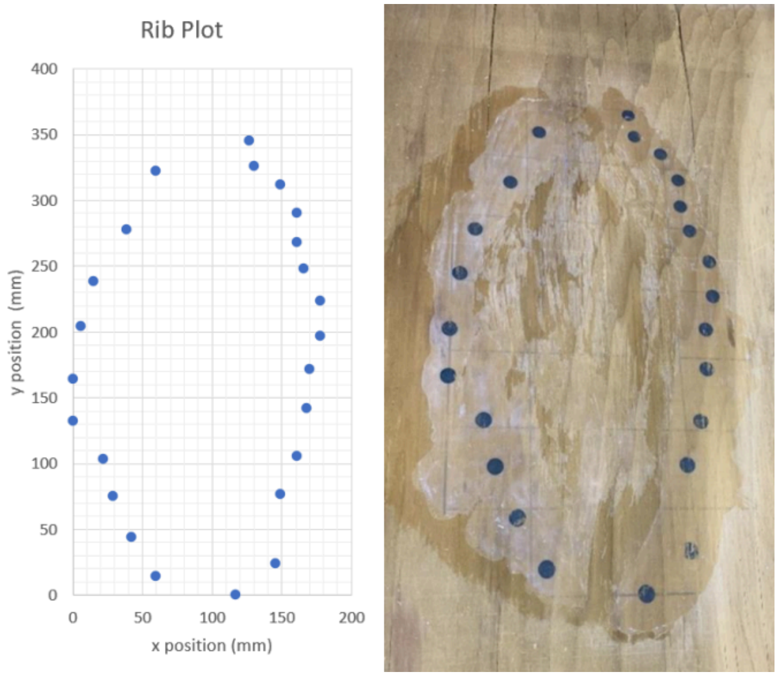

Figure 14: Corresponding digital plot (left) that was transferred to the plywood (right) for accurate bone positioning (Deane et al., 2022)

2.2.1.2 Mounting the Bones and Motors

The bones were mounted and secured to 3D printed blocks which were then fastened to the points on the plywood above as designated in section 2.2.1.1, the clavicle and motors were mounted differently due to their differing functions. The clavicle was mounted to a plywood block through a ball and socket joint that allowed for limited rotational movement. The motors and circuits of the 2021-2022 team were mounted to the rear side of the plywood to not obstruct the movements or visual of the shoulder during its abduction, however, doing so introduced a large amount of friction between the plywood board and the nylon monofilament fishing line utilized in their pulley system as they travel over the top of the plywood. The mounts utilized are displayed below in Figures 15 through 19.

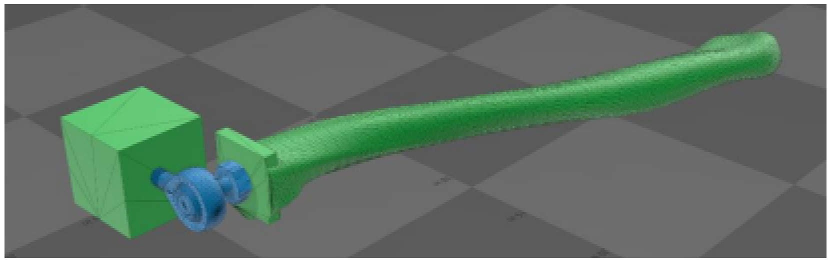

Figure 15: Clavicle and plywood attachments (Deane et al., 2022)

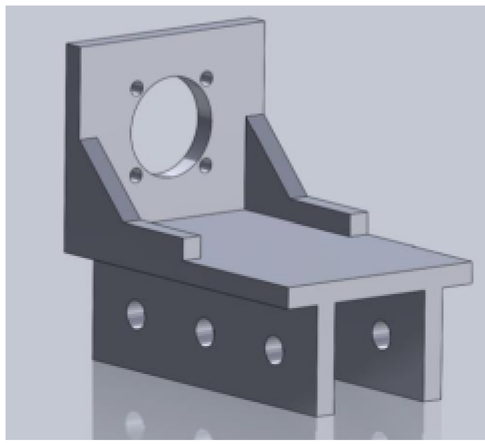

Figure 16: Second stationary motor mount designed to secure a single Planetary Gearbox Nema 17 stepper motor to the top of the t-slot beam (Deane et al., 2022)

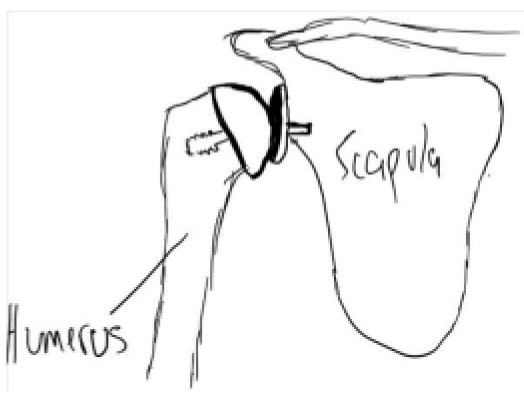

Figure 17: Sketch of glenohumeral joint design (Deane et al., 2022)

Figure 18: Sketch of glenohumeral joint design (Deane et al., 2022)

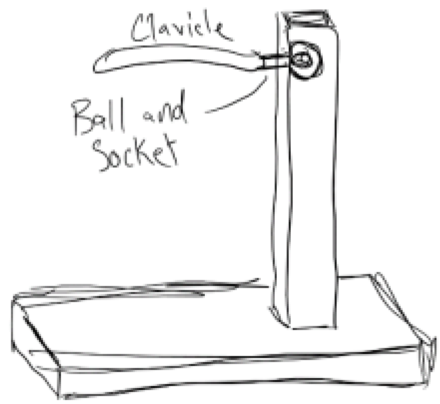

Figure 19: Isolated sketch of sternoclavicular joint design (Deane et al., 2022)

2.2.1.3 2021-2022 Team's Final Product and Data Collection

The 2021-2022 team unfortunately did not have the time to flesh out a fully functioning system to induce movement in the arm, as their project focused much more heavily on creating the models of the bones and bringing them to fruition. That being said, their motorized design had numerous aspects that our team built upon. Their design utilized a motor system that trailed the length of the arm and was attached to the other side of the wooden board used to hold and support the ribcage. While the use of a motor system is the most cost-effective option available, the angle the force was delivered made vertical movement of the arm incredibly intensive, as a vast majority of the force was pulling the arm into the shoulder socket rather than upwards to display abduction.

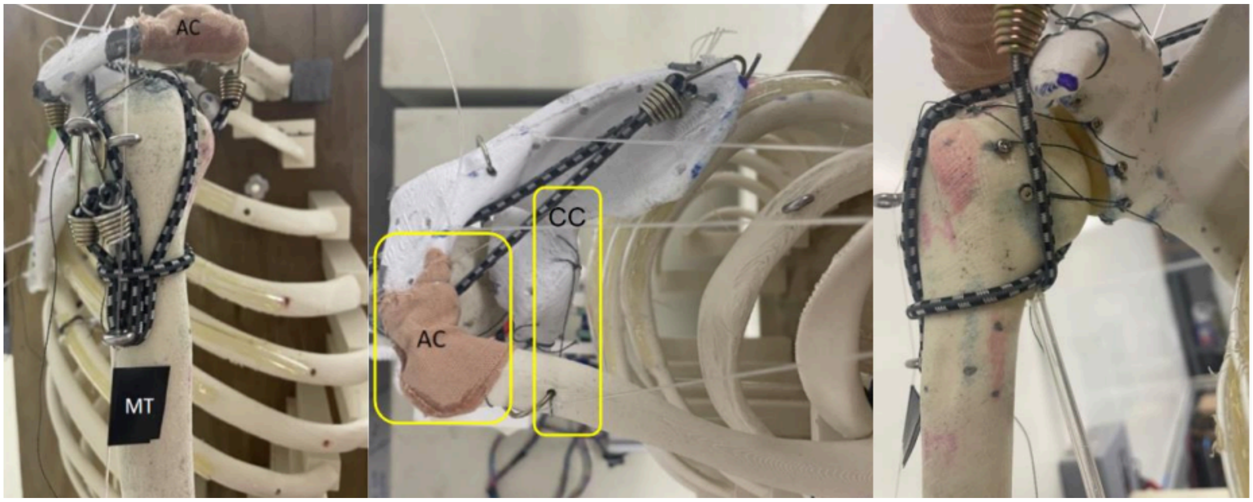

Figure 20: Side, top, and front view of shoulder model (Deane et al., 2022)

2.2.2 2022-2023 MQP: Realistic Shoulder Model with Soft Tissue Attachments

The 2022-2023 team focused far more heavily on the addition of materials that would replicate biological functions within the shoulder. The team 3D printed several slabs of polymer materials and attached them to the areas of important tendons and muscle groups to simulate restrictions the shoulder would experience. However, these materials had difficulty remaining attached in the intended manner and often blocked visibility of the more important movements of the model. The inefficiency and other problems introduced by these design options were heavily accounted for when forming our own designs for the rig.

3.0 Methodology

3.1 Design Goals and Constraints

Our team focused our project around numerous different smaller scale goals, each of which were focused on completing our overarching goal of improving upon the design of the two previous MQP team's models to more accurately reflect shoulder movement through abduction of the humerus. To achieve this goal, our team had to operate under a number of constraints, primarily seeking to minimize the cost of the project so as to make the final product as accessible and affordable as possible. Other constraints such as accessible materials and machines to shape these materials would additionally limit the abilities of the team to develop the final product. Further constraints including the size and the transportability of the rig similarly limited potential designs. Our overarching objective would be completed through our other goals which were similar to the previous MQP team's goals in developing an efficient rig that would provide adequate force to lift the humerus whilst rotating and transitioning the scapula appropriately in response to the humerus's position. The rig should additionally be capable of completing the motion displayed in figure 21 an indefinite number of times without requiring frequent replacement of parts.

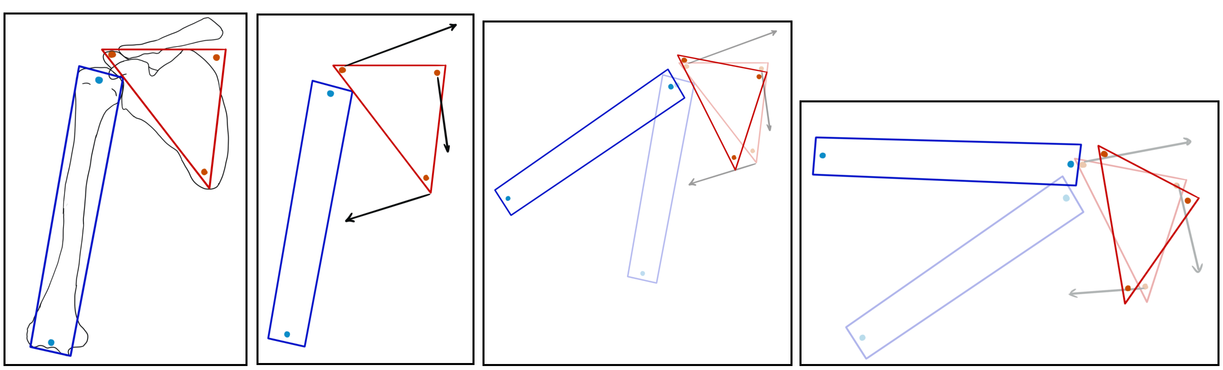

Figure 21: Humeral and Scapular Motion

Our first goal was to calculate and replicate accurate motion of the scapula. The scapula shifts and rotates on a three dimensional plane and has a constantly shifting center of rotation alongside a complex geometric shape that can make accurately replicating its movement difficult. Our efforts to reflect this movement are detailed in section 3.3 Scapular Rotation. Our second goal was to achieve the full desired range of motion, raising the humerus from 0° to 120° with the vertical axis. Our final goal to improve the previous design of the model was to accurately replicate relative motion between the scapula and humerus. While the movements of the scapula and humerus aren't causing relative motion between one another in the model like they would in the body, the motors in our rig would accurately replicate the relative movement of the bones associated with the scapulohumeral rhythm described in our design concepts and rig designs in section 3.2.

3.2 Rig Design Concepts

To achieve the scapulohumeral rhythm and motion desired, our team created two preliminary design concepts with a number of smaller variations.

3.2.1 Channel Design Concept

One preliminary design concept our team focused on was the idea of implementing a channel to follow the length of the humerus, mimicking the design of the previous MQP team's final implemented design. Where the previous teams had the wires and supporting materials on the outside surface of the humerus, this design would involve drilling a channel into and along the length of the humerus and a portion of the scapula, where the wires would be run through and over the collar into the plywood board supporting the model where the motors would pull from. This design is displayed below in figure 22.

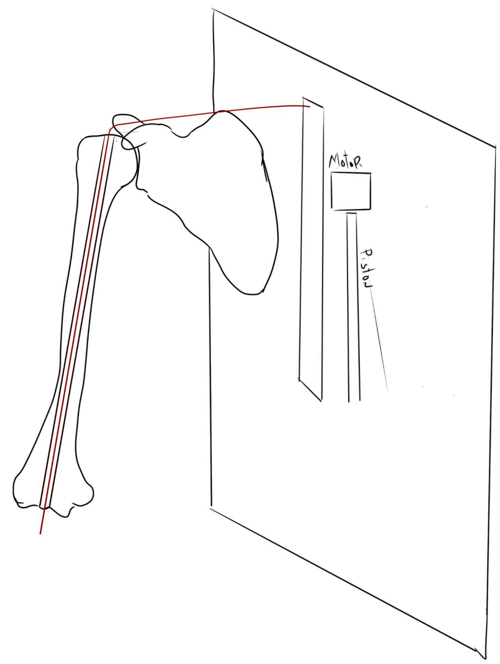

Figure 22: Channel Design Concept

The channel design allows for a much cleaner final model without the clutter of the previous team's approach in attaching motors, wires, and other materials to the exterior of the humerus and allows for the most visibility of the bones as they complete their motions. However, the channel design invites a number of complications including the question of force vector angles and friction of the internal wires as they travel through the bones and their connections.

3.2.2 Cage Design Concept

Our other preliminary design focused on creating a structure surrounding the bones to avoid the complications of the motor weights on the bones and suboptimal force vector angles. We referred to this structure as the cage. The bones of the ribcage and shoulder would be surrounded by a cage as shown in Figure 23 with the dimensions 0.5 meters x 0.8 meters x 0.8 meters, with the width of the cage being the shortest dimension of the cage.

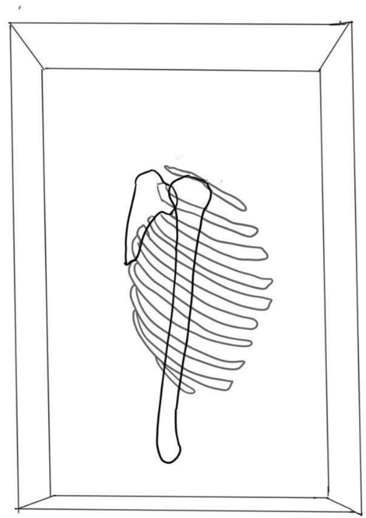

Figure 23: Model in the cage

Along the faces of the cage, motors would be placed and supported by beams with wire connections to the humerus and scapula to pull on the bones with adequate forces and angles to accurately replicate shoulder movement and scapulohumeral rhythm. The original placements of the motors were at the vertices of the cage where they would pull a final motor along the open face of the cage to guarantee the optimal angles for force vectors, however, this would restrict visibility and introduce difficulties in reinforcing the motor under its own force as well as simply being inefficient. Other placements were considered such as a moving track along the face of the cage, however, the final placements were decided to be constrained to one position and unmoving throughout the shoulder movements. These original motor placements and design concepts are shown below in figure 24.

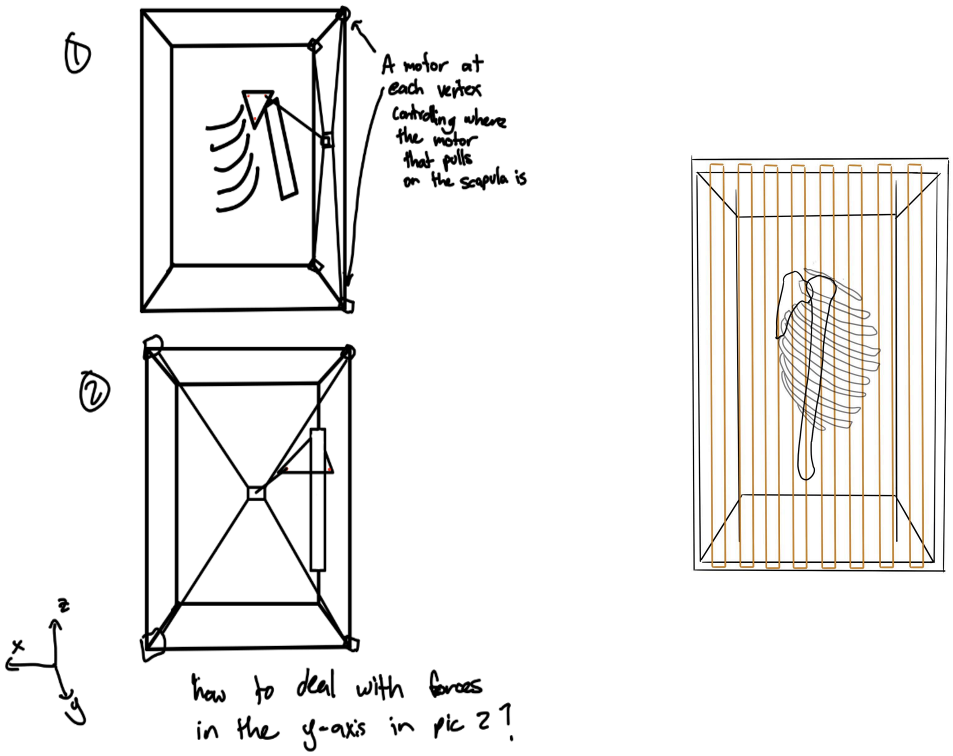

Figure 24: Vertice motor design (left) and track method design (right)

The exact placements of these motors are displayed in Figure 25 below. The materials utilized for the wires and the models of motors are discussed in sections 3.5.2 and 3.5.1 respectively. The motors were mounted to the cage through the use of 3D printed motor mounts left by the previous team. Additional mounts were printed using the same file the previous team used.

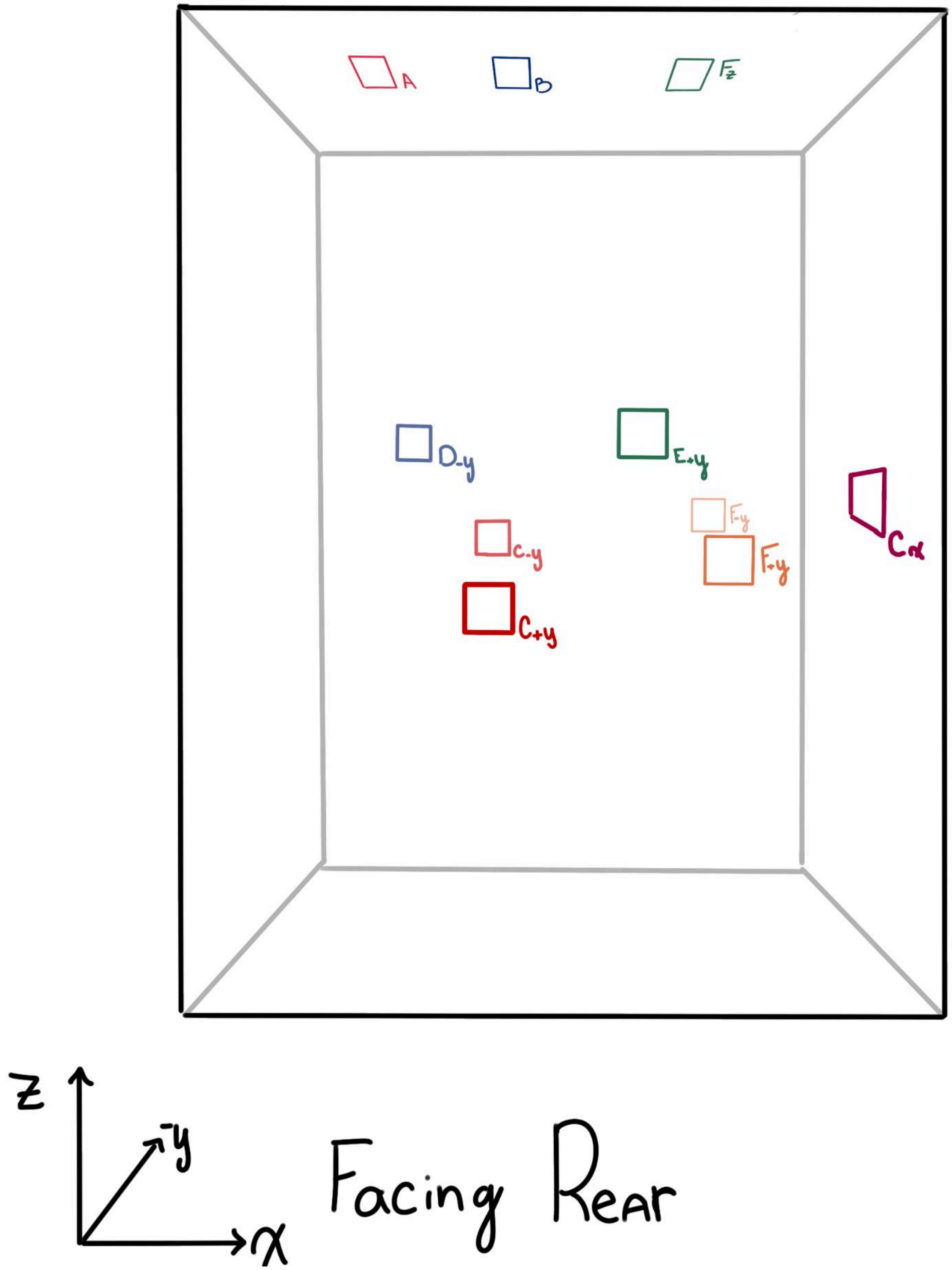

Figure 25: Motor placements along the faces of the cage

The primary benefits of the cage design were the visibility of the model and its movements it provided due to the lack of additional items cluttering up the humerus and scapula, which additionally made it much more efficient to design methods to move the humerus and scapula since factors such as the weight of the motors did not have to be considered when focusing on the movement of the bones.

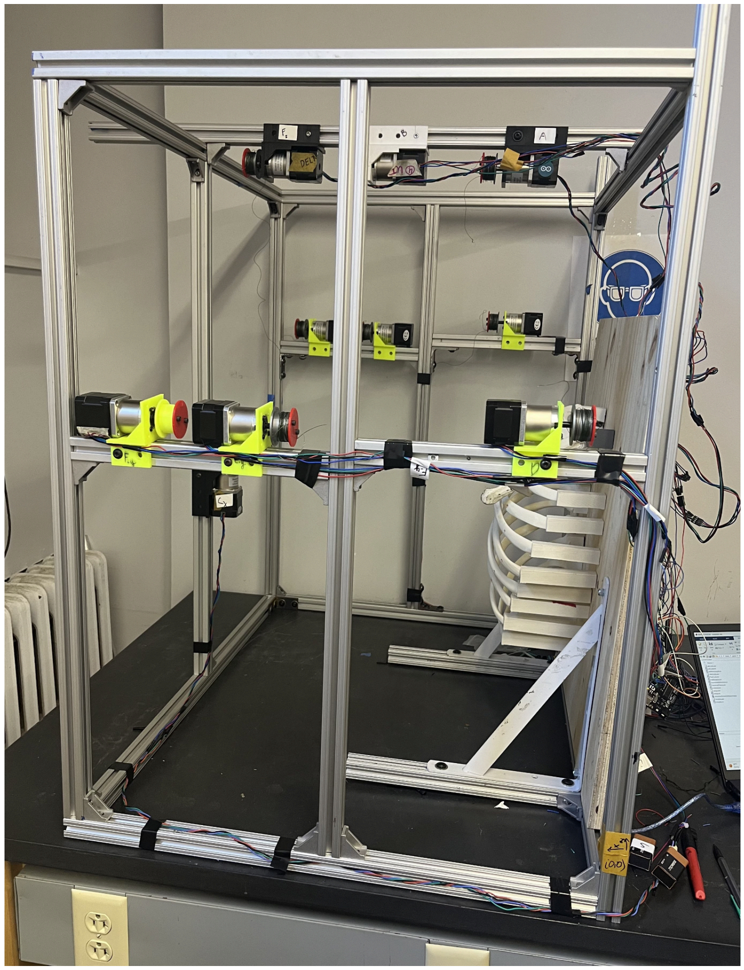

Figure 26: Constructed and Realized Cage

3.3 Scapular Rotation

Calculating how the scapula will rotate while the humerus abducts and flexes was vital to this project. The scapula's unorthodox shape and uneven distribution of mass caused the process to be complex, with the numerous directions of movement and constant uneven shifts of the scapula discussed in section 2.1 further complicating the process.

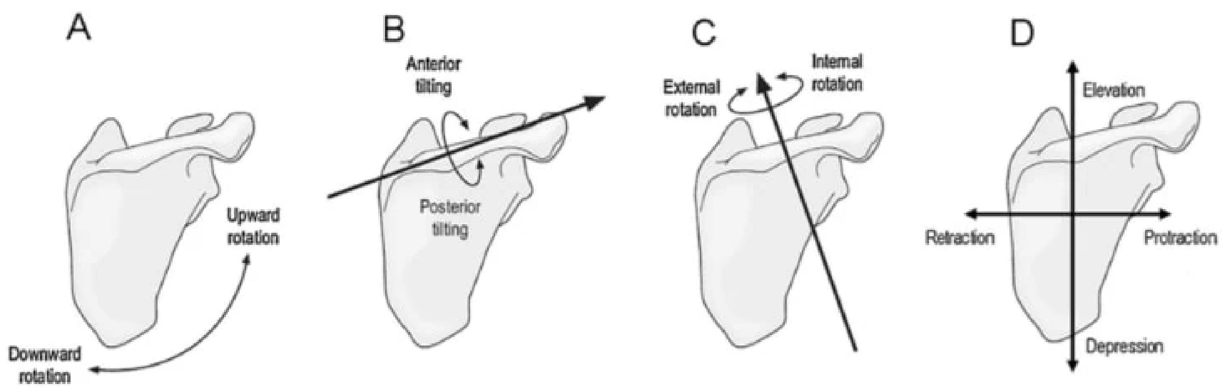

Figure 27: Potential movements of the Scapula (Physiotutors, 2023)

The position of the scapula was found by calculating where all five motor attachment points, A, B, C, D, and E, will be relative to the initial position of motor attachment point A. The attachment points are displayed below in figure 28. The amount of scapular rotation in each direction–upward rotation, anterior tilt, and external rotation–in 30° increments of humeral rotation were taken from tables 3 and 4. The team then assumed that rotation during each 30° increment was linear in its rotation, with the positions indicated on the scapula transitioning in a consistent fashion within the 30° periods.

Difference amount of change in scapula angle during shoulder abduction

Humeral abdction angle	Upward rotation angle (°)[a]			Anterior tilt angle (°)[b]			External rotation angle (°)[c]		
	Control	Thorax flexion	Thorax extention	Control	Thorax flexion	Thorax extention	Control	Thorax flexion	Thorax extention
0°–30°	3.39 ± 4.23	1.60 ± 3.23	2.05 ± 5.38	−6.04 ± 2.55	−4.95 ± 3.11	−5.55 ± 3.28	1.29 ± 1.65	0.93 ± 1.50	1.19 ± 1.72
30°–60°	26.34 ± 11.94	17.89 ± 8.98	22.38 ± 14.78	−22.28 ± 7.55	−24.13 ± 7.66	−19.31 ± 7.57	5.93 ± 2.90	6.45 ± 3.66	4.51 ± 3.63
60°–90°	41.25 ± 9.22	34.09 ± 9.32	35.20 ± 15.45	−31.45 ± 11.47	−39.25 ± 12.99[d]	−26.47 ± 10.80	8.92 ± 4.52	8.92 ± 4.52[d]	6.09 ± 5.15
90°–120°	50.85 ± 11.20	45.80 ± 10.60	42.58 ± 16.34	33.49 ± 16.02	−46.24 ± 17.61[d]	−26.97 ± 14.55	10.71 ± 6.08	15.69 ± 4.95[d]	6.46 ± 5.94

Table 3: Total rotation of the scapula in all three directions during 30° increments of abduction under "control" (Yabata, 2022)

Difference amount of change in scapula angle during shoulder flexion

Humeral flexion angle	Upward rotation angle (°)[a]			Anterior tilt angle (°)[b]			External rotation angle (°)[c]		
	Control	Thorax flexion	Thorax extention	Control	Thorax flexion	Thorax extention	Control	Thorax flexion	Thorax extention
0°–30°	1.55 ± 2.52	2.29 ± 2.55	0.83 ± 3.15	1.82 ± 2.38	1.91 ± 3.45	2.75 ± 2.68	−1.13 ± 1.52	−1.00 ± 1.76	−1.38 ± 1.07
30°–60°	9.14 ± 5.03	9.14 ± 5.33	7.91 ± 6.98	0.08 ± 6.01	0.09 ± 8.65	1.01 ± 6.38	−1.93 ± 1.76	−1.75 ± −3.09	−2.89 ± 2.97
60°–90°	15.03 ± 5.60	14.50 ± 5.62	14.85 ± 8.01	−3.80 ± 9.15	−4.65 ± 12.99	−2.71 ± 9.61	−1.95 ± 3.69	−1.95 ± 3.69	−3.71 ± 4.27
90°–120°	26.35 ± 7.02	24.84 ± 5.99	23.09 ± 8.62	−11.16 ± 13.45	−14.99 ± 16.86	−6.74 ± 13.52	0.22 ± 4.49	3.61 ± 5.64[d]	−3.47 ± 5.88

Table 4: Total rotation of the scapula in all three directions during 30° increments of flexion under "control" (Yabata, 2022)

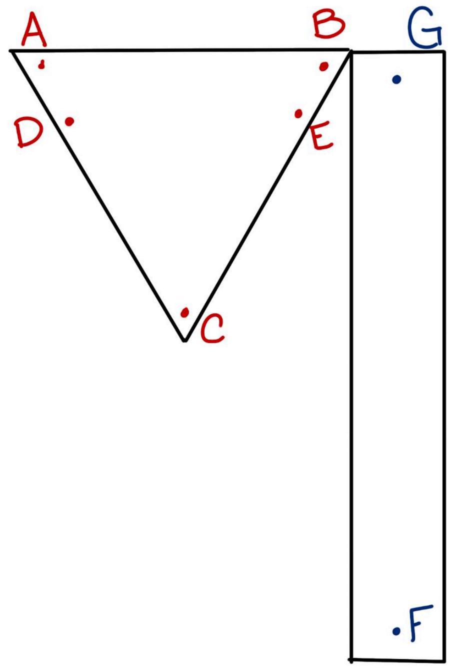

Figure 28: A simplified view of each of the motor attachment points

The team then defined two axes of rotation, one for upward rotation and one for anterior tilt, IJ and IK respectively. These axes of rotation were formed from the vector between two points on the scapula as displayed in figure 29. External rotation of the scapula is centered around a point called the instantaneous center of rotation, or the ICR.

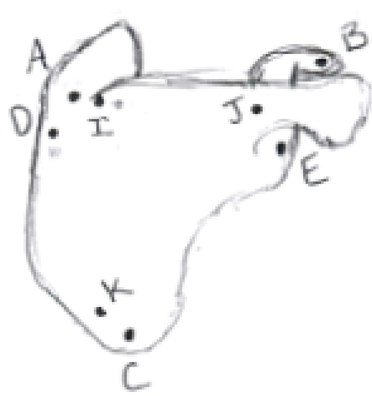

Figure 29: Key Points of Mass Distribution within the Scapula

The team then applied the Rodrigues' Rotation Formula, a formula for rotating a vector around a normal vector being used as an axis of rotation, for each direction of rotation in increments of 0.1° of humeral abduction or flexion up to 120°, and averaged the three outputs to find the new position of each motor attachment point. This strategy was also applied to points I, J, and K, so that the axes of rotation stayed within the scapula. An example of the Rodrigues' Rotation Formula is shown below. In this case, vector v is being rotated around the normal vector n by theta degrees to create vector v prime. The equation is the form of the Rodrigues' Rotation Formula the team used in their math.

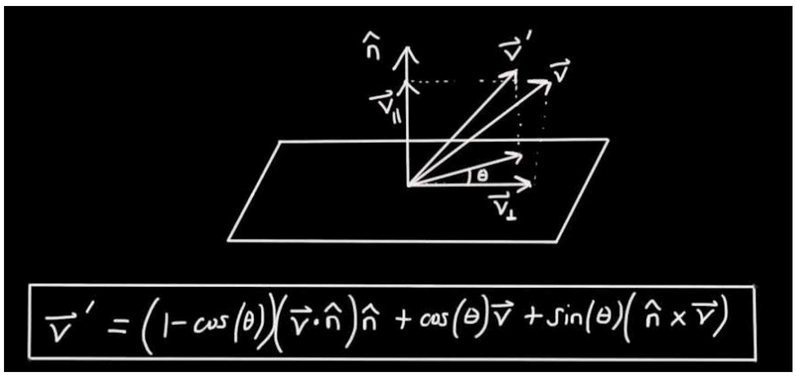

Figure 30: An example of the Rodrigues' Rotation Formula (YouTube channel Mathoma)

3.4 Humeral Abduction

Scapulohumeral rhythm was modeled for both humeral abduction and flexion. Humeral movement was confined to a two dimensional plane in its movements while it is moving in each direction, with additional safeguards in attached motors ensuring the humerus does not stray from its path of movement. The force required to lift the humerus was calculated through trigonometry utilizing a number of variables displayed in the nomenclature table below.

Variable	**Definition (unit)**
L_H	Length of the humerus (meters)
W_H	Mass of the humerus (kilograms)
α_i	Initial angle between the humerus and the vertical axis (°)
Θ_i	Initial angle between the humerus and the thread (°)

Θ’	Angle between the humerus and the thread post humeral abduction (°)
L_{Ti}	Initial thread length (meters)
L_T’	Thread length following humeral abduction (meters)
Δα	Desired change in angle alpha (°)
t	Time (seconds)
L_{HM}	Distance between the top of the humerus and the motor (meters)
β_i	Initial angle between the motor wire and the humerus (°)
β’	Angle β after abduction of the humerus (°)
d_{Hbot}	Distance traveled by the bottom of the humerus (meters)
v_{Hbot}	Velocity of the bottom of the humerus (meters/second)
a_{Hbot}	Acceleration of the bottom of the humerus (meters/second^2)
F_{gopp}	The force of gravity opposite to the direction of movement (Newtons)
F_{Net}	Net force acting on the humerus (Newtons)
$F_{Required}$	Required force to lift the humerus (Newtons)

Table 5: Nomenclature Table for Humeral Abduction

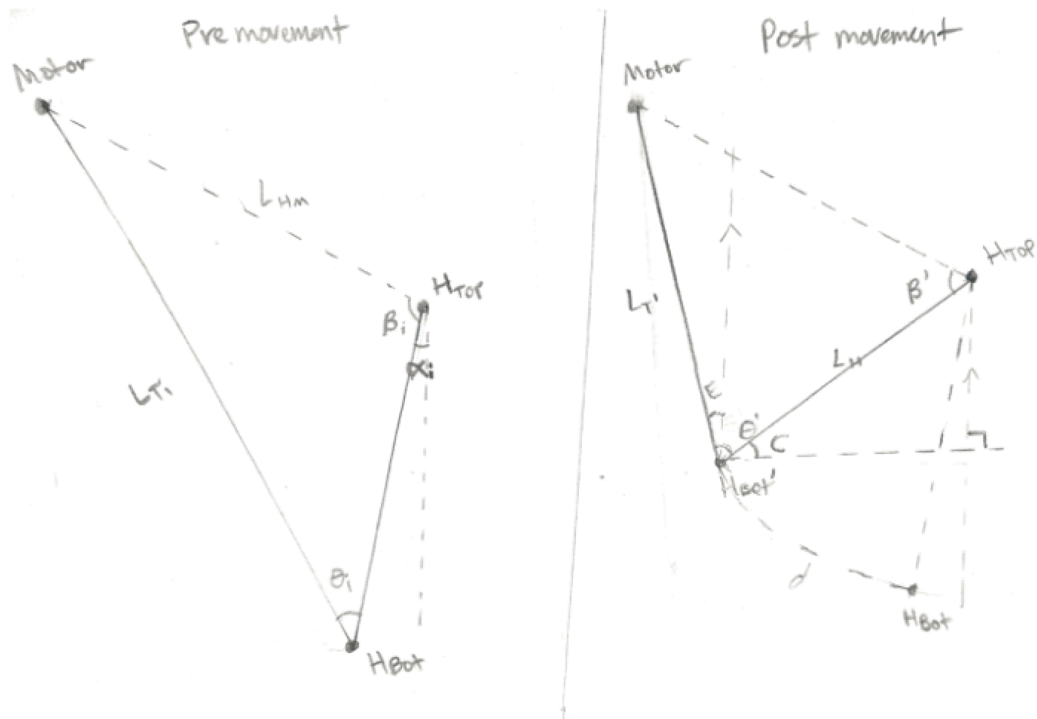

Figure 31: All Angles and Lengths during Humeral Abduction

The steps to solve for the force necessary to lift the humerus and the ideal position of the motor are displayed below in table 6.

Step Description	Equation
Beginning with the length of the humerus, the value for L_{HM} can be found using the law of cosines:	$L_{HM} = \sqrt{(L_H^2 + L_{Ti}^2 - 2L_H L_{Ti}\cos\Theta_i)}$
Using other given variables, L_{Ti}, L_H, and Θ_i the team can similarly solve for β_i	$(\sin\beta_i)/L_{Ti} = (\sin\Theta_i)/L_H$
With the initial value for the angle found, the team can then find the angle after the abduction of the humerus, β’:	$\beta' = \beta_i - \Delta\alpha$
With these variables defined, the desired thread length following humeral abduction can be found:	$L_T' = \sqrt{(L_H^2 + L_{HM}^2 - 2L_H L_{HM}\cos\beta')}$

With this final thread length determined, other values can be solved to determine the other kinematics needed to solve for the force required to lift the humerus.	$\Theta' = \sin^{-1}(L_{HM}\sin\beta'/L_T')$
With the new angle found, the team can then solve for the distance, velocity, and acceleration of the humerus.	$d_{Hbot} = \Delta\alpha\pi L_H/180$ $v_{Hbot} = d_{Hbot}/t$ $v_c = 2v_{Hbot}$ $a_{Hbot} = v_c/t$
New angles, E and C, must first be defined to relate F_g and solve for the force:	$C = 90 - (\alpha_i + \Delta\alpha)$ $E = (\Theta' + C) - 90$
Now with all values known, the force can be solved for using Newton's Second Law:	$F_g = 9.8W_H$ $F_{gopp} = F_g\cos E$ $F_{net} = W_H a_{Hbot}$ $F_{required} = F_{net} + F_{gopp}$

Table 6: Solving for the Necessary Force and the Position of the Motor

To track the position of the humerus, the team calculated where motor attachment point F will be relative to the initial position of motor attachment point A. This was done by setting the center of rotation at motor attachment point B and applying the Rodrigues' Rotation Formula and calculating the positions of motor attachment point F in increments of 0.1° of abduction and flexion up to 120°. In order to account for the movement of point B during this rotation, the team used geometry to "drag" the attachment point to its proper final position.

Givens:

All values are in meters and degrees. All distances are measured from the origin which is set at the bottom left corner of the cage as seen in figure 40.

Initial Wire Attachment Points:	Initial Axes Points:	Motor Placement:
A = [0.073;0.507;0.445];	I = [0.074; 0.508; 0.434];	motorA = [-0.045;.534;0.825];
B = [0.203; 0.509; 0.458];	J = [0.2; 0.508; 0.434];	motorB = [0.18;0.534;0.825];

C = [0.083;0.525;0.312];
D = [0.154;0.522;0.383];
E = [0.077;0.524;0.388];
F = [0.11;0.035;-0.3696];

K = [0.074; 0.508; 0.313];

motorCPosY = [0.44;0.852;0.54];
motorD = [0.185;0;0.5];
motorE = [0.19;0.852;0.366];
motorFZ = [0.405;0.534;0.825];
motorFPosY = [0.57;0.852;0.54];
motorFNegY = [0.505;0;0.48];

The variation in angle of the bones was based on a study (Yabata, 2022), where the angles are measured in cartesian coordinates relative to an origin.

Angle of Abduction	Total Upward Rotation	Total Anterior Tilt	Total External Rotation
0-30°	-3.39°	6.04°	1.29°
30-60°	-22.95°	16.24°	4.46°
60-90°	-14.91°	9.17°	2.99°
90-120°	-9.60°	2.04°	1.79°
Angle of Flexion			
0-30°	-1.55°	-1.82°	-1.13°
30-60°	-7.59°	1.74°	0.80°
60-90°	-5.89°	3.88°	-0.02°
90-120°	-11.32°	7.36°	2.17°

Table 7: Rotation of the Humerus during Abduction and Flexion (Yabata, 2022)

Calculations:

1. How to rotate the scapula. The example provided will be point E moving as the humerus abducts from 0° to 10°. Each step calculated by the code is 0.1°; the 10° increment is used for illustrative purposes.
   1.1. Find how much E externally rotates during this interval using the table and step size.
      1.1.1. 1.29°/(30/10) = 0.43°

1.2. Find the normal axis of rotation, $\widehat{IK}$ in this case

1.2.1. Move the origin to I

1.2.2. $I = [0.074;\ 0.508;\ 0.434] - [0.074;\ 0.508;\ 0.434] = [0;\ 0;\ 0]$

$K = [0.074;\ 0.508;\ 0.313] - [0.074;\ 0.508;\ 0.434] = [0;\ 0;\ -0.121]$

$E = [0.077;\ 0.524;\ 0.388] - [0.074;\ 0.508;\ 0.434] = [0.003; 0.016; -0.046]$

1.2.3. Find $\overline{IK}$

1.2.3.1. $\overline{IK}$ = [0; 0; -0.121] - [0; 0; 0] = [0; 0; -0.121]

1.2.4. Find $\widehat{IK}$

1.2.4.1. $|\overline{IK}| = \sqrt{0^2 + 0^2 + (-0.121)^2} = 0.121$

$\widehat{IK}$ = [0; 0; -0.121] / 0.121 = [0; 0; -1]

1.3. Derive the Rodrigues Rotation Formula: a formula for rotating a point a desired number of degrees around any normal axis

1.3.1. $\vec{E}_{parallel} = component\ of\ \vec{E}\ that\ is\ parallel\ to\ \widehat{IK}$

$\vec{E}_{perpendicular} = component\ of\ \vec{E}\ that\ is\ perpendicular\ to\ \widehat{IK}$

$\vec{E} = \vec{E}_{parallel} + \vec{E}_{perpendicular}$

1.3.2. $\vec{E}'_{parallel} = \vec{E}_{parallel}$

1.3.2.1. $\vec{E}' = \vec{E}_{parallel} + \vec{E}'_{perpendicular}$

1.3.3. $\Theta = angle\ from\ 1.1$

1.3.3.1. $\vec{E}' = \vec{E}_{parallel} + cos\Theta * \vec{E}_{parallel} + sin\Theta * (\widehat{IK} \times \vec{E})$

1.3.3.2. $\vec{E}' = (1 - cos\Theta) * \vec{E}_{parallel} + cos\Theta * \vec{E} + sin\Theta * (\widehat{IK} \times \vec{E})$

1.3.4. Final Rodrigues Rotation Formula

1.3.4.1. $\vec{E}' = (1 - cos\Theta) * (\vec{E} \bullet \widehat{IK}) * \widehat{IK} + cos\Theta * \vec{E} + sin\Theta * (\widehat{IK} \times \vec{E})$

1.4. Using the Rodrigues Rotation Formula

1.4.1. $\vec{E}'$ = (1 - cos(0.43)) * ([0.003; 0.016; -0.046] · [0; 0; -1]) * [0; 0; -1] + cos(0.43) * [0.003; 0.016; -0.046] + sin(0.43)*([0; 0; -1] x [0.003; 0.016; -0.046])

$\vec{E}'$ = [0.00288; 0.01602; -0.046]

1.5. Move the origin back

1.5.1. $\vec{E}'_{IK}$ = [0.00288; 0.01602; -0.046] + [0.074; 0.508; 0.434] = [0.07688; 0.52402; 0.388]

1.6. Repeat steps 1.1, 1.2, and 1.4 with the axes as IJ and the axis through the ICR orthogonal to the plane containing I, J, and K, making sure that the origin is moved to the correct point, I for IJ and the ICR.

1.6.1. $\vec{E}'_{IJ}$ = [0.077; 0.52238; 0.38747]

$\vec{E}'_{ICR}$ = [0.07836; 0.524; 0.38929]

1.7. Average the three E values

1.7.1. $\vec{E}'$ = [0.07741; 0.52347; 0.38825]

1.8. Repeat 1.1 - 1.7 for points A, B, C, D, I, J, K, and the ICR

1.9. Repeat 1.1 - 1.8 until total abduction angle is 120° while always using the most recent I, J, K and ICR values

1.10. Repeat 1.1 - 1.9 to find all points for flexion

2. How to find how point F rotates. The example used will also be the humerus abducting from 0° to 10°.

2.1. Move the origin to B, as this is where the humerus is rotating around

2.1.1. B = [0.203; 0.509; 0.458] - [0.203; 0.509; 0.458] = [0; 0; 0]

F = [0.21; 0.515; 0.13] - [0.203; 0.509; 0.458] = [0.007; 0.006; -0.328]

2.2. Set the axis of rotation for abduction

2.2.1. $\hat{n}$ = [0; -1; 0]

2.3. Use the Rodrigues Rotation Formula

2.3.1. $\vec{F'}$ = (1 - cos(10)) * ([0.007; 0.006; -0.328] · [0; -1; 0]) * [0; -1; 0] + cos(10) * [0.007; 0.006; -0.328] + sin(10)*([0; -1; 0] x [0.007; 0.006; -0.328]) = [-0.05001; 0.006; -0.32423]

2.4. Move origin back

2.4.1. $\vec{F'}$ = [-0.05001; 0.006; -0.32423] + [0.203; 0.509; 0.458] = [0.15299; 0.515; 0.13377]

2.5. Now the team must account for the movement of the humeral head in the XZ plane

2.5.1. B' - B = [0.2029; 0.509; 0.4589] - [0.203; 0.509; 0.458] = [-0.0001; 0; 0.0009]

2.5.2. F' - (B' - B) = [0.15299; 0.515; 0.13377] - [-0.0001; 0; 0.0009] = [0.15309; 0.515; 0.13287]

2.6. Now the team must account for the movement of the humeral head along the y-axis while keeping F in the same XZ plane.

2.6.1. Create a circle, in the same XZ plane as F, of all possible locations of F with the knowns of the length of the humerus and the location of the humeral head at B'

2.6.1.1. Center of Circle = [$X_{B'}$; 0.515; $Z_{B'}$] = [0.2029; 0.515; 0.4589]

2.6.1.2. Radius of Circle = $\sqrt{Length_{Humerus}^{2} - (Y_{B'} - 0.515)^{2}}$ = $\sqrt{0.3683^{2} - (0.509 - 0.515)^{2}}$ = 0.13561

2.6.2. Create a vector from the center of the circle to the current position of F and normalize it

2.6.2.1. $\vec{R}$ = [0.2029; 0.515; 0.4589] - [0.15309; 0.515; 0.13287] = [0.04981; 0; 0.32603]

2.6.2.2. $|\vec{R}|$ = [0.04981; 0; 0.32603] / 0.32894 = [0.15173; 0; 0.98842]

2.6.3. Find the closest point on the circle to the current position of F. This point on the circle is F'

2.6.3.1. F' = [0.15309; 0.515; 0.13287] + 0.3683 * [0.15173; 0; 0.98842] = [0.20897; 0.515; 0.49691]

2.7. Repeat 2.1 - 2.6, using the correct B and B', until the total abduction angle is 120°

2.8. Repeat 2.1 - 2.7, using [1; 0; 0] as the axis of rotation, to find the values of F for flexion

3.5 Material Selection

3.5.1 Motor Selection

The motors used by the 2021-2022 and 2022-2023 teams were the Nema 17 Stepper Motor model number 17HS19-1684S-PG27, which will be shortened to the 17HS19 model for the duration of the report. The 17HS19 model has a number of specifications that allows for efficiency, however, our team had concerns in utilizing these motors due to the amperage requirements of 1.2 Amps. To circumvent this issue, our team looked for other models of Nema 17 motors and decided on the use of the 17HS13-0404S-PG27 motor, which will similarly be shortened to the 17HS13 model. The qualities that make the 17HS13 motor the ideal are the rated current and the maximum permissible torque. The specifications for the 17HS13 motor are given in figure 32 below.

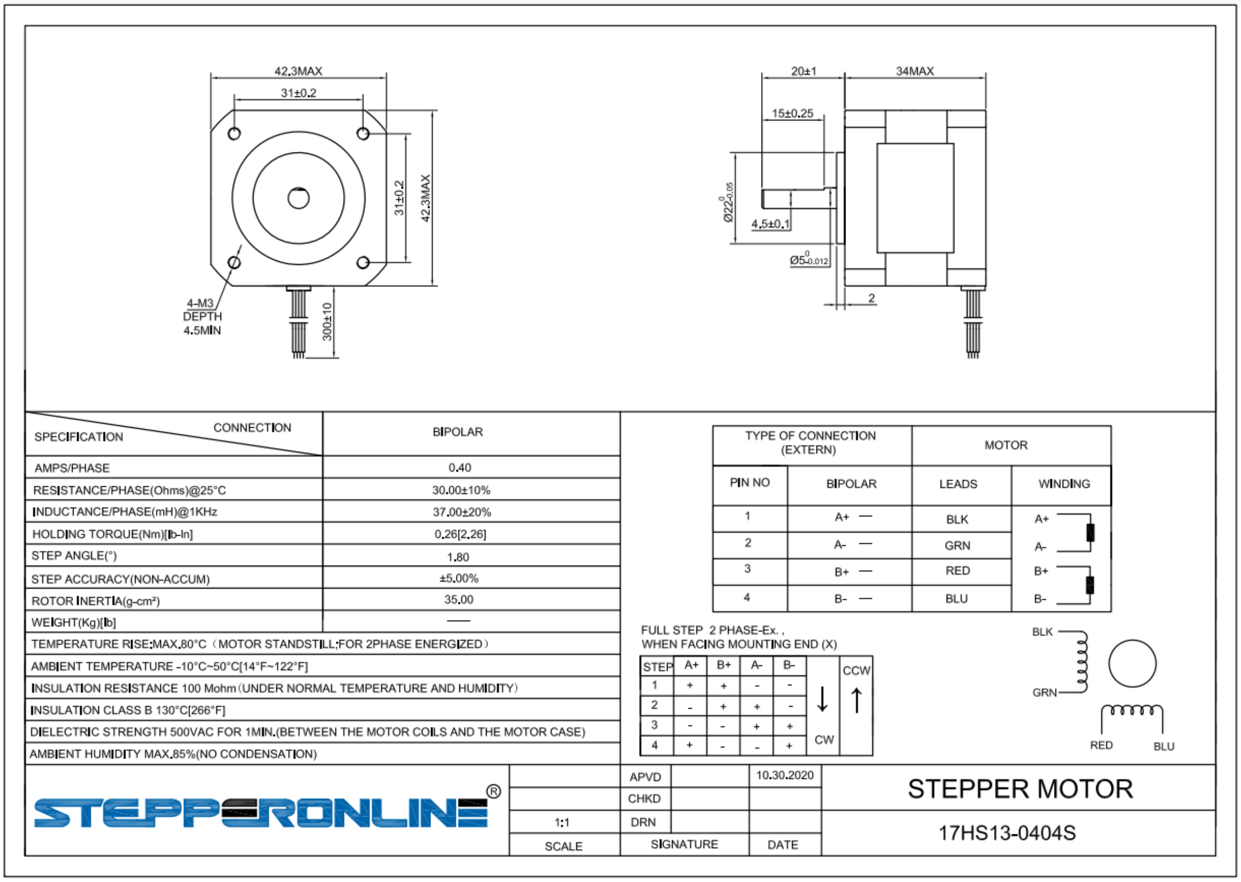

Figure 32: The model 17HS13-0404S-PG27 data sheet (StepperOnline, 2020)

The low rated current of 0.4 amps allows the motor to function on low currents, meaning the circuit will need to involve fewer pieces and draw less current and power into the system, exerting a lower strain on the equipment. The required voltage input for the motors was 12V. The max permissible torque of 3 Newton meters for each motor is capable of pulling on the humerus and scapula with adequate force to induce the rotation and transitions necessary to replicate scapulohumeral rhythm in the shoulder. The two previous teams both used five of the model 17HS19 motors to pull the humerus along their trail of wire that led along the length of the humerus.

Figure 33: The NEMA model 17HS19-1684S-PG27 motor (StepperOnline, 2020)

For our rig design, discussed in section 3.2, ten model 17HS13 motors were implemented to ensure correct positioning and stability of the humerus and scapula during humeral abduction. We attached the motors to the cage using 3D printed mounts. The locations of the motor placement within the cage are similarly discussed in section 3.2.

Figure 34: The NEMA model 17HS13-0404S-PG27 motor (StepperOnline, 2020)

3.5.2 Connecting Wires

Utilizing the correct materials for each material that the team is trying to simulate is crucial to the project's success. The first team developed a shoulder model with the bones properly assembled. For the material of the bones, they used PLA due to its rigid properties and ease of molding through 3D printing into the desired shapes. Our team will be similarly utilizing PLA for this aspect of the model.

The first team utilized nylon wires as a way of moving the bones to their desired location due to their strength and flexibility, whilst still remaining cost-effective. However, this team did not account for the issue of creep. The deformation caused by the creep changes the length of the nylon wires and thus changes the calculations that must be done to pull on the bones with the right force vectors. The second team utilized rubber threads to better operate under creep conditions, however, the friction caused between the rubber threads and other components of the rig hampered movements. To eliminate these issues, our team decided to change the material used for the connecting wires.

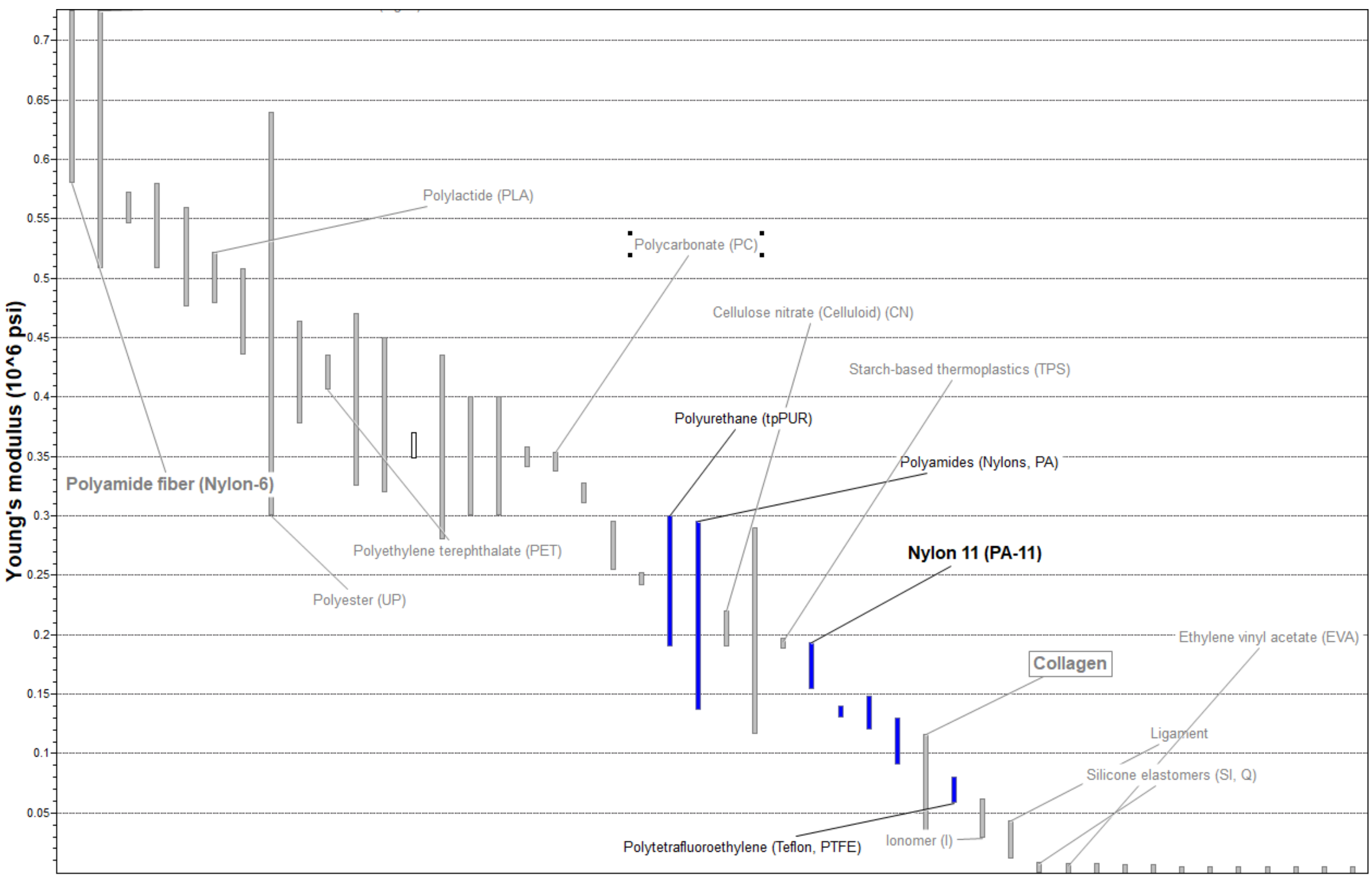

Figure 35: The Young's modulus (10^6 psi) of different materials similar to collagen

To decide on the material replacing the nylon wires, our team calculated the stress and strain that the nylon is under while it is pulling on the bones and the temperature of the room the model is in. With these two variables, the team calculated the strain rate of each nylon strand using the equation $\varepsilon = A\sigma^{n} exp(\frac{-Q}{RT})$, where ε is the strain rate, A and n are material based constants, σ is the applied stress, Q is the activation energy of the creep mechanism, R is the gas constant, and T is the ambient temperature in Kelvin. The applied stress, σ, was calculated with the equation $\sigma = \frac{F}{A}$, where F is the applied force delivered to the wire from the motor and A is the area of the wire that the force is delivered across. The area of the wire was calculated to be 0.00126 in^2 or $3.243 * 10^{-6}$ m^2 with an applied load of 11 kg. These calculations were completed in MatLab, with a final value of 3.4 MPa for the stress. We performed our calculations based on a wire with a diameter of 0.04 inches and material constants $A = 3.73 * 10^{-5}$ and n = 4.

Several options for the connective wires were discussed including a number of polymers and metals as well as the previously utilized nylon 6 wires, however, each material encountered different issues. The nylon 6 wires, as discussed above, were susceptible to creep. Silver alloys were too soft and could easily be damaged. Copper alloys would be susceptible to environmental factors. Finally, various steels were similarly proposed but were too resistant to bending, far too expensive, and had high friction coefficients, making their movements in the model more difficult than necessary. With this in mind, our team considered PTFE coated wires to mitigate friction between the wire and any surfaces contacted, however, a suitable product was not commercially available.

Ultimately, the team utilized braided nylon fishing line to achieve a balance between all sought characteristics. With the values attained, the wire would need to be capable of supporting a maximum of 13.8 MPa without experiencing creep, whilst still being capable of repeatedly bending in the process of pulling the shoulder model into position and not inhibiting the movement of the humerus and scapula all of which braided fishing line achieves.

Figure 36: Braided Fishing Line

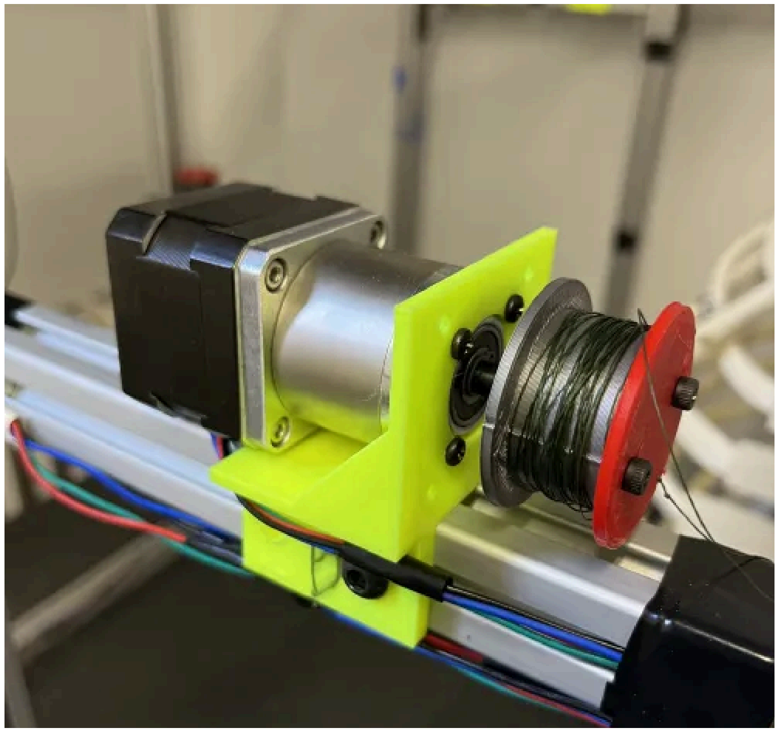

Figure 37: Braided Fishing Line attached to a 17HS13 motor

3.5.3 Motor Driver

We used a stepper motor driver in order to control the motors. The brand name for the driver is ‘Allegro’s A4988 DMOS Microstepping Driver with Translator and Overcurrent Protection’. We bought our driver from Amazon which was manufactured by Shenzhenshi Yongfukang Technology Co. The A4988 has 16 pins of which our team utilized 10 pins. The Vmot and GND are connected to a 9V battery. The 2B, 2A, 1A, and 1B pins are connected to the four motor wires. Figure 39 below shows the corresponding colors to letters. The wiring of the motor to the driver can be arranged in a multitude of variations as long as the one coil on the motor is not connected to two different coil outputs on the driver.

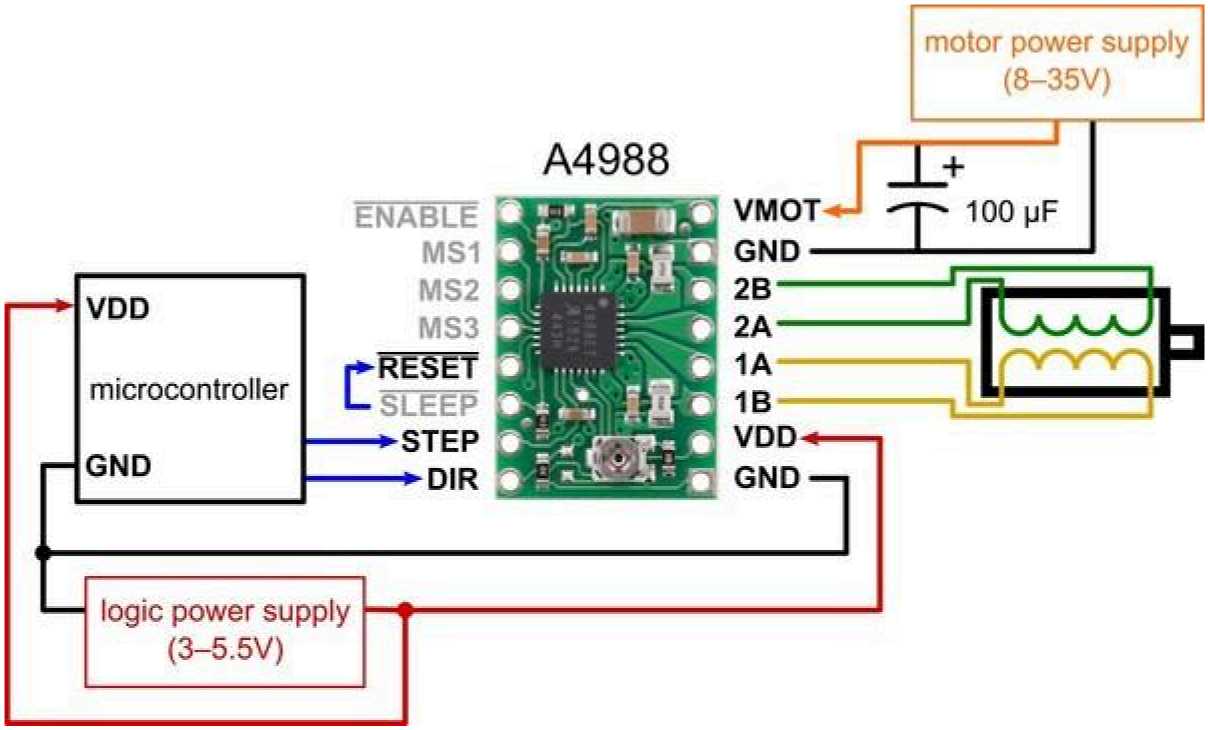

Figure 38: The figure of the driver (Pololu, 2024)

A4988	Motor Datasheet	Color
1A	A+	Green
1B	A-	Black
2A	B+	Red
2B	B-	Blue

Table 8: The chosen orientation of the wires from the motor to the driver

In order to set up the motor driver we needed to adjust the current limit potentiometer. The following equation is used to find the voltage reference for the driver.

$$V_{ref} = 8 * I_{max} * R_{cs}$$

The I_{max} is the max current rated for the motors we were using which is 0.4A. R_{cs} is the current sense resistance of the driver motor, $R_{cs} = 0.1\ \Omega$. Therefore, our V_{ref} would be 0.32V. The figure below shows one of the ways to adjust the current limit potentiometer. For the VΩ port we used an alligator clip that was connected to a screwdriver that we turned the potentiometer with.

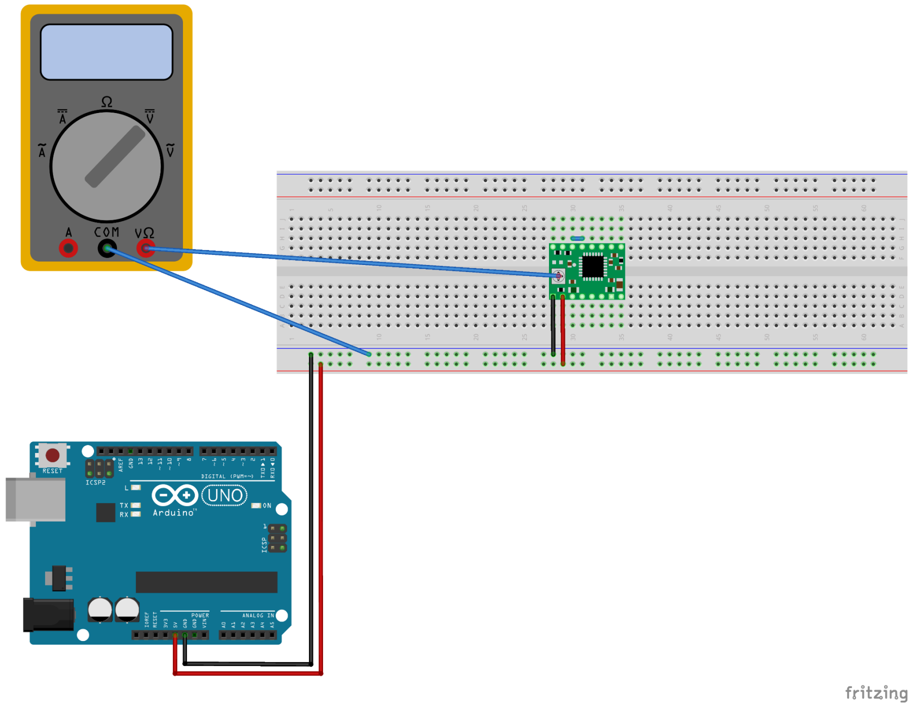

Figure 39: How to adjust current limit potentiometer

Another important note about the A4988 motor driver is the temperatures it can operate at. The maximum junction temperature is 150℃, attaching the heat sink to the motor driver aids to lower the temperature but the drivers can still cause burns with direct contact at operational temperatures.

3.6 Cage Construction and Implementing a Matlab Code

3.6.1 The Cage

The cage surrounding the rig was constructed utilizing aluminum extrusion rods present available in the WPI MQP labs. The exact dimensions of the rods vary and are listed in table 9 below.

Part No.	Aluminum Extrusion Rod Dimensions (Length X Width X Depth)	Quantity
1010 T-Slotted Profile - Four Open T-Slots	0.912m X 1.00” X 1.00”	2

1010 T-Slotted Profile - Four Open T-Slots	2.44m X 1.00" X 1.00"	2
20-2020 T-Slotted Profile - Four Open T-Slots	0.695m X 20mm X 20mm	2
1003-s Smooth Surface T-Slotted Profile - Three Adjacent Open T-Slots	2.44m X 1.00" X 1.00"	1

Table 9: Details of Aluminum Extrusion Rods

The minimum dimension of the cage is 0.5 meters by 0.8 meters by 0.8 meters. Different views of the cage are displayed below in figures 39 through 43. The varying dimensions displayed in the figures are from the different sizes of aluminum extrusion available to the team.

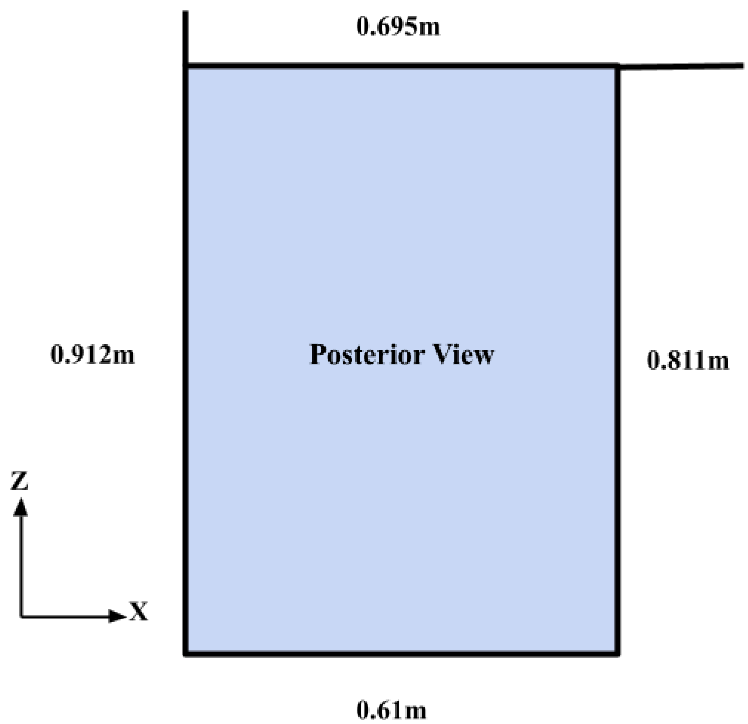

Figure 40: Posterior View of the Cage

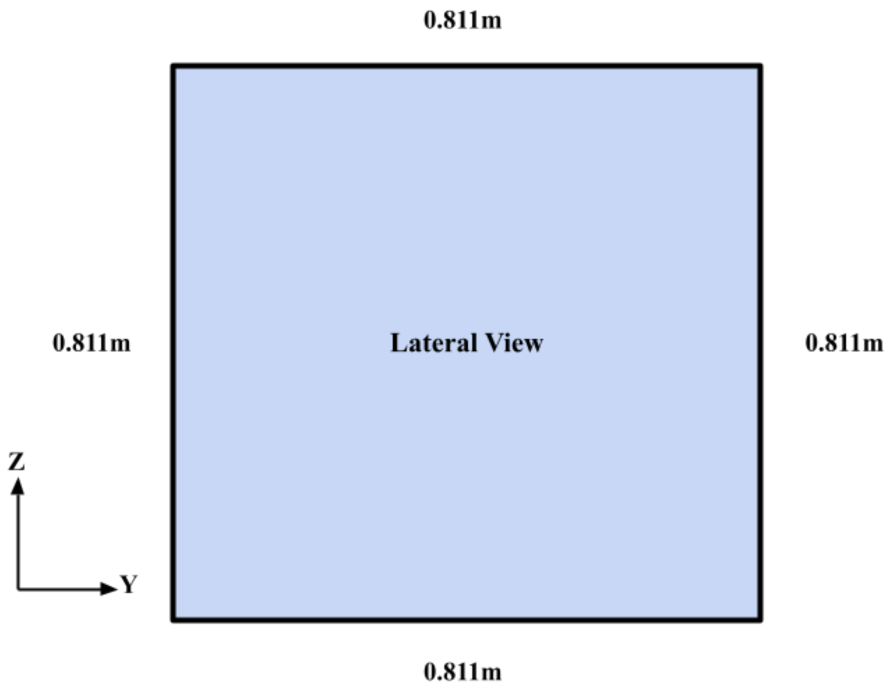

Figure 41: Lateral View of the Cage

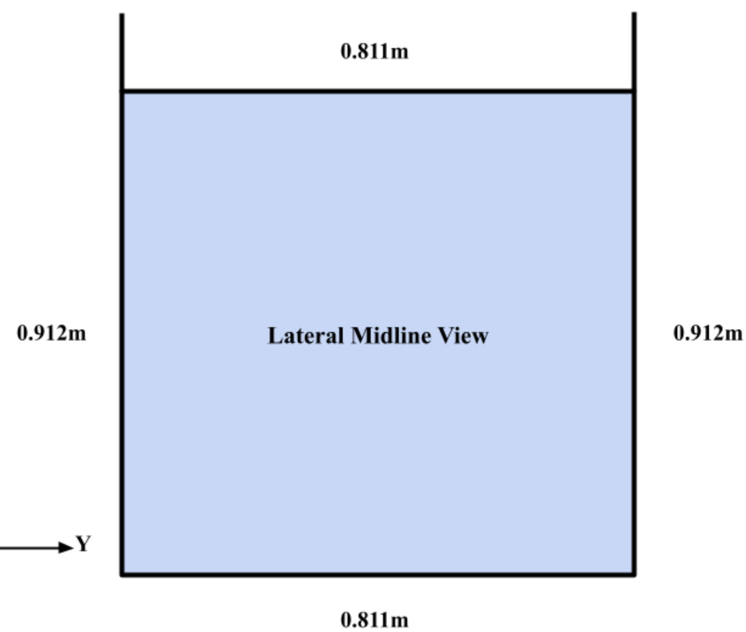

Figure 42: Lateral Midline View of the Cage

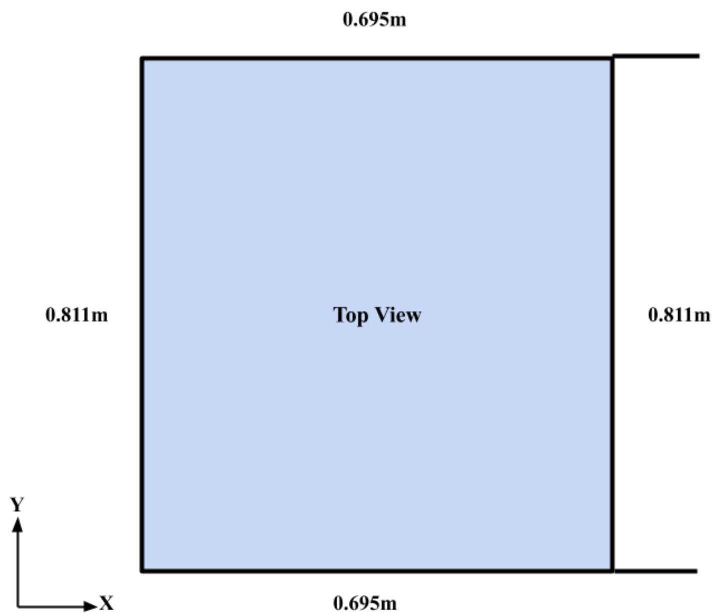

Figure 43: Top View of the Cage

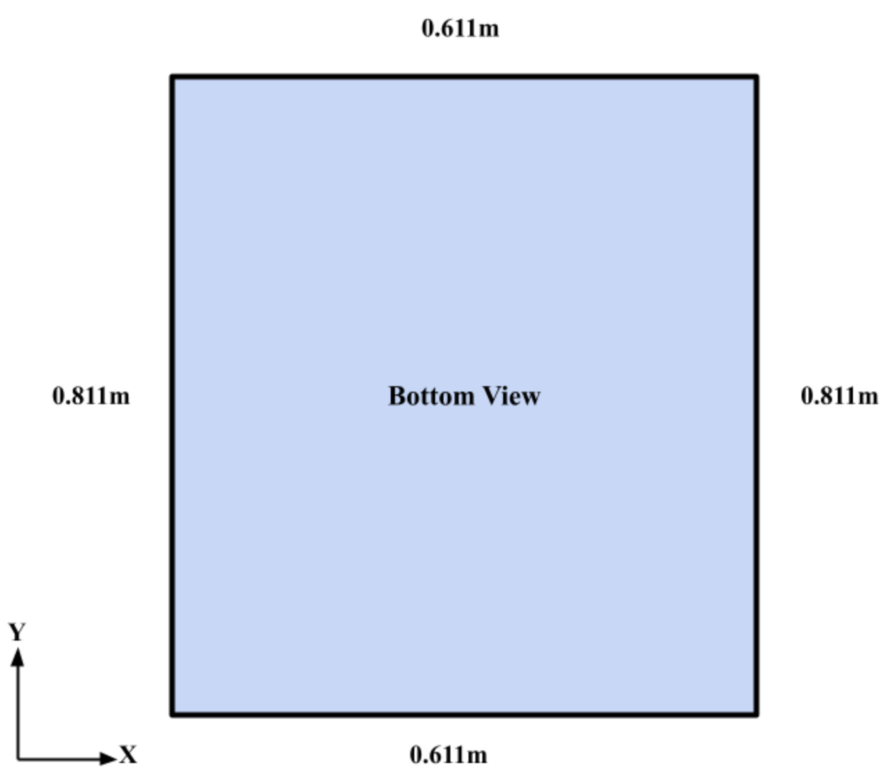

Figure 44: Bottom View of the Cage

Another issue we saw that was due to aging was wood rot in the original rib cage mount. We replaced the old plywood with a ¾ inch plywood. Additional bars were installed to further support the cage under the weight of the rig and correctly position the motors in accordance with Figure 25 in section 3.2.1 to pull on the 3D printed bones within. To minimize any clutter and ensure visibility of the rig, the wires for each motor run along the bars of the cage where they conjoin in the circuit on the backside of the board.

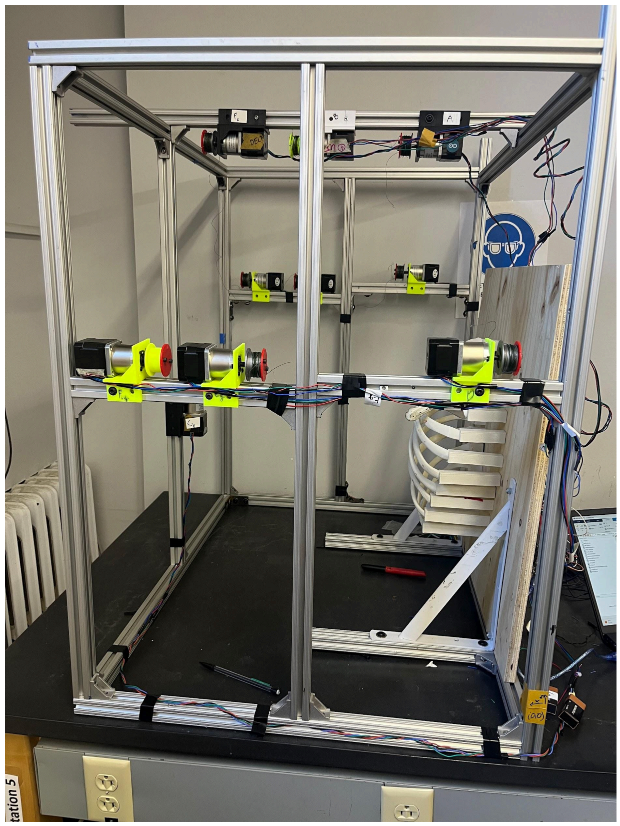

Figure 45: Supporting Bars of the Cage

3.6.2 Coding and the MATLAB Model

The team lead for this portion of the project was Cameron Leffler, who wrote MATLAB code to calculate the positions of each of the six wire attachment points on the scapula and humerus relative to the origin for every 0.1° of arm abduction and flexion. The following documentation explains how the code calculates all of the relative positions of the wire attachment points and wire lengths.

The Code is split into four scripts of code; Setup, Abduction, Flexion, and App code scripts, all of which are run using Arduinos connected in the circuit. The setup code defines the relative positions of the motors and of the motor attachment points for all possible angles of arm abduction and flexion.

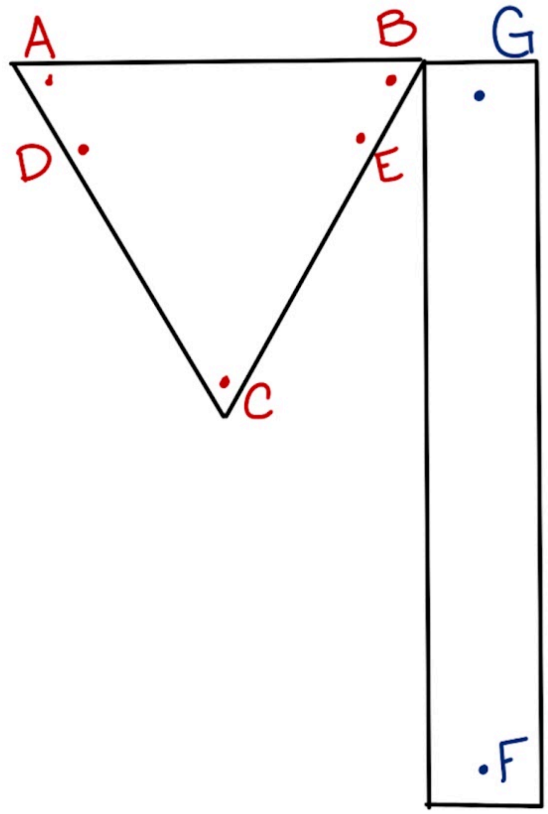

Figure 46: A simplified view of each of the motor attachment points

It does this in the manner stated in section 3.2 using the Rodrigues' Rotation Formula. It then calculates the distance from each motor to the motor attachment point, giving the length of thread between the attachment point and the motor and creates a global matrix of all thread lengths given a desired abduction or flexion angle. The app script graphs the movement of the rig through previous calculations and transmits that information to the abduction and flexion code. The abduction and flexion scripts take an input of a desired angle of abduction or flexion and change the thread length to the correct amount based on the desired abduction or flexion by rotating the motors. Cameron then took these relative positions and made an interactive 3D plot in MATLAB where a user can input an amount of abduction or flexion and the program will show four views of where the wire attachment points will be in 3D space, one angled view from behind and above, one view from straight on, one view from directly above, and one view straight on from the side, as shown in Figure 56 in Section 3.8.2.

Using the MATLAB code we can determine the position of each point throughout scapular and humeral movement. In figures 46 through 51, we determined the acceleration of each point throughout abduction. The data for the graphs below is made up of over 10,000 calculated data points so the addition of variables was made to make the data more digestible in an excel spreadsheet. The duration of abduction and flexion can be altered by changing the total time and the degrees between data points can be altered to get an average of the data for each point. We wanted to portray the change in acceleration of each point to convey the complexity of the scapulohumeral rhythm.

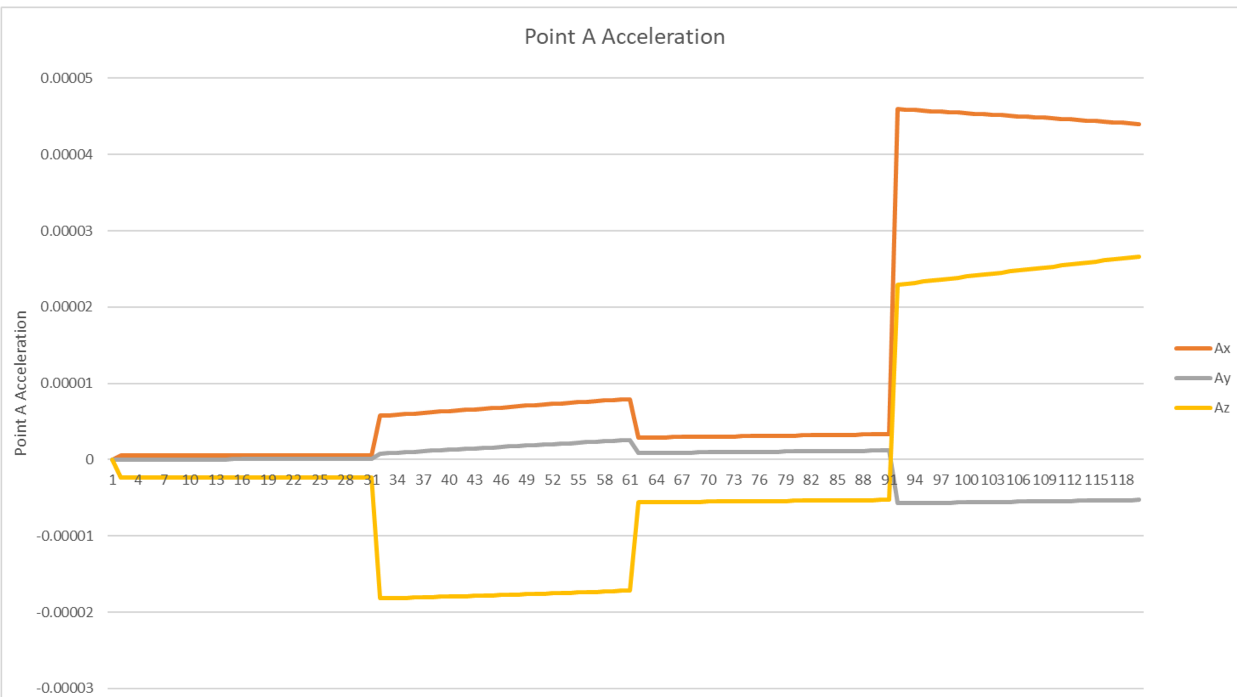

Figure 47: Acceleration of point A over degrees Abduction

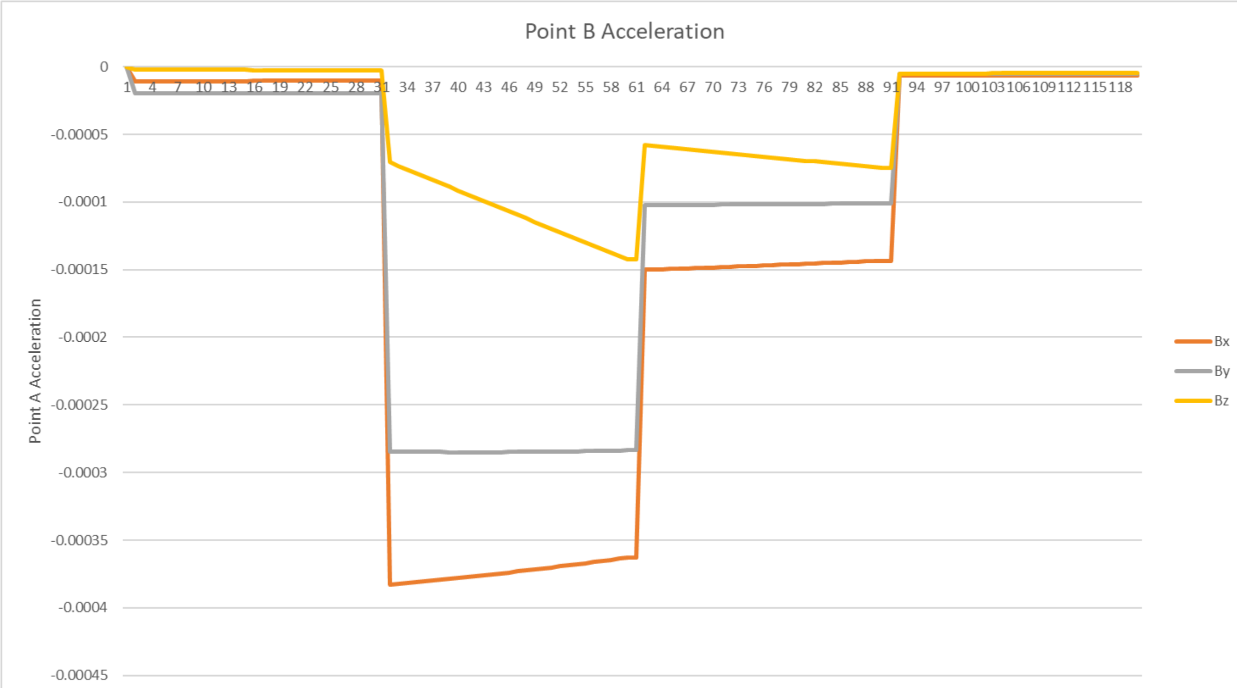

Figure 48: Acceleration of point B over degrees Abduction

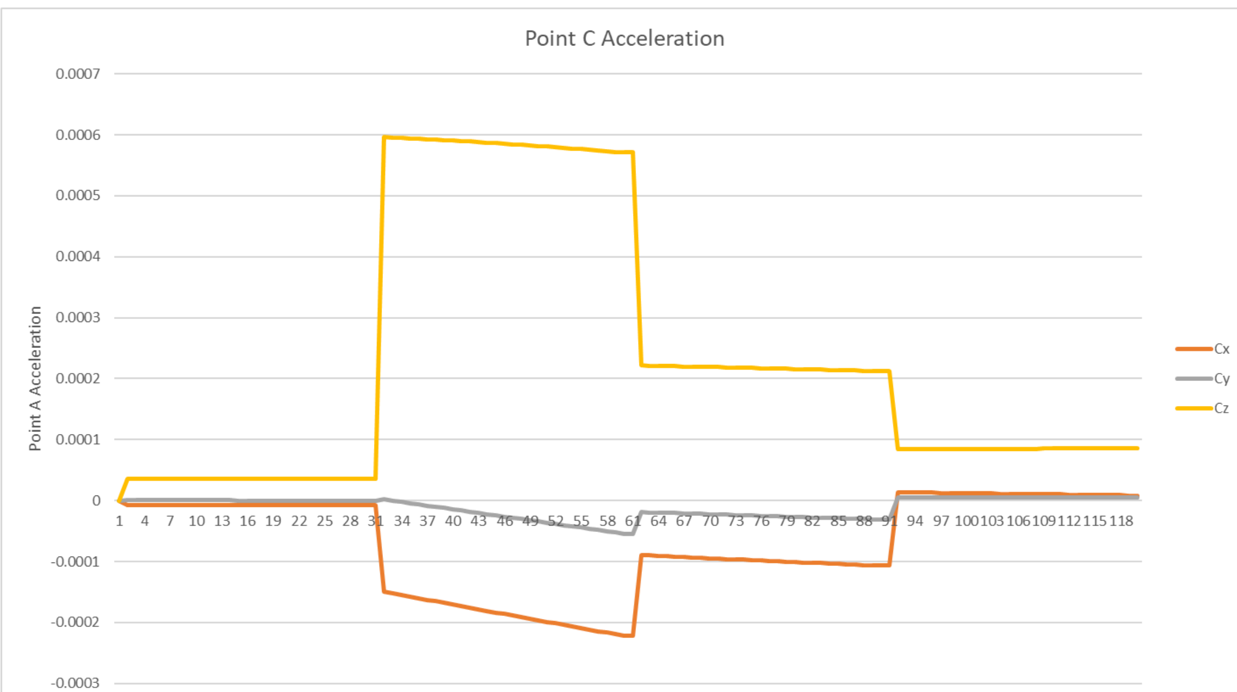

Figure 49: Acceleration of point C over degrees Abduction

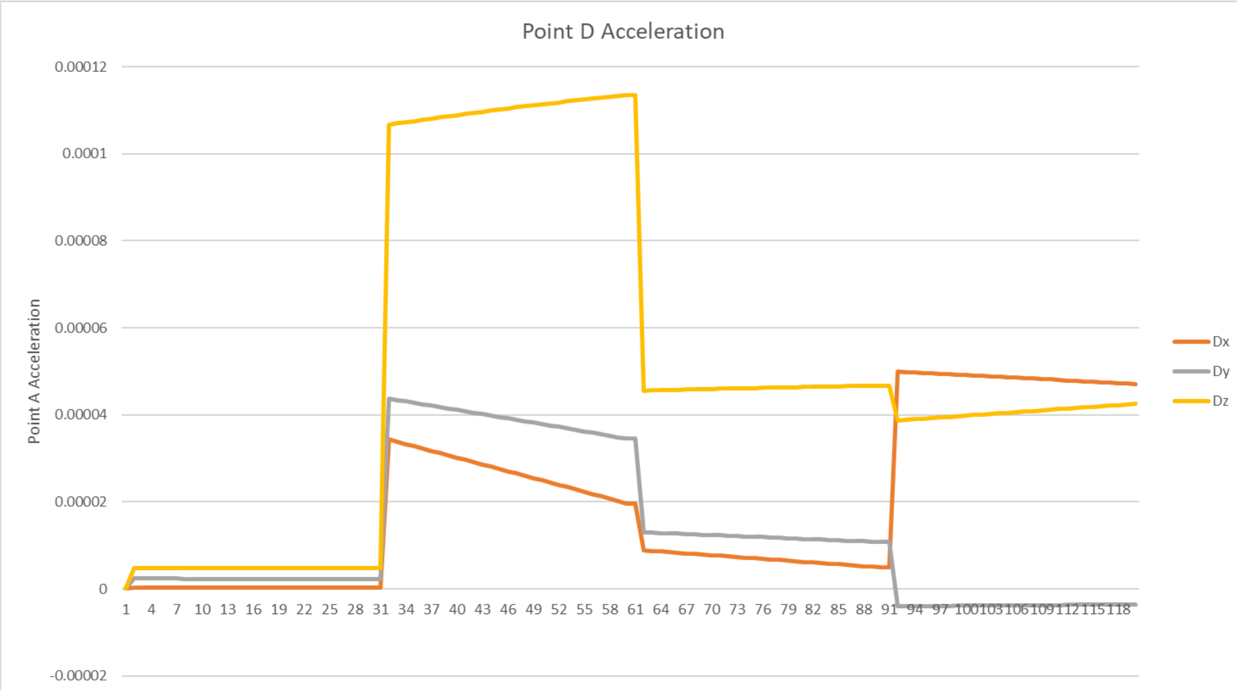

Figure 50: Acceleration of point D over degrees Abduction

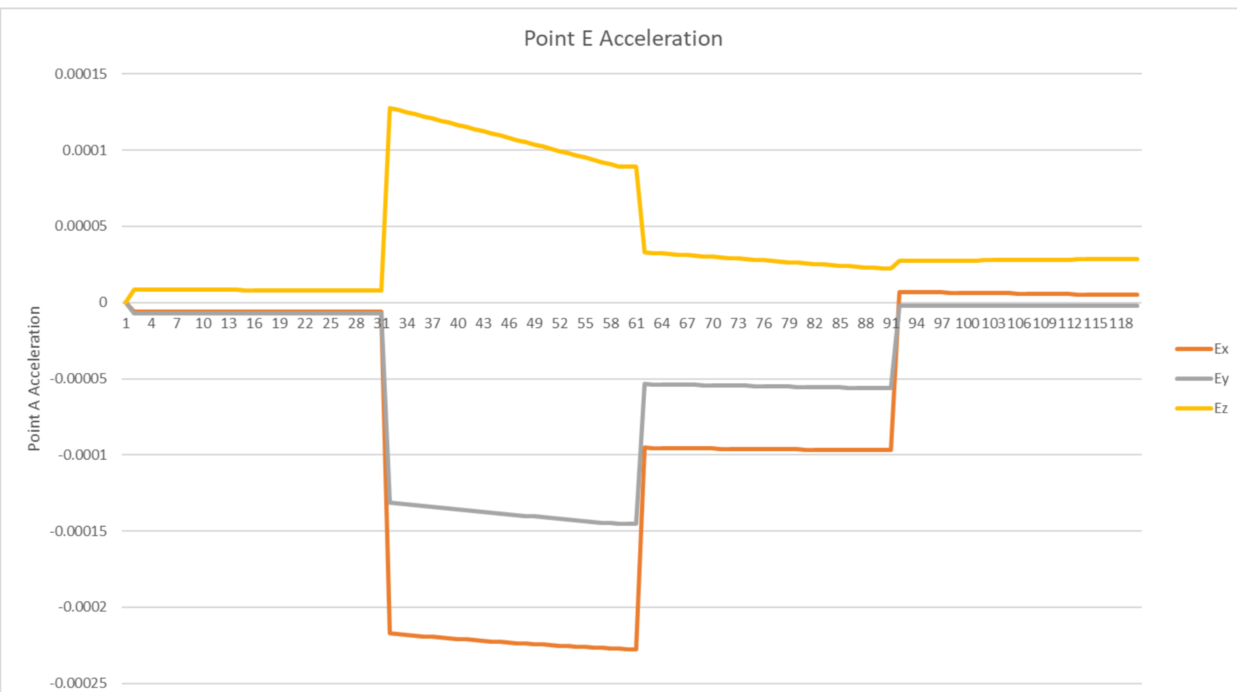

Figure 51: Acceleration of point E over degrees Abduction

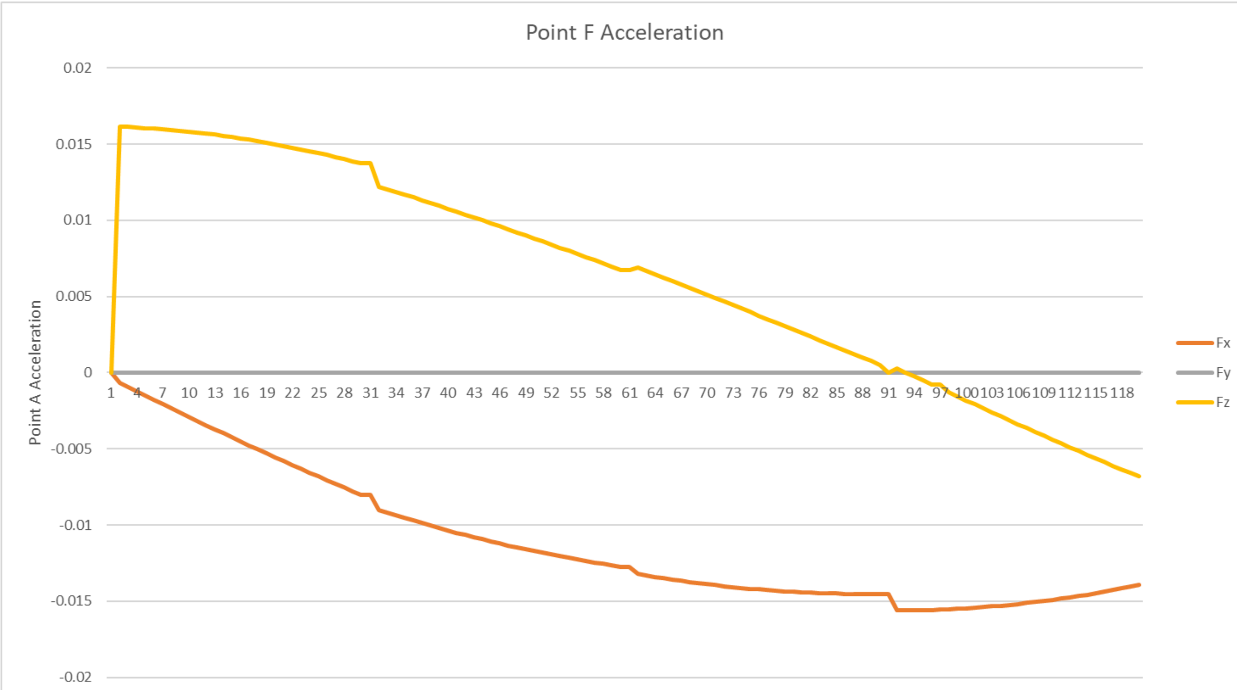

Figure 52: Acceleration of point F over degrees Abduction

We can also use the thread lengths to check our work along with measuring the distances by hand. We can measure xyz manually to make sure the ijk vectors are correct and measure the thread length to confirm the overall vector acceleration. The wire we are using is not elastic under the delivered force so we know that the delta thread length is relatively true.

3.6.3 Circuit Design

The circuitry utilized for the rig underwent a large change from the 2021-2022 team's circuit design. The 2021-2022 circuit design, displayed below in figure 54, is a simple parallel circuit designed to deliver power and current to each motor simultaneously so as to ensure each motor receives enough voltage to run, however, the equipment used to build the circuit was inadequate to serve our purpose. The breadboard utilized to build the circuit was not built to handle a current higher than 1 amp, which is almost doubled when all five motors are run through the circuit. Additionally a single 9-volt battery is incapable of delivering enough power to support five motors at once, let alone the ten our team would implement.

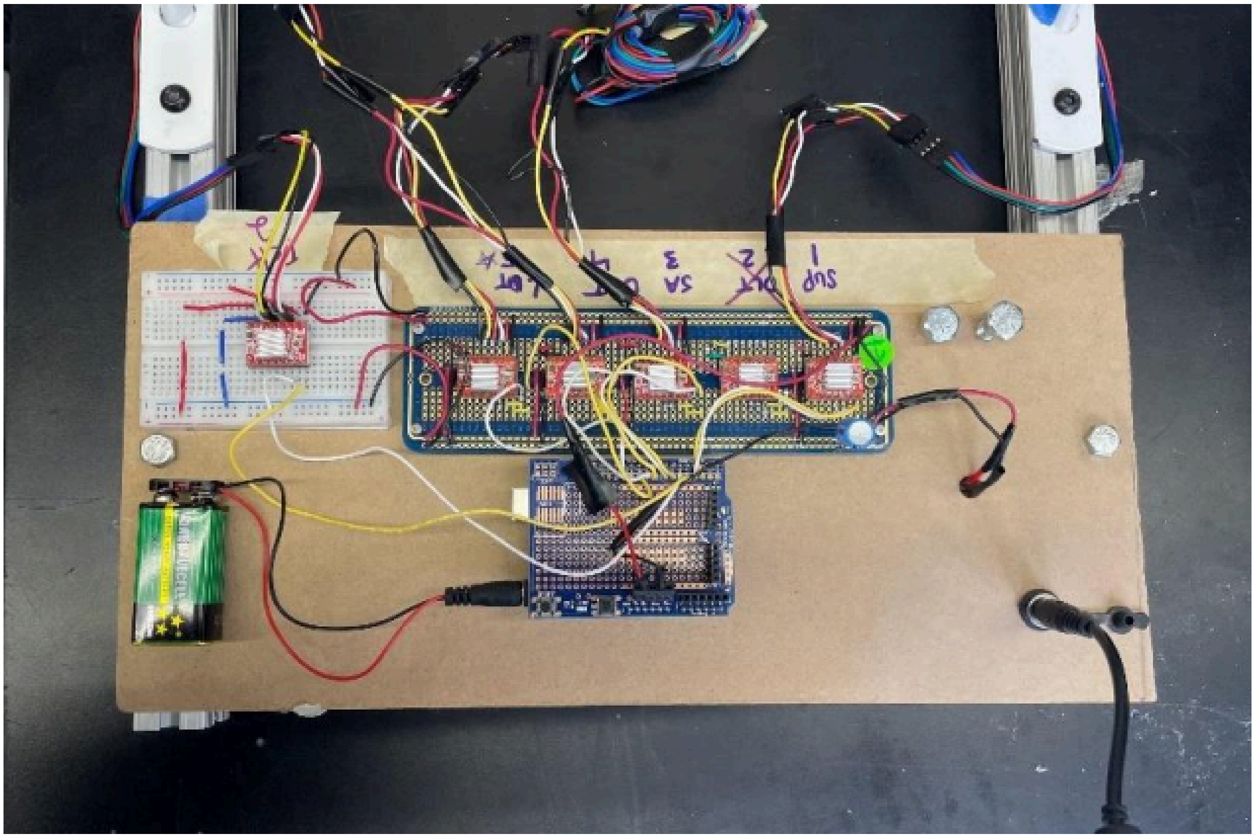

Figure 53: The 2022-2023 Team's Fully Assembled Physical Circuit (McEvilly et al., 2022)

To resolve these issues, our team opted to implement a larger power source, with the intention of utilizing numerous 9-volt batteries at once with plans to ultimately plug the system into a wall outlet. Further alterations to the circuit include the addition of a second Arduino board, to ensure enough pins were available for the ten microprocessor-motor systems. The next iteration of the circuit design is displayed below in Figure 55.

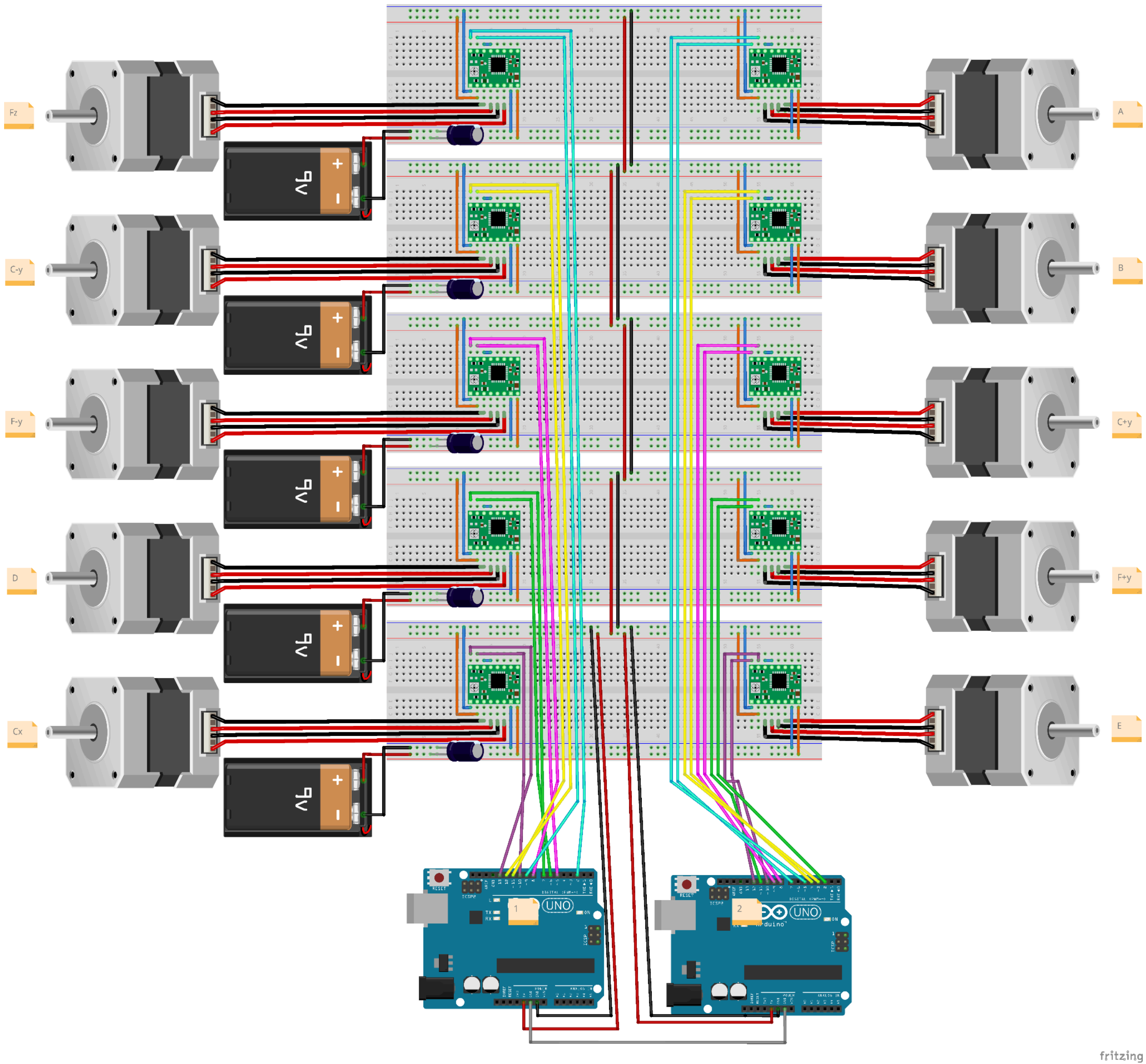

Figure 54: Second Iteration of the Circuitry

To circumvent the breadboard's limitations, breadboards were removed from the circuit, instead utilizing space studs, cable nail in clips, and terminal blocks to connect and complete the circuit. The team also experienced trouble operating both Arduinos. This was due to the fact that we had not adjusted the current limit potentiometer and we also had the power source of all ten motor drivers connected to the Arduinos in series. This overloaded one of the Arduinos causing it to short and turn off. To fix these issues we moved from breadboards to 10 circuit terminal blocks instead of the breadboard. We added five more 9V batteries to power each motor individually which we were hoping would improve our low voltage. To fix the shorting Arduino, instead of all ten of the motor drivers being powered by the 5V output from the Arduino which were run in one series we split it into two series of five. Each series is powered by their own 5V Arduino output pin . The updated schematic is shown in Figure 56 below. One of our largest oversights came with the needed voltage. The original consensus was that 9V would have been enough to power the motors but that was for the previous model of motor used. The 17HS19-1684S-PG27 only needed 2.8V in order to turn, whereas the 17HS13-0404S-PG27 needed around 12V to properly run.

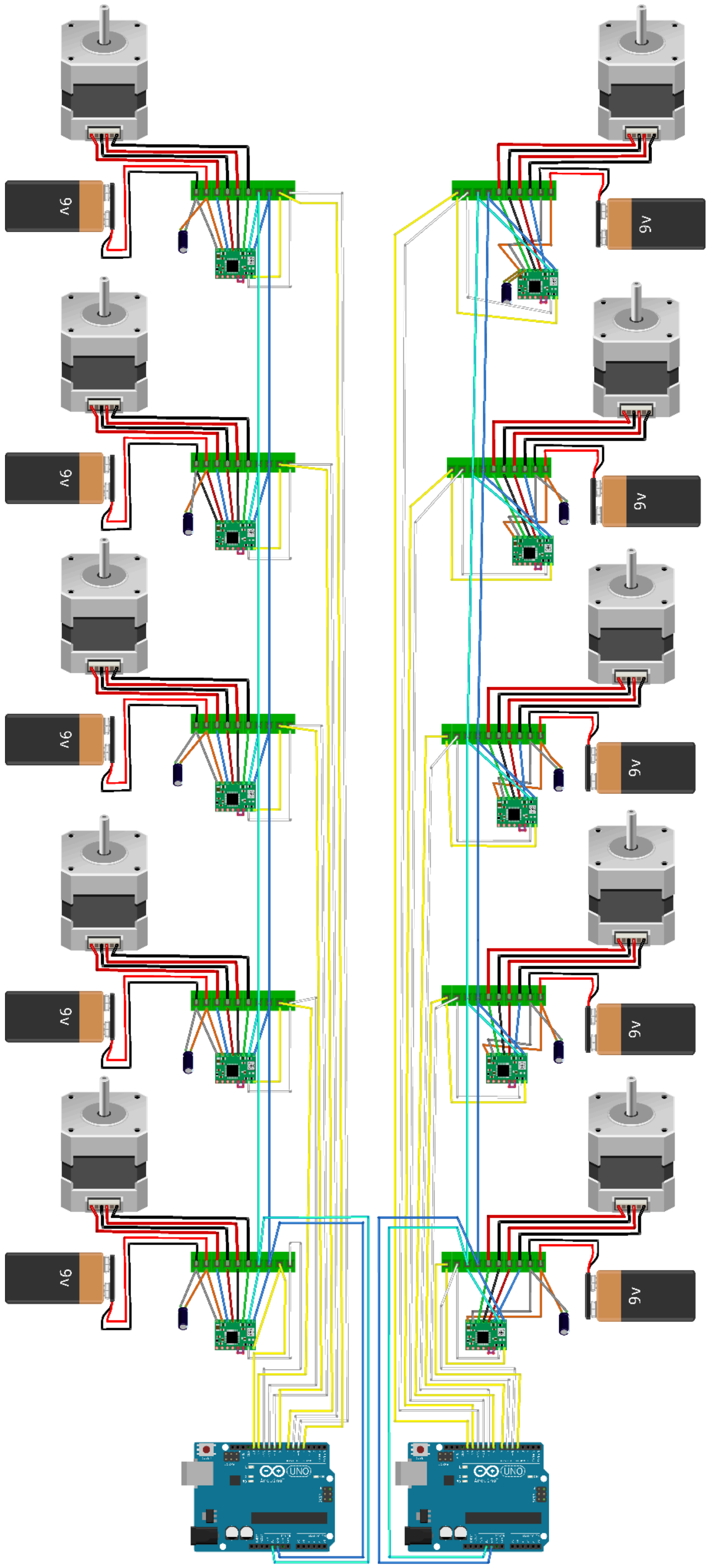

Figure 55: Updated Schematic of the Circuitry

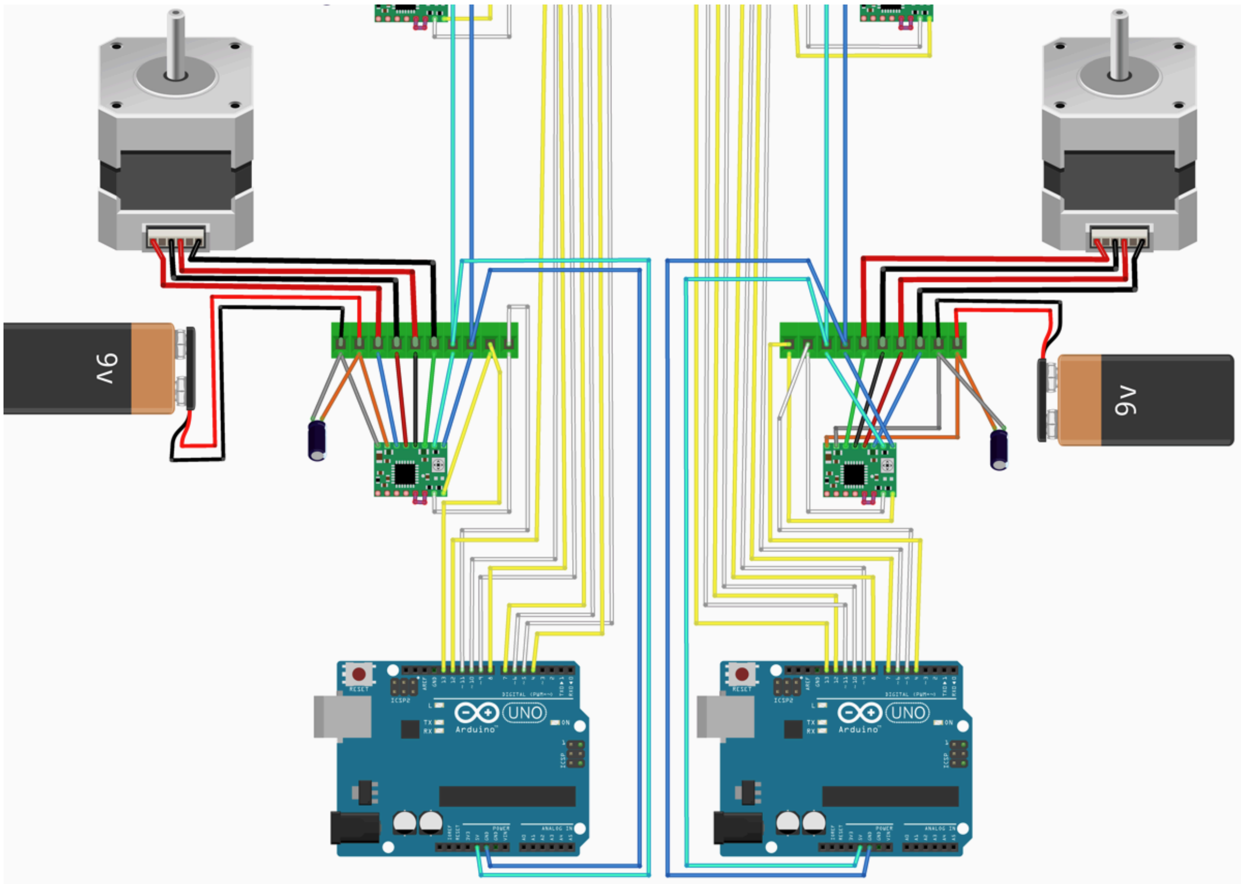

Figure 56: Zoomed in Schematics of the Arduinos

On the A4988 motor driver the RESET pin needs to be HIGH so our team connected it to the SLEEP pin which pulls HIGH by default. We also used a 100uF capacitor to regulate power surges between our motor power supply and the driver. When picking a place for the capacitor our team placed it as close to the driver as possible so there is less surge throughout the wire, especially if the distance between the batteries and the motor drivers is larger.

3.7 Testing

The team successfully constructed the cage and attached the motors to the surrounding frame. We were unable to conduct testing of the fully constructed rig due to time constraints. While the team did not get all 10 motors rotating at once, the team did test to see if the motor positions were able to achieve the correct relative positions of the 3D printed bones. In doing so, all ten motors were tested for their ability to function as expected in the rig as described in the following sections.

3.7.1 Testing Relative Position

The team chose to test the relative positions at rest, then in intervals of 30 degrees of abduction and flexion, ending at 120 degrees of abduction and flexion. The team made this decision as in between these checkpoints the motion of points on the scapula is linear. The team manually set the thread lengths to their desired lengths and suspended the scapula and humerus from the motors in order to show a proof of concept.

3.7.2 Motor Calibration

Motor calibration tests were based on finding out how much the motor changed the thread length as a function of how many seconds the motor spun. The first step was finding the angular velocity of the motor. To do this, the team made a simple MatLab code that can spin the motor for an inputted number of seconds. The team then imputed one, five, and ten seconds and measured the degrees of rotation with tape and a protractor. To refine this measurement, the team made a MatLab code that inputted desired degrees of rotation and spun the motor as such. The team ran the code to ten full rotations, or 3600 degrees, and manually changed the conversion coefficient–that converts from degrees to seconds of rotation–until the motor could spin 3600 degrees within half a degree of precision. The team then calculated, based on the diameter of the spool, how much thread the motor was moving per second and updated the conversion coefficient–that converts from thread length moved to seconds of rotation–in the final code.

3.7.3 Testing Individual Motors

In order to test individual motors, the team wrote a simple MatLab code that talked to one motor at a time and was easily switched between which motor was being tested. Motors were identified to work upon the activation of the code, where the motor would begin to rotate in accordance with the written script. Motors that did not rotate or did not rotate in accordance with the script were identified as dysfunctional. Once it was determined if a motor was functioning properly or not the team troubleshooted to identify the error, further details on the troubleshooting process are provided in the following section.

3.7.4 Troubleshooting

Our team troubleshooted for errors throughout the process of constructing the rig. During the construction of the cage, an angle bar was utilized to ensure proper angles within the cage as well as physical exertion upon the bars to ensure they were properly secured and firm in their placement. Motor tests revealed a number of faulty motors that were promptly replaced or repaired through corrections within the code.

Throughout construction of the rig, circuitry for the rig proved to be the most complex to troubleshoot, as the presence of numerous components, each with their own individual ability to experience error, posed difficult to sort through. Removal of the breadboards, as described in section 3.3, was the first correction to attempt to circumvent the amperage limit that was imposed by the hardware. Further corrections to the circuitry were completed with the use of a DC power supply, allowing the circuitry to experience a number of different amperes and voltages combinations to expose further faults within the wiring by observing if the motors would rotate with varying inputs. By providing a wider range of voltages and amperes, we were able to identify what range of voltages and amperes worked with the circuit and ten 17HS13 motors. Additionally, a voltmeter was utilized to measure the voltage across circuitry components, to ensure the proper distribution of voltage across all key components. Several faulty capacitors were replaced in addition to corrections made to the Arduino boards.

3.8 Results

While relative motion of the scapular and the humerus was not achieved, the team made progress that will be valuable for the continuation of this project, by Cameron Leffler, in D-term and for future teams that continue the project.

3.8.1 Partially Completed Rig

The team constructed a nearly complete model, including the surrounding frame to support the humerus and scapula. The cage is capable of supporting all necessary forces to manipulate the bones to reproduce anatomically correct movements. The circuitry was similarly completed and is capable of delivering sufficient power and commands from the MatLab scripts to all ten motors of the rig.

3.8.2 Interactive Scapula Position Graph

The team lead for this portion of the project, Cameron Leffler, created an interactive graph within MatLab that, with two inputs between 0 and 120 degrees, one for abduction and one for flexion, would replicate the proper anatomical movements of a humerus and scapula within a confined graph. This motion is viewable from numerous varying angles. The viewer can select a degree from a bar located at the bottom of the window with intervals of 1 degree. Figure 57 shows the interactive app set at 90 degrees of humeral flexion. Due to constraints with setting the camera in the MatLab app maker, the axes in the figure are not correct, however, the relative motion is the same.

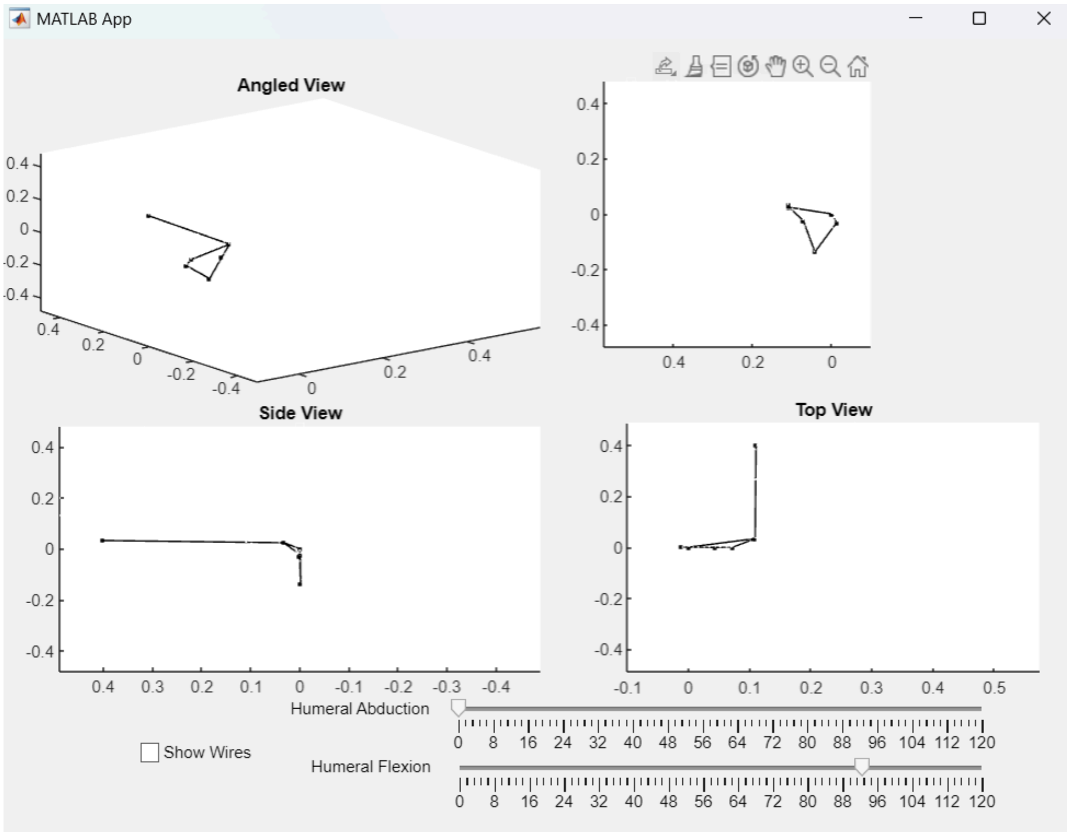

Figure 57: The Interactive Scapula Position Graph

4.0 Discussion

Through review of the results, the team was able to determine our successes and shortcomings throughout the duration of the project.

4.1 Motor Location and Function

The team was successful in relocating the motors to positions that would more easily allow freedom of movement for the rig through abduction and flexion. The surrounding motors pulling from locations external to the humerus and scapula provided far more ideal force vectors to manipulate the movement of the bones. The motors were summarily successful in their calibration with the Arduino code to accurately rotate the correct amount and retract the correct length of braided nylon fishing wire. Unfortunately, due to time constraints, a full test of the motors system was unable to be completed, leaving a lack of experimental results to supplement the theoretical calculations.

4.2 Materials Selection

The team was successful in selecting proper materials to mitigate complications during humeral abduction and flexion. The removal of the restrictive materials placed on the model by the 2022-2023 team, meant to simulate the muscles, ligaments, tendons, and other biological material within the shoulder allowed for freedom in designing the rig, with a larger focus on properly replicating the relative motion between the scapula and humerus. To minimize friction from connection points between the motors and the scapula and humerus, a selection of nylon braided fishing wire was utilized for the low friction coefficient alongside the durability presented by the material. Similar to section 4.1, time constraints limited the team's ability to experimentally test the materials selected.

4.3 MatLab Code and Interactive Position Graph

The team was successful in writing MatLab code to control 10 motors to accurately recreate the scapular humeral rhythm during humeral abduction and flexion and used MatLab to create an interactive position graph showing the relative motion of the scapula and humerus during humeral abduction and flexion. To the team's knowledge, both of these codes did not exist before this project. The relative positions from the code will be saved and used by Cameron Leffler to create a 3D animation of the scapular humeral rhythm during D-term.

5.0 Broader Impacts

5.1 Engineering Ethics

Over the duration of our project, we sought to be in accordance with the engineering code of ethics implemented by the American Society of Mechanical Engineering. Our team strived to use our understanding of engineering for the benefits of human welfare, increase the competence and prestige of the engineering profession, and to act with honesty and impartiality when developing this project. These fundamental principles integral to the engineering guidelines were held to high standards as we completed our project. This project was completed with the goal of aiding medical students through the use of an educational device, a goal created through the intention of providing beneficial material for the betterment of others. Through this report, we have strived to be truthful to establish integrity within our work. It is our sincere hope that our efforts reflect well on the engineering field.

Our project was a continuation of two prior MQP teams' efforts on the shoulder model, with great effort made to build off the foundation they created whilst respecting their own accomplishments. Their results and data were supplemented by our own research and findings to create a sufficient project that reflects the efforts of all the teams who have dedicated their work towards this shoulder model.

Finally, we sought to complete this project to the fullest extent of our abilities and produce the highest quality model we were capable of creating, utilizing numerous reputable sources and our own engineering expertise to create as accurate a model as possible.

5.2 Social and Global Impact

The primary intention of this model is to be utilized as an educational tool for the improved understanding of the shoulder joint and the complexities that lie within it. This educational tool would go on to benefit a subset of people within society, primarily students seeking to study human anatomy to become doctors. This tool would further these student's understanding of the complexities of the human shoulder, producing more educated doctors who will go on to provide better treatment to patients suffering from injuries, conditions caused by chronic illnesses or genetic defects. Additional applications towards injury prevention are possible such as utilizing the model to explore protective gear for athletes.

5.3 Environmental Impact

The new materials used for the model and rig include braided nylon fishing line wires and aluminum extrusion to construct the cage for the model. The aluminum extrusion used to construct the cage was reused from previous projects. The braided nylon fishing line connecting the bones to the 17HS13 motors is biodegradable and environmentally friendly. motors used within the rig are composed of a number of metals and electronic components which are not environmentally friendly. Finally, the current power source for the rig, 9V batteries are not environmentally friendly, however, the rig can be modified to draw power from electrical outlets to reduce this waste. The model is projected to be utilized over a large quantity of years without requiring significant replacements to parts or significant upkeep, allowing a minimum amount of materials to be wasted over the course of the model's lifespan.

Materials utilized for our rig by the previous teams includes PLA 3D printed bones and 3D printed motor mounts which are biodegradable and will decompose in 12 weeks time when disposed of. Other materials utilized by the previous teams included KT Tape, Formlabs Elastic 50A, Formlabs Flexible 80A, and Thermoplastic Polyurethane, all of which we removed from our design and model to minimize waste and clutter of the rig, and reduce our environmental impact.

5.4 Economic Impact

The cost of our modifications to the model was relatively low, and is tabulated below. The braided fishing line and aluminum extrusion were inexpensive. The stepper motors contributed to a far larger comparative expense. The electrical components consisting of the arduino board and microprocessors along with all the other circuitry additionally contributed to the cost.

Material	Quantity	Independent Vendor	Cost
17HS13 Motors	10	Yes	~$300
A4988 Stepper Motor Drivers	10	Yes	~$13
Braided Nylon Fishing Line	1	Yes	~$12
Aluminum Extrusion (Varying Sizes)	19	No	Not Applicable
Cable Nail-in Clips	12	Yes	~$8
Terminal Blocks	12	Yes	~$120

Table 10: Cost of Materials Used to Modify the Rig

The projected cost of the model is about $453 USD. While initially costly, when compared to current commercially available educational models, the model becomes far more reasonable in price as most other models range from around $300 to $900, with varying features included within the product. Most models on the lower end do not include any joints to display movement, rather they are entirely static. Many models on the upper end of the range include very detailed depictions of muscles and ligaments, but still do not display movement.

6.0 Conclusions and Future Work

The goal of this project was to demonstrate accurate scapulohumeral rhythm by revising and improving upon the previous team's iterations. To accomplish this, the team stripped the model of the restrictive soft tissue materials and constructed a surrounding cage where motors would deliver force upon the humerus and scapula to direct its movements to replicate the scapulohumeral rhythm during abduction and flexion of the humerus. The team then wrote MatLab scripts to direct movements of the motors to produce our desired motion. Although time constraints prevented further testing of the model and developments of the system, the modifications our team made to the model present potential for a fully realized system that replicates proper anatomical movements.

With the aforementioned time constraints restricting the progress of the model, our team believes that future teams can complete the construction of the rig and upon discovery of any potential errors, can continue to develop it. Of the potential foci future teams can follow, we suggest to first continue development of the current model, finalizing its testing to provide experimental results to supplement theoretical calculations. Additionally, altering the power source for the model from batteries to outlets to avoid repeated purchases of power sources is heavily suggested. Other avenues of development involve a number of options. The development of specialized printed bones to replicate medical conditions such as arthritis would allow for easily swappable parts to bolster the versatility of the model. Adjustments to the model to account for the removed materials from the 2022-2023 team could similarly be made to allow for the biological materials that the 2022-2023 team sought to replicate in their model.

With a fully realized rig and modifications made to the power source, this model has incredible potential as an educational tool. Medical experts that have a deeper understanding of injuries of any type, are capable of treating a greater number of patients. The value in providing medical students a deeper and more thorough understanding of the human shoulder cannot be understated.

7.0 Appendix: MatLab Code

Provided below is the complete code written by our team. The team lead for this portion of the project, as previously stated, was Cameron Leffler

setup.m abCode.m motortest2.m

```
global currentAbIndex;
global currentFlexIndex;
global abMatrix;
global flexMatrix;

global xCoordsPositionMatrixAb;
global yCoordsPositionMatrixAb;
global zCoordsPositionMatrixAb;
global FMasterXAb;
global FMasterYAb;
global FMasterZAb;

global xCoordsPositionMatrixFlex;
global yCoordsPositionMatrixFlex;
global zCoordsPositionMatrixFlex;
global FMasterXFlex;
global FMasterYFlex;
global FMasterZFlex;

global masterPointAAb
global masterPointBAb
global masterPointCAb
global masterPointDAb
global masterPointEAb
global masterPointFAb

global masterPointAFlex
global masterPointBFlex
global masterPointCFlex
global masterPointDFlex
global masterPointEFlex
global masterPointFFlex

global motorA
global motorB
```

```
global motorCPosY
global motorCNegY
global motorCX
global motorD
global motorE
global motorFZ
global motorFPosY
global motorFNegY

currentFlexIndex = 1;
currentAbIndex = 1;

%%%Initial Positions
A = [0;0;0];
B = [0.11; 0.035; 0.013];
C = [0.03;-0.004;-0.141];
D = [-0.015;0;-0.03];
E = [0.07;0;-0.03];
F = [0.11;0.035;-0.3696];
G = B;
I = [0.01; 0; -0.01];
J = [0.12; 0; -0.01];
K = [0.01; 0; -0.15];

motorA = [A(1); A(2); 0.38203];
motorB = [B(1); B(2); 0.38203];
motorCPosY = [C(1); 0.39; C(3)];
motorCNegY = [C(1); -0.39; C(3)];
motorCX = [0.474; C(2); 0];
motorD = [D(1); -0.39; D(3)];
motorE = [E(1); 0.39; E(3)];
motorFZ = [0.237; 0; 0.38203];
motorFPosY = [0.237; 0.39; 0];
motorFNegY = [0.237; -0.39; 0];
```

```
%%%%%%%%%% Abduction

%Other starting conditions
armAngleAbduction = 0;
increaseAbA = 0.1;
radIncreaseAbA = deg2rad(increaseAbA);
armAngleAbductionMatrix = [armAngleAbduction;0];

masterPositionMatrixAb = [A,B,C,D,E,I,J,K];
xCoordsPositionMatrixAb = [0,0,0,0,0];
yCoordsPositionMatrixAb = [0,0,0,0,0];
zCoordsPositionMatrixAb = [0,0,0,0,0];
FMasterAb = [0.11;0.035;-0.3696];
FMasterXAb = [0;0];
FMasterYAb = [0;0];
FMasterZAb = [0;0];

wireLengthMatrixAb = allWireLength(motorA,motorB,motorCPosY,motorCNegY,...
    motorCX,motorD,motorE,motorFPosY,motorFNegY,motorFZ,A,B,C,D,E,F);
threadLengthAMatrixAb = [0;0];
threadLengthBMatrixAb = [0;0];
threadLengthCPosYMatrixAb = [0;0];
threadLengthCNegYMatrixAb = [0;0];
threadLengthCXMatrixAb = [0;0];
threadLengthDMatrixAb = [0;0];
threadLengthEMatrixAb = [0;0];
threadLengthFPosYMatrixAb = [0;0];
threadLengthFNegYMatrixAb = [0;0];
threadLengthFZMatrixAb = [0;0];

counter = 4;
i = 1;

while armAngleAbduction <= 120
```

```
%%%Set correct angles or rotation
if armAngleAbduction <= 30
    %SA angles for 0-30 degrees of AA
    upwardRotation = deg2rad(-3.39);
    anteriorTilt = deg2rad(6.04);
    externalRotation = deg2rad(1.29);

    upwardRotationStep = upwardRotation/(30/increaseAbA);
    anteriorTiltStep = anteriorTilt/(30/increaseAbA);
    externalRotationStep = externalRotation/(30/increaseAbA);

    ICR = A;
end

if armAngleAbduction > 30 && armAngleAbduction <= 60
    %SA angles for 30-60 degrees of AA
    upwardRotation = deg2rad(-26.34)-deg2rad(-3.39);
    anteriorTilt = deg2rad(22.28)-deg2rad(6.04);
    externalRotation = deg2rad(5.93)-deg2rad(1.29);

    upwardRotationStep = upwardRotation/(30/increaseAbA);
    anteriorTiltStep = anteriorTilt/(30/increaseAbA);
    externalRotationStep = externalRotation/(30/increaseAbA);

    ICR = A;
end

if armAngleAbduction > 60 && armAngleAbduction <= 90
    %SA angles for 60-90 degrees of AA
    upwardRotation = deg2rad(-41.25)-deg2rad(-26.34);
    anteriorTilt = deg2rad(31.45)-deg2rad(22.28);
    externalRotation = deg2rad(8.92)-deg2rad(5.93);

    upwardRotationStep = upwardRotation/(30/increaseAbA);
    anteriorTiltStep = anteriorTilt/(30/increaseAbA);
```

```
    externalRotationStep = externalRotation/(30/increaseAbA);

    ICR = A;
end

if armAngleAbduction > 90 && armAngleAbduction <= 120
    %SA angles for 90-120 degrees of AA
    upwardRotation = deg2rad(-50.85)-deg2rad(-41.25);
    anteriorTilt = deg2rad(33.49)-deg2rad(31.45);
    externalRotation = deg2rad(10.71)-deg2rad(8.92);

    upwardRotationStep = upwardRotation/(30/increaseAbA);
    anteriorTiltStep = anteriorTilt/(30/increaseAbA);
    externalRotationStep = externalRotation/(30/increaseAbA);

    changeX = B(1) - A(1);
    changeY = B(2) - A(2);
    changeZ = B(3) - A(3);
    ICR = [changeX/2;changeY/2;changeZ/2];
end

BforF = B;

masterPositionMatrixAb(counter:counter+2,:) = fullRotation(A,B,C,D,E,I,...
    J,K,upwardRotationStep,anteriorTiltStep,externalRotationStep,ICR);
FMasterAb(counter:counter+2,1) = humeralAbduction(BforF,B,F,radIncreaseAbA);

A = masterPositionMatrixAb(counter:counter+2,1);
B = masterPositionMatrixAb(counter:counter+2,2);
C = masterPositionMatrixAb(counter:counter+2,3);
D = masterPositionMatrixAb(counter:counter+2,4);
E = masterPositionMatrixAb(counter:counter+2,5);
I = masterPositionMatrixAb(counter:counter+2,6);
J = masterPositionMatrixAb(counter:counter+2,7);
K = masterPositionMatrixAb(counter:counter+2,8);
```

setup.m | abCode.m | motortest2.m

```matlab
    F = FMasterAb(counter:counter+2,1);

    armAngleAbductionMatrix((counter-1)/3) = armAngleAbduction;

    threadLengthAMatrixAb((counter-1)/3) = wireLengthMatrixAb(1,1);
    threadLengthBMatrixAb((counter-1)/3) = wireLengthMatrixAb(1,2);
    threadLengthCPosYMatrixAb((counter-1)/3) = wireLengthMatrixAb(1,3);
    threadLengthCNegYMatrixAb((counter-1)/3) = wireLengthMatrixAb(1,4);
    threadLengthCXMatrixAb((counter-1)/3) = wireLengthMatrixAb(1,5);
    threadLengthDMatrixAb((counter-1)/3) = wireLengthMatrixAb(1,6);
    threadLengthEMatrixAb((counter-1)/3) = wireLengthMatrixAb(1,7);
    threadLengthFPosYMatrixAb((counter-1)/3) = wireLengthMatrixAb(1,8);
    threadLengthFNegYMatrixAb((counter-1)/3) = wireLengthMatrixAb(1,9);
    threadLengthFZMatrixAb((counter-1)/3) = wireLengthMatrixAb(1,10);

    wireLengthMatrixAb = allWireLength(motorA,motorB,motorCPosY,motorCNegY,...
        motorCX,motorD,motorE,motorFPosY,motorFNegY,motorFZ,A,B,C,D,E,F);

    armAngleAbduction = armAngleAbduction + increaseAbA;
    counter = counter + 3;
end

while i < size(masterPositionMatrixAb,1)/3
xCoordsPositionMatrixAb(i,:) = masterPositionMatrixAb(i*3-2,1:5);
yCoordsPositionMatrixAb(i,:) = masterPositionMatrixAb(i*3-1,1:5);
zCoordsPositionMatrixAb(i,:) = masterPositionMatrixAb(i*3,1:5);

i = i+1;
end

i = 1;

while i < size(FMasterAb,1)/3
FMasterXAb(i,:) = FMasterAb(i*3-2,:);
FMasterYAb(i,:) = FMasterAb(i*3-1,:);
```

setup.m | abCode.m | motortest2.m

```matlab
FMasterZAb(i,:) = FMasterAb(i*3,:);

i = i+1;
end


abMatrix = [threadLengthAMatrixAb,threadLengthBMatrixAb,threadLengthCPosYMatrixAb,...
    threadLengthCNegYMatrixAb, threadLengthDMatrixAb,threadLengthEMatrixAb,...
    threadLengthFPosYMatrixAb,threadLengthFNegYMatrixAb,threadLengthFZMatrixAb];


%%% Reset Initial Positions
A = [0;0;0];
B = [0.11; 0.035; 0.013];
C = [0.03;-0.004;-0.141];
D = [-0.015;0;-0.03];
E = [0.07;0;-0.03];
F = [0.11;0.035;-0.3696];
I = [0.01; 0; -0.01];
J = [0.12; 0; -0.01];
K = [0.01; 0; -0.15];

%%%%%%%%%% Flexion Time

wireLengthMatrixFlex = allWireLength(motorA,motorB,motorCPosY,motorCNegY,...
    motorCX,motorD,motorE,motorFPosY,motorFNegY,motorFZ,A,B,C,D,E,F);

% Other starting conditions
armAngleFlex = 0;
increaseFlexA = 0.1;
radIncreaseFlexA = deg2rad(increaseFlexA);
armAngleFlexMatrix = [armAngleFlex;0];

masterPositionMatrixFlex = [A,B,C,D,E,I,J,K];
xCoordsPositionMatrixFlex = [0,0,0,0,0];
```

setup.m | abCode.m | motortest2.m

```
yCoordsPositionMatrixFlex = [0,0,0,0,0];
zCoordsPositionMatrixFlex = [0,0,0,0,0];
FMasterFlex = [0.11;0.035;-0.3696];
FMasterXFlex = [0;0];
FMasterYFlex = [0;0];
FMasterZFlex = [0;0];

threadLengthAMatrixFlex = [0;0];
threadLengthBMatrixFlex = [0;0];
threadLengthCPosYMatrixFlex = [0;0];
threadLengthCNegYMatrixFlex = [0;0];
threadLengthCXMatrixFlex = [0;0];
threadLengthDMatrixFlex = [0;0];
threadLengthEMatrixFlex = [0;0];
threadLengthFPosYMatrixFlex = [0;0];
threadLengthFNegYMatrixFlex = [0;0];
threadLengthFZMatrixFlex = [0;0];

counter = 4;
i = 1;

while armAngleFlex <= 120


    %%%Set correct angles or rotation
    if armAngleFlex <= 30
        %SA angles for 0-30 degrees of AA
        upwardRotation = deg2rad(-1.55);
        anteriorTilt = deg2rad(-1.82);
        externalRotation = deg2rad(-1.13);

        upwardRotationStep = upwardRotation/(30/increaseFlexA);
        anteriorTiltStep = anteriorTilt/(30/increaseFlexA);
        externalRotationStep = externalRotation/(30/increaseFlexA);

        ICR = A;
```

setup.m | abCode.m | motortest2.m

```
        ICR = A;
    end

    if armAngleFlex > 30 && armAngleFlex <= 60
        %SA angles for 30-60 degrees of AA
        upwardRotation = deg2rad(-9.14)-deg2rad(-1.55);
        anteriorTilt = deg2rad(-0.08)-deg2rad(-1.82);
        externalRotation = deg2rad(-1.93)-deg2rad(-1.13);

        upwardRotationStep = upwardRotation/(30/increaseFlexA);
        anteriorTiltStep = anteriorTilt/(30/increaseFlexA);
        externalRotationStep = externalRotation/(30/increaseFlexA);

        ICR = A;
    end

    if armAngleFlex > 60 && armAngleFlex <= 90
        %SA angles for 60-90 degrees of AA
        upwardRotation = deg2rad(-15.03)-deg2rad(-9.14);
        anteriorTilt = deg2rad(3.80)-deg2rad(-0.08);
        externalRotation = deg2rad(-1.95)-deg2rad(-1.93);

        upwardRotationStep = upwardRotation/(30/increaseFlexA);
        anteriorTiltStep = anteriorTilt/(30/increaseFlexA);
        externalRotationStep = externalRotation/(30/increaseFlexA);

        ICR = A;
    end

    if armAngleFlex > 90 && armAngleFlex <= 120
        %SA angles for 90-120 degrees of AA
        upwardRotation = deg2rad(-26.35)-deg2rad(-15.03);
        anteriorTilt = deg2rad(11.16)-deg2rad(3.80);
        externalRotation = deg2rad(0.22)-deg2rad(-1.95);

```

setup.m | abCode.m | motortest2.m

```
        upwardRotationStep = upwardRotation/(30/increaseFlexA);
        anteriorTiltStep = anteriorTilt/(30/increaseFlexA);
        externalRotationStep = externalRotation/(30/increaseFlexA);

        changeX = B(1) - A(1);
        changeY = B(2) - A(2);
        changeZ = B(3) - A(3);
        ICR = [changeX/2;changeY/2;changeZ/2];
    end

    BforF = B;

    masterPositionMatrixFlex(counter:counter+2,:) = fullRotation(A,B,C,D,E,...
        I,J,K,upwardRotationStep,anteriorTiltStep,externalRotationStep,ICR);
    FMasterFlex(counter:counter+2,1) = humeralFlexion(BforF,B,F,radIncreaseFlexA);

    A = masterPositionMatrixFlex(counter:counter+2,1);
    B = masterPositionMatrixFlex(counter:counter+2,2);
    C = masterPositionMatrixFlex(counter:counter+2,3);
    D = masterPositionMatrixFlex(counter:counter+2,4);
    E = masterPositionMatrixFlex(counter:counter+2,5);
    I = masterPositionMatrixFlex(counter:counter+2,6);
    J = masterPositionMatrixFlex(counter:counter+2,7);
    K = masterPositionMatrixFlex(counter:counter+2,8);
    F = FMasterFlex(counter:counter+2,1);

    armAngleFlexMatrix((counter-1)/3) = armAngleFlex;

    threadLengthAMatrixFlex((counter-1)/3) = wireLengthMatrixFlex(1,1);
    threadLengthBMatrixFlex((counter-1)/3) = wireLengthMatrixFlex(1,2);
    threadLengthCPosYMatrixFlex((counter-1)/3) = wireLengthMatrixFlex(1,3);
    threadLengthCNegYMatrixFlex((counter-1)/3) = wireLengthMatrixFlex(1,4);
    threadLengthCXMatrixFlex((counter-1)/3) = wireLengthMatrixFlex(1,5);
    threadLengthDMatrixFlex((counter-1)/3) = wireLengthMatrixFlex(1,6);
    threadLengthEMatrixFlex((counter-1)/3) = wireLengthMatrixFlex(1,7);
```

setup.m abCode.m motortest2.m

```
        threadLengthFPosYMatrixFlex((counter-1)/3) = wireLengthMatrixFlex(1,8);
        threadLengthFNegYMatrixFlex((counter-1)/3) = wireLengthMatrixFlex(1,9);
        threadLengthFZMatrixFlex((counter-1)/3) = wireLengthMatrixFlex(1,10);

        wireLengthMatrixFlex = allWireLength(motorA,motorB,motorCPosY,motorCNegY,...
            motorCX,motorD,motorE,motorFPosY,motorFNegY,motorFZ,A,B,C,D,E,F);

        armAngleFlex = armAngleFlex + increaseFlexA;
        counter = counter + 3;
    end

    while i < size(masterPositionMatrixFlex,1)/3
    xCoordsPositionMatrixFlex(i,:) = masterPositionMatrixFlex(i*3-2,1:5);
    yCoordsPositionMatrixFlex(i,:) = masterPositionMatrixFlex(i*3-1,1:5);
    zCoordsPositionMatrixFlex(i,:) = masterPositionMatrixFlex(i*3,1:5);

    i = i+1;
    end

    i = 1;

    while i < size(FMasterFlex,1)/3
    FMasterXFlex(i,:) = FMasterFlex(i*3-2,:);
    FMasterYFlex(i,:) = FMasterFlex(i*3-1,:);
    FMasterZFlex(i,:) = FMasterFlex(i*3,:);

    i = i+1;
    end

    flexMatrix = [threadLengthAMatrixFlex,threadLengthBMatrixFlex,...
        threadLengthCPosYMatrixFlex, threadLengthCNegYMatrixFlex, threadLengthDMatrixFlex,...
        threadLengthEMatrixFlex, threadLengthFPosYMatrixFlex,threadLengthFNegYMatrixFlex,...
        threadLengthFZMatrixFlex];



    masterPointAAb = [xCoordsPositionMatrixAb(:,1),yCoordsPositionMatrixAb(:,1),zCoordsPositionMatrixAb(:,1)];
    masterPointBAb = [xCoordsPositionMatrixAb(:,2),yCoordsPositionMatrixAb(:,2),zCoordsPositionMatrixAb(:,2)];
    masterPointCAb = [xCoordsPositionMatrixAb(:,3),yCoordsPositionMatrixAb(:,3),zCoordsPositionMatrixAb(:,3)];
    masterPointDAb = [xCoordsPositionMatrixAb(:,4),yCoordsPositionMatrixAb(:,4),zCoordsPositionMatrixAb(:,4)];
    masterPointEAb = [xCoordsPositionMatrixAb(:,5),yCoordsPositionMatrixAb(:,5),zCoordsPositionMatrixAb(:,5)];
    masterPointFAb = [FMasterXAb(:,1),FMasterYAb(:,1),FMasterZAb(:,1)];

    masterPointAFlex = [xCoordsPositionMatrixFlex(:,1),yCoordsPositionMatrixFlex(:,1),zCoordsPositionMatrixFlex(:,1)];
    masterPointBFlex = [xCoordsPositionMatrixFlex(:,2),yCoordsPositionMatrixFlex(:,2),zCoordsPositionMatrixFlex(:,2)];
    masterPointCFlex = [xCoordsPositionMatrixFlex(:,3),yCoordsPositionMatrixFlex(:,3),zCoordsPositionMatrixFlex(:,3)];
    masterPointDFlex = [xCoordsPositionMatrixFlex(:,4),yCoordsPositionMatrixFlex(:,4),zCoordsPositionMatrixFlex(:,4)];
    masterPointEFlex = [xCoordsPositionMatrixFlex(:,5),yCoordsPositionMatrixFlex(:,5),zCoordsPositionMatrixFlex(:,5)];
    masterPointFFlex = [FMasterXFlex(:,1),FMasterYFlex(:,1),FMasterZFlex(:,1)];
```

Editor - C:\Users\grayr\OneDrive\Documents\MATLAB\mqp\rotation.m *

+2 | motortest2.m | allWireLength.m | fullRotation.m | humeralAbduction.m | humeralFlexion.m | rotation.m *

```
function [outputVector] = rotation(inputVector,thetaICR,thetaIJ,thetaIK,I,J,K,ICR)
%ROTATION Finds where a point will be after a rotation around all three
%axes

%moving orgin to I
ICenteredI = [0;0;0];
ICenteredJ = [J(1)-I(1);J(2)-I(2);J(3)-I(3)];
ICenteredK = [K(1)-I(1);K(2)-I(2);K(3)-I(3)];
ICenteredInputVector = [inputVector(1)-I(1);inputVector(2)-I(2);inputVector(3)-I(3)];

%Finding axis IJ and IK
IJ = ICenteredJ - ICenteredI;
magIJ = sqrt((IJ(1))^2+(IJ(2))^2+(IJ(3))^2);
normalIJ = IJ/magIJ;

IK = ICenteredI - ICenteredK;
magIK = sqrt((IK(1))^2+(IK(2))^2+(IK(3))^2);
normalIK = IK/magIK;

%Rotation around IJ
dotIJ = dot(ICenteredInputVector,normalIJ);
crossIJ = cross(normalIJ,ICenteredInputVector);

ICenteredMidVector1 = ((1-cos(thetaIJ))*(dotIJ)*(normalIJ))+((cos(thetaIJ))*...
    (ICenteredInputVector))+((sin(thetaIJ))*(crossIJ));
midVector1 = [ICenteredMidVector1(1)+I(1);ICenteredMidVector1(2)+I(2);ICenteredMidVector1(3)+I(3);];

%Rotation around JK
dotIK = dot(ICenteredInputVector,normalIK);
crossIK = cross(normalIK,ICenteredInputVector);

ICenteredMidVector2 = ((1-cos(thetaIK))*(dotIK)*(normalIK))+((cos(thetaIK))*...
    (ICenteredInputVector))+((sin(thetaIK))*(crossIK));
midVector2 = [ICenteredMidVector2(1)+I(1);ICenteredMidVector2(2)+I(2);ICenteredMidVector2(3)+I(3);];

%Moving Orgin to ICR
ICRCenteredJ = [J(1)-ICR(1);J(2)-ICR(2);J(3)-ICR(3)];
ICRCenteredK = [K(1)-ICR(1);K(2)-ICR(2);K(3)-ICR(3)];
ICRCenteredICR = [0;0;0];
ICRCenteredInputVector = [inputVector(1)-ICR(1);inputVector(2)-ICR(2);inputVector(3)-ICR(3)];

%Making ICR Axis
ICRJ = ICRCenteredJ - ICRCenteredICR;
ICRK = ICRCenteredK - ICRCenteredICR;
axisICR = cross(ICRJ,ICRK);
magICR = sqrt((axisICR(1))^2+(axisICR(2))^2+(axisICR(3))^2);
normalICR = axisICR/magICR;

%Rotation around ICR
dotICR = dot(ICRCenteredInputVector,normalICR);
crossICR = cross(normalICR, ICRCenteredInputVector);

ICRCenteredMidVector3 = ((1-cos(thetaICR))*(dotICR)*(normalICR))+((cos(thetaICR))...
    *(ICRCenteredInputVector))+((sin(thetaICR))*(crossICR));
midVector3 = [ICRCenteredMidVector3(1)+ICR(1);ICRCenteredMidVector3(2)+ICR(2);ICRCenteredMidVector3(3)+ICR(3)];

%Calc the output
outputVector = (midVector1+midVector2+midVector3)/3;

end
```

+2 | abCode.m | motortest2.m | allWireLength.m | fullRotation.m | humeralAbduction.m | humeralFlexion.m *

```
function [FPrime] = humeralFlexion(B,BPrime,F,armAngleChange)
%humeralFlexion finds the position of the bottom of the humerus

BCenteredF = [F(1)-B(1);F(2)-B(2);F(3)-B(3)];

normalAxisofRotation = [1;0;0];

%Rotation around B
dotHumeral = dot(BCenteredF,normalAxisofRotation);
crossHumeral = cross(normalAxisofRotation,BCenteredF);

BCenteredMidVector = ((1-cos(armAngleChange))*(dotHumeral)*(normalAxisofRotation))+...
    ((cos(armAngleChange))*(BCenteredF))+((sin(armAngleChange))*(crossHumeral));


midVector = [0.11; BCenteredMidVector(2)+B(2);BCenteredMidVector(3)+B(3)];
midVector2 = [0.11; midVector(2) + (B(2)-BPrime(2)); midVector(3) + (B(3)-BPrime(3))];

circleCenter = [0.11; BPrime(2); BPrime(3)];
radiusOfCircle = sqrt(abs(((0.3683)^2)-((BPrime(1)-0.11)^2)));

center2F = midVector2 - circleCenter;
magCenter2F = sqrt(center2F(2)^2+center2F(3)^2);
normalCenter2F = center2F/magCenter2F;
FPrime = [0.11;circleCenter(2)+(radiusOfCircle*normalCenter2F(2));circleCenter(3)+(radiusOfCircle*normalCenter2F(3))];

end
```

setup.m | abCode.m | motortest2.m | allWireLength.m | fullRotation.m | humeralAbduction.m *

```
function [FPrime] = humeralAbduction(B,BPrime,F,armAngleChange)
%humeralAbduction finds the position of the bottom of the humerus

BCenteredF = [F(1)-B(1);F(2)-B(2);F(3)-B(3)];

normalAxisofRotation = [0;-1;0];

%Rotation around B
dotHumeral = dot(BCenteredF,normalAxisofRotation);
crossHumeral = cross(normalAxisofRotation,BCenteredF);

BCenteredMidVector = ((1-cos(armAngleChange))*(dotHumeral)*(normalAxisofRotation))+...
    ((cos(armAngleChange))*(BCenteredF))+((sin(armAngleChange))*(crossHumeral));


midVector = [BCenteredMidVector(1)+B(1);0.035;BCenteredMidVector(3)+B(3)];
midVector2 = [midVector(1) + (B(1)-BPrime(1)); 0.035; midVector(3) + (B(3)-BPrime(3))];

circleCenter = [BPrime(1); 0.035; BPrime(3)];
radiusOfCircle = sqrt(((0.3683)^2)-((BPrime(2)-0.035)^2));

center2F = midVector2 - circleCenter;
magCenter2F = sqrt(center2F(1)^2+center2F(3)^2);
normalCenter2F = center2F/magCenter2F;
FPrime = [circleCenter(1)+(radiusOfCircle*normalCenter2F(1));0.035;circleCenter(3)+(radiusOfCircle*normalCenter2F(3))];

end
```

setup.m | abCode.m | motortest2.m | allWireLength.m | fullRotation.m

```
function [positionMatrix] = fullRotation(A,B,C,D,E,I,J,K,upwardRotation,anteriorTilt,externalRotation,ICR)
%FULLROTATION rotates all points around the three axes

rotatedA = rotation (A,upwardRotation,anteriorTilt,externalRotation,I,J,K,ICR);
rotatedB = rotation (B,upwardRotation,anteriorTilt,externalRotation,I,J,K,ICR);
rotatedC = rotation (C,upwardRotation,anteriorTilt,externalRotation,I,J,K,ICR);
rotatedD = rotation (D,upwardRotation,anteriorTilt,externalRotation,I,J,K,ICR);
rotatedE = rotation (E,upwardRotation,anteriorTilt,externalRotation,I,J,K,ICR);
rotatedK = rotation (K,upwardRotation,anteriorTilt,externalRotation,I,J,K,ICR);
rotatedJ = rotation (J,upwardRotation,anteriorTilt,externalRotation,I,J,K,ICR);
rotatedI = rotation (I,upwardRotation,anteriorTilt,externalRotation,I,J,K,ICR);

positionMatrix = [rotatedA,rotatedB,rotatedC,rotatedD,rotatedE,rotatedI,rotatedJ,rotatedK];

end
```

```
function [wireLengths] = allWireLength(motorA,motorB,motorCPosY,motorCNegY,motorCX,motorD,motorE,motorFPosY,...
    motorFNegY,motorFZ,A,B,C,D,E,F)
%WIRELENGTH Finds the length of a wire given the motor and the attachment
%point

threadLengthA = sqrt((motorA(1)-A(1))^2+(motorA(2)-A(2))^2+(motorA(3)-A(3))^2);
threadLengthB = sqrt((motorB(1)-B(1))^2+(motorB(2)-B(2))^2+(motorB(3)-B(3))^2);
threadLengthCPosY = sqrt((motorCPosY(1)-C(1))^2+(motorCPosY(2)-C(2))^2+(motorCPosY(3)-C(3))^2);
threadLengthCNegY = sqrt((motorCNegY(1)-C(1))^2+(motorCNegY(2)-C(2))^2+(motorCNegY(3)-C(3))^2);
threadLengthCX = sqrt((motorCX(1)-C(1))^2+(motorCX(2)-C(2))^2+(motorCX(3)-C(3))^2);
threadLengthD = sqrt((motorD(1)-D(1))^2+(motorD(2)-D(2))^2+(motorD(3)-D(3))^2);
threadLengthE = sqrt((motorE(1)-E(1))^2+(motorE(2)-E(2))^2+(motorE(3)-E(3))^2);
threadLengthFPosY = sqrt((motorFPosY(1)-F(1))^2+(motorFPosY(2)-F(2))^2+(motorFPosY(3)-F(3))^2);
threadLengthFNegY = sqrt((motorFNegY(1)-F(1))^2+(motorFNegY(2)-F(2))^2+(motorFNegY(3)-F(3))^2);
threadLengthFZ = sqrt((motorFZ(1)-F(1))^2+(motorFZ(2)-C(2))^2+(motorFZ(3)-F(3))^2);

wireLengths = [threadLengthA,threadLengthB,threadLengthCPosY,threadLengthCNegY,threadLengthCX,threadLengthD,threadLengthE,
    threadLengthFPosY,threadLengthFNegY,threadLengthFZ];
end
```

```
function [currentAbIndex] = abCode(desiredIndex)

global currentAbIndex;

global currentFlexIndex
global abMatrix;

if currentFlexIndex > 1
    flexCode(1)
end

a1 = arduino('COM4', 'Uno');
a2 = arduino('COM3', 'Uno');

pinDirA = "D12";
pinDirB = "D13";
pinDirCPosY = "D8";
pinDirCNegY = "D8";
pinDirCX = "D13";

pinDirD = "D12";
pinDirE = "D4";
pinDirFPosY = "D7";
pinDirFNegY = "D7";
pinDirFZ = "D4";



pinMovA = "D10";
pinMovB = "D11";
pinMovCPosY = "D9";
pinMovCNegY = "D9";
pinMovCX = "D11";

pinMovD = "D10";
pinMovE = "D5";
```

```
pinMovFPosY = "D6";
pinMovFNegY = "D6";
pinMovFZ = "D5";


configurePin(a2,pinDirA,'DigitalOutput')
configurePin(a2,pinDirB,'DigitalOutput')
configurePin(a2,pinDirCPosY,'DigitalOutput')
configurePin(a1,pinDirCNegY,'DigitalOutput')
configurePin(a1,pinDirCX,'DigitalOutput')
configurePin(a1,pinDirD,'DigitalOutput')
configurePin(a2,pinDirE,'DigitalOutput')
configurePin(a1,pinDirFZ,'DigitalOutput')
configurePin(a2,pinDirFPosY,'DigitalOutput')
configurePin(a1,pinDirFNegY,'DigitalOutput')

configurePin(a2,pinMovA,'PWM')
configurePin(a2,pinMovB,'PWM')
configurePin(a2,pinMovCPosY,'PWM')
configurePin(a1,pinMovCNegY,'PWM')
configurePin(a1,pinMovCX,'PWM')
configurePin(a1,pinMovD,'PWM')
configurePin(a2,pinMovE,'PWM')
configurePin(a1,pinMovFPosY,'PWM')
configurePin(a2,pinMovFNegY,'PWM')
configurePin(a1,pinMovFZ,'PWM')



directionPins1 = [pinDirFZ, pinDirCNegY, pinDirFNegY, pinDirD, pinDirCX];
directionPins2 = [pinDirA, pinDirB, pinDirCPosY, pinDirFPosY, pinDirE];


movementPins1 = [pinMovFZ, pinMovCNegY, pinMovFNegY, pinMovD, pinMovCX];
movementPins2 = [pinMovA, pinMovB, pinMovCPosY, pinMovFPosY, pinMovE];
```

```
i = 1;

if currentAbIndex < desiredIndex
    while currentAbIndex < desiredIndex

        while i<=5
            deltaLength = abs(abMatrix(currentAbIndex,i)-abMatrix(currentAbIndex+1,i));

            rotationDuration = deltaLength*116.886;

            if abMatrix(currentAbIndex+1,i) > abMatrix(currentAbIndex,i)
                writeDigitalPin(a1,directionPins1(i),1)
                writeDigitalPin(a2,directionPins2(i),1)
            else
                writeDigitalPin(a1,directionPins1(i),0)
                writeDigitalPin(a2,directionPins2(i),0)
            end

            writePWMDutyCycle(a1,movementPins1(i),0)
            writePWMDutyCycle(a2,movementPins2(i),0)
            minDutyCycle = 0.03;
            maxDutyCycle = 0.12;
            writePWMDutyCycle(a1,movementPins1(i),minDutyCycle)
            writePWMDutyCycle(a2,movementPins2(i),minDutyCycle)

            startTime = tic;

            while toc(startTime) < rotationDuration
                % Adjust the duty cycle gradually to simulate rotation
                currentDutyCycle = minDutyCycle + (toc(startTime) / rotationDuration)...
                    * (maxDutyCycle - minDutyCycle);
                writePWMDutyCycle(a1, movementPins1(i), currentDutyCycle);
                writePWMDutyCycle(a2, movementPins2(i), currentDutyCycle);

                % Add a small delay for smoother motion (adjust as needed)
                pause(0.01);
```

setup.m | abCode.m | motortest2.m

```
            end

            % Stop the servo motor by setting the duty cycle to 0
            writePWMDutyCycle(a1, movementPins1(i), 0);
            writePWMDutyCycle(a2, movementPins2(i), 0);
            i = i+1;
        end
        currentAbIndex = currentAbIndex + 1;
        i = 1;
    end
else
    while currentAbIndex > desiredIndex

        while i<=5
            deltaLength = abs(abMatrix(currentAbIndex,i)-abMatrix(currentAbIndex-1,i));


            rotationDuration = deltaLength*116.886;


            if abMatrix(currentAbIndex-1,i) > abMatrix(currentAbIndex,i)
                writeDigitalPin(a1,directionPins1(i),1)
                writeDigitalPin(a2, directionPins2(i),1)
            else
                writeDigitalPin(a1,directionPins1(i),0)
                writeDigitalPin(a2,directionPins2(i),0)
            end

            writePWMDutyCycle(a1,movementPins1(i),0)
            writePWMDutyCycle(a2,movementPins2(i),0)
            minDutyCycle = 0.03;
            maxDutyCycle = 0.12;
            writePWMDutyCycle(a1,movementPins1(i),minDutyCycle)
            writePWMDutyCycle(a2,movementPins2(i),minDutyCycle)


            startTime = tic;

            while toc(startTime) < rotationDuration
                % Adjust the duty cycle gradually to simulate rotation
                currentDutyCycle = minDutyCycle + (toc(startTime) / rotationDuration)...
                    * (maxDutyCycle - minDutyCycle);
                writePWMDutyCycle(a1, movementPins1(i), currentDutyCycle);
                writePWMDutyCycle(a2, movementPins2(i), currentDutyCycle);

                % Add a small delay for smoother motion (adjust as needed)
                pause(0.01);
            end

            % Stop the servo motor by setting the duty cycle to 0
            writePWMDutyCycle(a1, movementPins1(i), 0);
            writePWMDutyCycle(a2, movementPins2(i), 0);
            i = i+1;
        end
        currentAbIndex = currentAbIndex - 1;
        i = 1;
    end
end
clear a1
clear a2
end
```

```
function [currentFlexIndex] = flexCode(desiredIndex)

global currentFlexIndex;

global currentFlexIndex
global flexMatrix;

if currentFlexIndex > 1
    flexCode(1)
end

a1 = arduino('COM4', 'Uno');
a2 = arduino('COM3', 'Uno');

pinDirA = "D12";
pinDirB = "D13";
pinDirCPosY = "D8";
pinDirCNegY = "D8";
pinDirCX = "D13";

pinDirD = "D12";
pinDirE = "D4";
pinDirFPosY = "D7";
pinDirFNegY = "D7";
pinDirFZ = "D4";



pinMovA = "D10";
pinMovB = "D11";
pinMovCPosY = "D9";
pinMovCNegY = "D9";
pinMovCX = "D11";

pinMovD = "D10";
pinMovE = "D5";
pinMovFPosY = "D6";
pinMovFNegY = "D6";
pinMovFZ = "D5";


configurePin(a2,pinDirA,'DigitalOutput')
configurePin(a2,pinDirB,'DigitalOutput')
configurePin(a2,pinDirCPosY,'DigitalOutput')
configurePin(a1,pinDirCNegY,'DigitalOutput')
configurePin(a1,pinDirCX,'DigitalOutput')
configurePin(a1,pinDirD,'DigitalOutput')
configurePin(a2,pinDirE,'DigitalOutput')
configurePin(a1,pinDirFZ,'DigitalOutput')
configurePin(a2,pinDirFPosY,'DigitalOutput')
configurePin(a1,pinDirFNegY,'DigitalOutput')

configurePin(a2,pinMovA,'PWM')
configurePin(a2,pinMovB,'PWM')
configurePin(a2,pinMovCPosY,'PWM')
configurePin(a1,pinMovCNegY,'PWM')
configurePin(a1,pinMovCX,'PWM')
configurePin(a1,pinMovD,'PWM')
configurePin(a2,pinMovE,'PWM')
configurePin(a1,pinMovFPosY,'PWM')
configurePin(a2,pinMovFNegY,'PWM')
configurePin(a1,pinMovFZ,'PWM')



directionPins1 = [pinDirFZ, pinDirCNegY, pinDirFNegY, pinDirD, pinDirCX];
directionPins2 = [pinDirA, pinDirB, pinDirCPosY, pinDirFPosY, pinDirE];


movementPins1 = [pinMovFZ, pinMovCNegY, pinMovFNegY, pinMovD, pinMovCX];
movementPins2 = [pinMovA, pinMovB, pinMovCPosY, pinMovFPosY, pinMovE];

```

```
i = 1;

if currentFlexIndex < desiredIndex
    while currentFlexIndex < desiredIndex

        while i<=5
            deltaLength = abs(flexMatrix(currentFlexIndex,i)-flexMatrix(currentFlexIndex+1,i));

            rotationDuration = deltaLength*116.886;

            if flexMatrix(currentFlexIndex+1,i) > flexMatrix(currentFlexIndex,i)
                writeDigitalPin(a1,directionPins1(i),1)
                writeDigitalPin(a2,directionPins2(i),1)
            else
                writeDigitalPin(a1,directionPins1(i),0)
                writeDigitalPin(a2,directionPins2(i),0)
            end

            writePWMDutyCycle(a1,movementPins1(i),0)
            writePWMDutyCycle(a2,movementPins2(i),0)
            minDutyCycle = 0.03;
            maxDutyCycle = 0.12;
            writePWMDutyCycle(a1,movementPins1(i),minDutyCycle)
            writePWMDutyCycle(a2,movementPins2(i),minDutyCycle)

            startTime = tic;

            while toc(startTime) < rotationDuration
                % Adjust the duty cycle gradually to simulate rotation
                currentDutyCycle = minDutyCycle + (toc(startTime) / rotationDuration)...
                    * (maxDutyCycle - minDutyCycle);
                writePWMDutyCycle(a1, movementPins1(i), currentDutyCycle);
                writePWMDutyCycle(a2, movementPins2(i), currentDutyCycle);

                % Add a small delay for smoother motion (adjust as needed)
                pause(0.01);
```

setup.m | motortest2.m | flexCode.m | abCode.m | +

```
            end

            % Stop the servo motor by setting the duty cycle to 0
            writePWMDutyCycle(a1, movementPins1(i), 0);
            writePWMDutyCycle(a2, movementPins2(i), 0);
            i = i+1;
        end
        currentFlexIndex = currentFlexIndex + 1;
        i = 1;
    end
else
    while currentFlexIndex > desiredIndex

        while i<=5
            deltaLength = abs(flexMatrix(currentFlexIndex,i)-flexMatrix(currentFlexIndex-1,i));


            rotationDuration = deltaLength*116.886;


            if flexMatrix(currentFlexIndex-1,i) > flexMatrix(currentFlexIndex,i)
                writeDigitalPin(a1,directionPins1(i),1)
                writeDigitalPin(a2, directionPins2(i),1)
            else
                writeDigitalPin(a1,directionPins1(i),0)
                writeDigitalPin(a2,directionPins2(i),0)
            end

            writePWMDutyCycle(a1,movementPins1(i),0)
            writePWMDutyCycle(a2,movementPins2(i),0)
            minDutyCycle = 0.03;
            maxDutyCycle = 0.12;
            writePWMDutyCycle(a1,movementPins1(i),minDutyCycle)
            writePWMDutyCycle(a2,movementPins2(i),minDutyCycle)

```

```
            startTime = tic;

            while toc(startTime) < rotationDuration
                % Adjust the duty cycle gradually to simulate rotation
                currentDutyCycle = minDutyCycle + (toc(startTime) / rotationDuration)...
                    * (maxDutyCycle - minDutyCycle);
                writePWMDutyCycle(a1, movementPins1(i), currentDutyCycle);
                writePWMDutyCycle(a2, movementPins2(i), currentDutyCycle);

                % Add a small delay for smoother motion (adjust as needed)
                pause(0.01);
            end

            % Stop the servo motor by setting the duty cycle to 0
            writePWMDutyCycle(a1, movementPins1(i), 0);
            writePWMDutyCycle(a2, movementPins2(i), 0);
            i = i+1;
        end
        currentFlexIndex = currentFlexIndex - 1;
        i = 1;
    end
end
clear a1
clear a2
end
```

```
classdef shoulderRotationApp < matlab.apps.AppBase

    % Properties that correspond to app components
    properties (Access = public)
        UIFigure                     matlab.ui.Figure
        ShowWiresCheckBox            matlab.ui.control.CheckBox
        HumeralFlexionSlider         matlab.ui.control.Slider
        HumeralFlexionSliderLabel    matlab.ui.control.Label
        HumeralAbductionSlider       matlab.ui.control.Slider
        HumeralAbductionSliderLabel  matlab.ui.control.Label
        UIAxesTop                    matlab.ui.control.UIAxes
        UIAxesSide                   matlab.ui.control.UIAxes
        UIAxesFront                  matlab.ui.control.UIAxes
        UIAxesAngled                 matlab.ui.control.UIAxes
    end

    % Callbacks that handle component events
    methods (Access = private)

        % Code that executes after component creation
        function startupFcn(app)
            setup
            global xCoordsPositionMatrixAb
            global yCoordsPositionMatrixAb
            global zCoordsPositionMatrixAb
            global FMasterXAb
            global FMasterYAb
            global FMasterZAb

            plot3(app.UIAxesAngled,[xCoordsPositionMatrixAb(1,1) xCoordsPositionMatrixAb(1,2) xCoordsPositionMatrixAb(1,5) xCoordsPositionMatrixAb(1,3) xCoordsPositionMatrixAb(1,4) xCoordsPositionMatrixAb(1,1)], ...
            [yCoordsPositionMatrixAb(1,1) yCoordsPositionMatrixAb(1,2) yCoordsPositionMatrixAb(1,5) yCoordsPositionMatrixAb(1,3) yCoordsPositionMatrixAb(1,4) yCoordsPositionMatrixAb(1,1)], ...
            [zCoordsPositionMatrixAb(1,1) zCoordsPositionMatrixAb(1,2) zCoordsPositionMatrixAb(1,5) zCoordsPositionMatrixAb(1,3) zCoordsPositionMatrixAb(1,4) zCoordsPositionMatrixAb(1,1)],'.k-', ...
            [xCoordsPositionMatrixAb(5,2) FMasterXAb(1)], [yCoordsPositionMatrixAb(5,2) FMasterYAb(1)], [zCoordsPositionMatrixAb(5,2) FMasterZAb(1)], '.k-');


            plot3(app.UIAxesFront,[xCoordsPositionMatrixAb(1,1) xCoordsPositionMatrixAb(1,2) xCoordsPositionMatrixAb(1,5) xCoordsPositionMatrixAb(1,3) xCoordsPositionMatrixAb(1,4) xCoordsPositionMatrixAb(1,1)], ...
            [yCoordsPositionMatrixAb(1,1) yCoordsPositionMatrixAb(1,2) yCoordsPositionMatrixAb(1,5) yCoordsPositionMatrixAb(1,3) yCoordsPositionMatrixAb(1,4) yCoordsPositionMatrixAb(1,1)], ...
            [zCoordsPositionMatrixAb(1,1) zCoordsPositionMatrixAb(1,2) zCoordsPositionMatrixAb(1,5) zCoordsPositionMatrixAb(1,3) zCoordsPositionMatrixAb(1,4) zCoordsPositionMatrixAb(1,1)],'.k-', ...
            [xCoordsPositionMatrixAb(5,2) FMasterXAb(1)], [yCoordsPositionMatrixAb(5,2) FMasterYAb(1)], [zCoordsPositionMatrixAb(5,2) FMasterZAb(1)], '.k-');

            app.UIAxesFront.CameraPosition = [0.05,1,-0.075];
            app.UIAxesFront.CameraTarget = [0.05,0,-0.075];

```

```
            plot3(app.UIAxesSide,[xCoordsPositionMatrixAb(1,1) xCoordsPositionMatrixAb(1,2) xCoordsPositionMatrixAb(1,5) xCoordsPositionMatrixAb(1,3) xCoordsPositionMatrixAb(1,4) xCoordsPositionMatrixAb(1,1)], ...
            [yCoordsPositionMatrixAb(1,1) yCoordsPositionMatrixAb(1,2) yCoordsPositionMatrixAb(1,5) yCoordsPositionMatrixAb(1,3) yCoordsPositionMatrixAb(1,4) yCoordsPositionMatrixAb(1,1)], ...
            [zCoordsPositionMatrixAb(1,1) zCoordsPositionMatrixAb(1,2) zCoordsPositionMatrixAb(1,5) zCoordsPositionMatrixAb(1,3) zCoordsPositionMatrixAb(1,4) zCoordsPositionMatrixAb(1,1)],'.k-', ...
            [xCoordsPositionMatrixAb(5,2) FMasterXAb(1)], [yCoordsPositionMatrixAb(5,2) FMasterYAb(1)], [zCoordsPositionMatrixAb(5,2) FMasterZAb(1)], '.k-');

            app.UIAxesSide.CameraPosition = [-1,0,-0.075];
            app.UIAxesSide.CameraTarget = [0,0,-0.075];

            plot3(app.UIAxesTop,[xCoordsPositionMatrixAb(1,1) xCoordsPositionMatrixAb(1,2) xCoordsPositionMatrixAb(1,5) xCoordsPositionMatrixAb(1,3) xCoordsPositionMatrixAb(1,4) xCoordsPositionMatrixAb(1,1)], ...
            [yCoordsPositionMatrixAb(1,1) yCoordsPositionMatrixAb(1,2) yCoordsPositionMatrixAb(1,5) yCoordsPositionMatrixAb(1,3) yCoordsPositionMatrixAb(1,4) yCoordsPositionMatrixAb(1,1)], ...
            [zCoordsPositionMatrixAb(1,1) zCoordsPositionMatrixAb(1,2) zCoordsPositionMatrixAb(1,5) zCoordsPositionMatrixAb(1,3) zCoordsPositionMatrixAb(1,4) zCoordsPositionMatrixAb(1,1)],'.k-', ...
            [xCoordsPositionMatrixAb(5,2) FMasterXAb(1)], [yCoordsPositionMatrixAb(5,2) FMasterYAb(1)], [zCoordsPositionMatrixAb(5,2) FMasterZAb(1)], '.k-');

            app.UIAxesTop.CameraPosition = [0.05,0,2];
            app.UIAxesTop.CameraTarget = [0.05,0,0];

            global currentAbIndex
            global currentFlexIndex

            if currentAbIndex > 1
                abCode(1)
            elseif currentFlexIndex > 1
                flexCode(1)
            end

        end

        % Value changed function: HumeralAbductionSlider
        function HumeralAbductionSliderValueChanging(app, event)
            global xCoordsPositionMatrixAb
            global yCoordsPositionMatrixAb
            global zCoordsPositionMatrixAb
            global FMasterXAb
            global FMasterYAb
            global FMasterZAb

            global masterPointAAb
            global masterPointBAb
            global masterPointCAb
            global masterPointDAb
            global masterPointEAb
            global masterPointFAb

```

```
            global motorA
            global motorB
            global motorCPosY
            global motorCNegY
            global motorCX
            global motorD
            global motorE
            global motorFZ
            global motorFPosY
            global motorFNegY

            changingValue = event.Value;
            app.HumeralFlexionSlider.Value = 0;
            getPoint = round((round(changingValue,1)/0.1)+1);

            if app.ShowWiresCheckBox.Value == 1
                wireColor = 'r.--';
            elseif app.ShowWiresCheckBox.Value == 0
                wireColor = 'w:square';
            end

            plot3(app.UIAxesAngled,[masterPointAAb(getPoint,1) motorA(1)], [masterPointAAb(getPoint,2) motorA(2)], [masterPointAAb(getPoint,3) motorA(3)], wireColor, ...
            [masterPointBAb(getPoint,1) motorB(1)], [masterPointBAb(getPoint,2) motorB(2)], [masterPointBAb(getPoint,3) motorB(3)], wireColor, ...
            [masterPointCAb(getPoint,1) motorCPosY(1)], [masterPointCAb(getPoint,2) motorCPosY(2)], [masterPointCAb(getPoint,3) motorCPosY(3)], wireColor, ...
            [masterPointCAb(getPoint,1) motorCNegY(1)], [masterPointCAb(getPoint,2) motorCNegY(2)], [masterPointCAb(getPoint,3) motorCNegY(3)], wireColor, ...
            [masterPointCAb(getPoint,1) motorCX(1)], [masterPointCAb(getPoint,2) motorCX(2)], [masterPointCAb(getPoint,3) motorCX(3)], wireColor, ...
            [masterPointDAb(getPoint,1) motorD(1)], [masterPointDAb(getPoint,2) motorD(2)], [masterPointDAb(getPoint,3) motorD(3)], wireColor, ...
            [masterPointEAb(getPoint,1) motorE(1)], [masterPointEAb(getPoint,2) motorE(2)], [masterPointEAb(getPoint,3) motorE(3)], wireColor, ...
            [masterPointFAb(getPoint,1) motorFPosY(1)], [masterPointFAb(getPoint,2) motorFPosY(2)], [masterPointFAb(getPoint,3) motorFPosY(3)], wireColor, ...
            [masterPointFAb(getPoint,1) motorFNegY(1)], [masterPointFAb(getPoint,2) motorFNegY(2)], [masterPointFAb(getPoint,3) motorFNegY(3)], wireColor, ...
            [masterPointFAb(getPoint,1) motorFZ(1)], [masterPointFAb(getPoint,2) motorFZ(2)], [masterPointFAb(getPoint,3) motorFZ(3)], wireColor,...
            [xCoordsPositionMatrixAb(getPoint,1) xCoordsPositionMatrixAb(getPoint,2) xCoordsPositionMatrixAb(getPoint,5) xCoordsPositionMatrixAb(getPoint,3) xCoordsPositionMatrixAb(getPoint,4) xCoordsPositionMatrixAb(getPoint,1)], ...
            [yCoordsPositionMatrixAb(getPoint,1) yCoordsPositionMatrixAb(getPoint,2) yCoordsPositionMatrixAb(getPoint,5) yCoordsPositionMatrixAb(getPoint,3) yCoordsPositionMatrixAb(getPoint,4) yCoordsPositionMatrixAb(getPoint,1)], ...
            [zCoordsPositionMatrixAb(getPoint,1) zCoordsPositionMatrixAb(getPoint,2) zCoordsPositionMatrixAb(getPoint,5) zCoordsPositionMatrixAb(getPoint,3) zCoordsPositionMatrixAb(getPoint,4) zCoordsPositionMatrixAb(getPoint,1)],'.k-', ..
            [xCoordsPositionMatrixAb(getPoint,2) FMasterXAb(getPoint,1)], [yCoordsPositionMatrixAb(getPoint,2) FMasterYAb(getPoint,1)], [zCoordsPositionMatrixAb(getPoint,2) FMasterZAb(getPoint,1)], '.k-')


            plot3(app.UIAxesFront,[masterPointAAb(getPoint,1) motorA(1)], [masterPointAAb(getPoint,2) motorA(2)], [masterPointAAb(getPoint,3) motorA(3)], wireColor, ...
            [masterPointBAb(getPoint,1) motorB(1)], [masterPointBAb(getPoint,2) motorB(2)], [masterPointBAb(getPoint,3) motorB(3)], wireColor, ...
            [masterPointCAb(getPoint,1) motorCPosY(1)], [masterPointCAb(getPoint,2) motorCPosY(2)], [masterPointCAb(getPoint,3) motorCPosY(3)], wireColor, ...
            [masterPointCAb(getPoint,1) motorCNegY(1)], [masterPointCAb(getPoint,2) motorCNegY(2)], [masterPointCAb(getPoint,3) motorCNegY(3)], wireColor, ...
            [masterPointCAb(getPoint,1) motorCX(1)], [masterPointCAb(getPoint,2) motorCX(2)], [masterPointCAb(getPoint,3) motorCX(3)], wireColor, ...
            [masterPointDAb(getPoint,1) motorD(1)], [masterPointDAb(getPoint,2) motorD(2)], [masterPointDAb(getPoint,3) motorD(3)], wireColor, ...
```

```
            [masterPointEAb(getPoint,1) motorE(1)], [masterPointEAb(getPoint,2) motorE(2)], [masterPointEAb(getPoint,3) motorE(3)], wireColor, ...
            [masterPointFAb(getPoint,1) motorFPosY(1)], [masterPointFAb(getPoint,2) motorFPosY(2)], [masterPointFAb(getPoint,3) motorFPosY(3)], wireColor, ...
            [masterPointFAb(getPoint,1) motorFNegY(1)], [masterPointFAb(getPoint,2) motorFNegY(2)], [masterPointFAb(getPoint,3) motorFNegY(3)], wireColor, ...
            [masterPointFAb(getPoint,1) motorFZ(1)], [masterPointFAb(getPoint,2) motorFZ(2)], [masterPointFAb(getPoint,3) motorFZ(3)], wireColor,...
            [xCoordsPositionMatrixAb(getPoint,1) xCoordsPositionMatrixAb(getPoint,2) xCoordsPositionMatrixAb(getPoint,5) xCoordsPositionMatrixAb(getPoint,3) xCoordsPositionMatrixAb(getPoint,4) xCoordsPositionMatrixAb(getPoint,1)], ...
            [yCoordsPositionMatrixAb(getPoint,1) yCoordsPositionMatrixAb(getPoint,2) yCoordsPositionMatrixAb(getPoint,5) yCoordsPositionMatrixAb(getPoint,3) yCoordsPositionMatrixAb(getPoint,4) yCoordsPositionMatrixAb(getPoint,1)], ...
            [zCoordsPositionMatrixAb(getPoint,1) zCoordsPositionMatrixAb(getPoint,2) zCoordsPositionMatrixAb(getPoint,5) zCoordsPositionMatrixAb(getPoint,3) zCoordsPositionMatrixAb(getPoint,4) zCoordsPositionMatrixAb(getPoint,1)],'.k-', ..
            [xCoordsPositionMatrixAb(getPoint,2) FMasterXAb(getPoint,1)], [yCoordsPositionMatrixAb(getPoint,2) FMasterYAb(getPoint,1)], [zCoordsPositionMatrixAb(getPoint,2) FMasterZAb(getPoint,1)], '.k-')


            app.UIAxesFront.CameraPosition = [0.05,1,-0.075];
            app.UIAxesFront.CameraTarget = [0.05,0,-0.075];

            plot3(app.UIAxesSide,[masterPointAAb(getPoint,1) motorA(1)], [masterPointAAb(getPoint,2) motorA(2)], [masterPointAAb(getPoint,3) motorA(3)], wireColor, ...
            [masterPointBAb(getPoint,1) motorB(1)], [masterPointBAb(getPoint,2) motorB(2)], [masterPointBAb(getPoint,3) motorB(3)], wireColor, ...
            [masterPointCAb(getPoint,1) motorCPosY(1)], [masterPointCAb(getPoint,2) motorCPosY(2)], [masterPointCAb(getPoint,3) motorCPosY(3)], wireColor, ...
            [masterPointCAb(getPoint,1) motorCNegY(1)], [masterPointCAb(getPoint,2) motorCNegY(2)], [masterPointCAb(getPoint,3) motorCNegY(3)], wireColor, ...
            [masterPointCAb(getPoint,1) motorCX(1)], [masterPointCAb(getPoint,2) motorCX(2)], [masterPointCAb(getPoint,3) motorCX(3)], wireColor, ...
            [masterPointDAb(getPoint,1) motorD(1)], [masterPointDAb(getPoint,2) motorD(2)], [masterPointDAb(getPoint,3) motorD(3)], wireColor, ...
            [masterPointEAb(getPoint,1) motorE(1)], [masterPointEAb(getPoint,2) motorE(2)], [masterPointEAb(getPoint,3) motorE(3)], wireColor, ...
            [masterPointFAb(getPoint,1) motorFPosY(1)], [masterPointFAb(getPoint,2) motorFPosY(2)], [masterPointFAb(getPoint,3) motorFPosY(3)], wireColor, ...
            [masterPointFAb(getPoint,1) motorFNegY(1)], [masterPointFAb(getPoint,2) motorFNegY(2)], [masterPointFAb(getPoint,3) motorFNegY(3)], wireColor, ...
            [masterPointFAb(getPoint,1) motorFZ(1)], [masterPointFAb(getPoint,2) motorFZ(2)], [masterPointFAb(getPoint,3) motorFZ(3)], wireColor,...
            [xCoordsPositionMatrixAb(getPoint,1) xCoordsPositionMatrixAb(getPoint,2) xCoordsPositionMatrixAb(getPoint,5) xCoordsPositionMatrixAb(getPoint,3) xCoordsPositionMatrixAb(getPoint,4) xCoordsPositionMatrixAb(getPoint,1)], ...
            [yCoordsPositionMatrixAb(getPoint,1) yCoordsPositionMatrixAb(getPoint,2) yCoordsPositionMatrixAb(getPoint,5) yCoordsPositionMatrixAb(getPoint,3) yCoordsPositionMatrixAb(getPoint,4) yCoordsPositionMatrixAb(getPoint,1)], ...
            [zCoordsPositionMatrixAb(getPoint,1) zCoordsPositionMatrixAb(getPoint,2) zCoordsPositionMatrixAb(getPoint,5) zCoordsPositionMatrixAb(getPoint,3) zCoordsPositionMatrixAb(getPoint,4) zCoordsPositionMatrixAb(getPoint,1)],'.k-', ..
            [xCoordsPositionMatrixAb(getPoint,2) FMasterXAb(getPoint,1)], [yCoordsPositionMatrixAb(getPoint,2) FMasterYAb(getPoint,1)], [zCoordsPositionMatrixAb(getPoint,2) FMasterZAb(getPoint,1)], '.k-')


            app.UIAxesSide.CameraPosition = [-1,0,-0.075];
            app.UIAxesSide.CameraTarget = [0,0,-0.075];

            plot3(app.UIAxesTop,[masterPointAAb(getPoint,1) motorA(1)], [masterPointAAb(getPoint,2) motorA(2)], [masterPointAAb(getPoint,3) motorA(3)], wireColor, ...
            [masterPointBAb(getPoint,1) motorB(1)], [masterPointBAb(getPoint,2) motorB(2)], [masterPointBAb(getPoint,3) motorB(3)], wireColor, ...
            [masterPointCAb(getPoint,1) motorCPosY(1)], [masterPointCAb(getPoint,2) motorCPosY(2)], [masterPointCAb(getPoint,3) motorCPosY(3)], wireColor, ...
            [masterPointCAb(getPoint,1) motorCNegY(1)], [masterPointCAb(getPoint,2) motorCNegY(2)], [masterPointCAb(getPoint,3) motorCNegY(3)], wireColor, ...
            [masterPointCAb(getPoint,1) motorCX(1)], [masterPointCAb(getPoint,2) motorCX(2)], [masterPointCAb(getPoint,3) motorCX(3)], wireColor, ...
            [masterPointDAb(getPoint,1) motorD(1)], [masterPointDAb(getPoint,2) motorD(2)], [masterPointDAb(getPoint,3) motorD(3)], wireColor, ...
            [masterPointEAb(getPoint,1) motorE(1)], [masterPointEAb(getPoint,2) motorE(2)], [masterPointEAb(getPoint,3) motorE(3)], wireColor, ...
            [masterPointFAb(getPoint,1) motorFPosY(1)], [masterPointFAb(getPoint,2) motorFPosY(2)], [masterPointFAb(getPoint,3) motorFPosY(3)], wireColor, ...
            [masterPointFAb(getPoint,1) motorFNegY(1)], [masterPointFAb(getPoint,2) motorFNegY(2)], [masterPointFAb(getPoint,3) motorFNegY(3)], wireColor, ...
            [masterPointFAb(getPoint,1) motorFZ(1)], [masterPointFAb(getPoint,2) motorFZ(2)], [masterPointFAb(getPoint,3) motorFZ(3)], wireColor,...
            [xCoordsPositionMatrixAb(getPoint,1) xCoordsPositionMatrixAb(getPoint,2) xCoordsPositionMatrixAb(getPoint,5) xCoordsPositionMatrixAb(getPoint,3) xCoordsPositionMatrixAb(getPoint,4) xCoordsPositionMatrixAb(getPoint,1)], ...
```

```
            [yCoordsPositionMatrixAb(getPoint,1) yCoordsPositionMatrixAb(getPoint,2) yCoordsPositionMatrixAb(getPoint,5) yCoordsPositionMatrixAb(getPoint,3) yCoordsPositionMatrixAb(getPoint,4) yCoordsPositionMatrixAb(getPoint,1)], ...
            [zCoordsPositionMatrixAb(getPoint,1) zCoordsPositionMatrixAb(getPoint,2) zCoordsPositionMatrixAb(getPoint,5) zCoordsPositionMatrixAb(getPoint,3) zCoordsPositionMatrixAb(getPoint,4) zCoordsPositionMatrixAb(getPoint,1)],'.k-', ..
            [xCoordsPositionMatrixAb(getPoint,2) FMasterXAb(getPoint,1)], [yCoordsPositionMatrixAb(getPoint,2) FMasterYAb(getPoint,1)], [zCoordsPositionMatrixAb(getPoint,2) FMasterZAb(getPoint,1)], '.k-')


            app.UIAxesTop.CameraPosition = [0.05,0,1];
            app.UIAxesTop.CameraTarget = [0.05,0,0];


            global currentFlexIndex

            if currentFlexIndex > 1
                flexCode(1)
                abCode(getPoint(1))
            else
                abCode(getPoint(1))
            end
        end

        % Value changed function: HumeralFlexionSlider
        function HumeralFlexionSliderValueChanging(app, event)

            global xCoordsPositionMatrixFlex
            global yCoordsPositionMatrixFlex
            global zCoordsPositionMatrixFlex
            global FMasterXFlex
            global FMasterYFlex
            global FMasterZFlex


            global masterPointAFlex
            global masterPointBFlex
            global masterPointCFlex
            global masterPointDFlex
            global masterPointEFlex
            global masterPointFFlex


            global motorA
            global motorB
            global motorCPosY
            global motorCNegY
            global motorCX
```

```
global motorD
global motorE
global motorFZ
global motorFPosY
global motorFNegY

changingValue = event.Value;
app.HumeralAbductionSlider.Value = 0;
getPoint = round((round(changingValue,1)/0.1)+1);

if app.ShowWiresCheckBox.Value == 1
    wireColor = 'r.--';
elseif app.ShowWiresCheckBox.Value == 0
    wireColor = 'w:square';
end


plot3(app.UIAxesAngled,[masterPointAFlex(getPoint,1) motorA(1)], [masterPointAFlex(getPoint,2) motorA(2)], [masterPointAFlex(getPoint,3) motorA(3)], wireColor, ...
[masterPointBFlex(getPoint,1) motorB(1)], [masterPointBFlex(getPoint,2) motorB(2)], [masterPointBFlex(getPoint,3) motorB(3)], wireColor, ...
[masterPointCFlex(getPoint,1) motorCPosY(1)], [masterPointCFlex(getPoint,2) motorCPosY(2)], [masterPointCFlex(getPoint,3) motorCPosY(3)], wireColor, ...
[masterPointCFlex(getPoint,1) motorCNegY(1)], [masterPointCFlex(getPoint,2) motorCNegY(2)], [masterPointCFlex(getPoint,3) motorCNegY(3)], wireColor, ...
[masterPointCFlex(getPoint,1) motorCX(1)], [masterPointCFlex(getPoint,2) motorCX(2)], [masterPointCFlex(getPoint,3) motorCX(3)], wireColor, ...
[masterPointDFlex(getPoint,1) motorD(1)], [masterPointDFlex(getPoint,2) motorD(2)], [masterPointDFlex(getPoint,3) motorD(3)], wireColor, ...
[masterPointEFlex(getPoint,1) motorE(1)], [masterPointEFlex(getPoint,2) motorE(2)], [masterPointEFlex(getPoint,3) motorE(3)], wireColor, ...
[masterPointFFlex(getPoint,1) motorFPosY(1)], [masterPointFFlex(getPoint,2) motorFPosY(2)], [masterPointFFlex(getPoint,3) motorFPosY(3)], wireColor, ...
[masterPointFFlex(getPoint,1) motorFNegY(1)], [masterPointFFlex(getPoint,2) motorFNegY(2)], [masterPointFFlex(getPoint,3) motorFNegY(3)], wireColor, ...
[masterPointFFlex(getPoint,1) motorFZ(1)], [masterPointFFlex(getPoint,2) motorFZ(2)], [masterPointFFlex(getPoint,3) motorFZ(3)], wireColor,...
[xCoordsPositionMatrixFlex(getPoint,1) xCoordsPositionMatrixFlex(getPoint,2) xCoordsPositionMatrixFlex(getPoint,5) xCoordsPositionMatrixFlex(getPoint,3) xCoordsPositionMatrixFlex(getPoint,4) xCoordsPositionMatrixFlex(getPoint,1)], ...
[yCoordsPositionMatrixFlex(getPoint,1) yCoordsPositionMatrixFlex(getPoint,2) yCoordsPositionMatrixFlex(getPoint,5) yCoordsPositionMatrixFlex(getPoint,3) yCoordsPositionMatrixFlex(getPoint,4) yCoordsPositionMatrixFlex(getPoint,1)], ...
[zCoordsPositionMatrixFlex(getPoint,1) zCoordsPositionMatrixFlex(getPoint,2) zCoordsPositionMatrixFlex(getPoint,5) zCoordsPositionMatrixFlex(getPoint,3) zCoordsPositionMatrixFlex(getPoint,4) zCoordsPositionMatrixFlex(getPoint,1)],'.k-', ...
[xCoordsPositionMatrixFlex(getPoint,2) FMasterXFlex(getPoint,1)], [yCoordsPositionMatrixFlex(getPoint,2) FMasterYFlex(getPoint,1)], [zCoordsPositionMatrixFlex(getPoint,2) FMasterZFlex(getPoint,1)], '.k-')


plot3(app.UIAxesFront,[masterPointAFlex(getPoint,1) motorA(1)], [masterPointAFlex(getPoint,2) motorA(2)], [masterPointAFlex(getPoint,3) motorA(3)], wireColor, ...
[masterPointBFlex(getPoint,1) motorB(1)], [masterPointBFlex(getPoint,2) motorB(2)], [masterPointBFlex(getPoint,3) motorB(3)], wireColor, ...
[masterPointCFlex(getPoint,1) motorCPosY(1)], [masterPointCFlex(getPoint,2) motorCPosY(2)], [masterPointCFlex(getPoint,3) motorCPosY(3)], wireColor, ...
[masterPointCFlex(getPoint,1) motorCNegY(1)], [masterPointCFlex(getPoint,2) motorCNegY(2)], [masterPointCFlex(getPoint,3) motorCNegY(3)], wireColor, ...
[masterPointCFlex(getPoint,1) motorCX(1)], [masterPointCFlex(getPoint,2) motorCX(2)], [masterPointCFlex(getPoint,3) motorCX(3)], wireColor, ...
[masterPointDFlex(getPoint,1) motorD(1)], [masterPointDFlex(getPoint,2) motorD(2)], [masterPointDFlex(getPoint,3) motorD(3)], wireColor, ...
[masterPointEFlex(getPoint,1) motorE(1)], [masterPointEFlex(getPoint,2) motorE(2)], [masterPointEFlex(getPoint,3) motorE(3)], wireColor, ...
[masterPointFFlex(getPoint,1) motorFPosY(1)], [masterPointFFlex(getPoint,2) motorFPosY(2)], [masterPointFFlex(getPoint,3) motorFPosY(3)], wireColor, ...
[masterPointFFlex(getPoint,1) motorFNegY(1)], [masterPointFFlex(getPoint,2) motorFNegY(2)], [masterPointFFlex(getPoint,3) motorFNegY(3)], wireColor, ...
[masterPointFFlex(getPoint,1) motorFZ(1)], [masterPointFFlex(getPoint,2) motorFZ(2)], [masterPointFFlex(getPoint,3) motorFZ(3)], wireColor,...
```

```
[xCoordsPositionMatrixFlex(getPoint,1) xCoordsPositionMatrixFlex(getPoint,2) xCoordsPositionMatrixFlex(getPoint,5) xCoordsPositionMatrixFlex(getPoint,3) xCoordsPositionMatrixFlex(getPoint,4) xCoordsPositionMatrixFlex(getPoint,1)], ...
[yCoordsPositionMatrixFlex(getPoint,1) yCoordsPositionMatrixFlex(getPoint,2) yCoordsPositionMatrixFlex(getPoint,5) yCoordsPositionMatrixFlex(getPoint,3) yCoordsPositionMatrixFlex(getPoint,4) yCoordsPositionMatrixFlex(getPoint,1)], ...
[zCoordsPositionMatrixFlex(getPoint,1) zCoordsPositionMatrixFlex(getPoint,2) zCoordsPositionMatrixFlex(getPoint,5) zCoordsPositionMatrixFlex(getPoint,3) zCoordsPositionMatrixFlex(getPoint,4) zCoordsPositionMatrixFlex(getPoint,1)],'.k-', ...
[xCoordsPositionMatrixFlex(getPoint,2) FMasterXFlex(getPoint,1)], [yCoordsPositionMatrixFlex(getPoint,2) FMasterYFlex(getPoint,1)], [zCoordsPositionMatrixFlex(getPoint,2) FMasterZFlex(getPoint,1)], '.k-')

app.UIAxesFront.CameraPosition = [0.05,1,-0.075];
app.UIAxesFront.CameraTarget = [0.05,0,-0.075];

plot3(app.UIAxesSide,[masterPointAFlex(getPoint,1) motorA(1)], [masterPointAFlex(getPoint,2) motorA(2)], [masterPointAFlex(getPoint,3) motorA(3)], wireColor, ...
[masterPointBFlex(getPoint,1) motorB(1)], [masterPointBFlex(getPoint,2) motorB(2)], [masterPointBFlex(getPoint,3) motorB(3)], wireColor, ...
[masterPointCFlex(getPoint,1) motorCPosY(1)], [masterPointCFlex(getPoint,2) motorCPosY(2)], [masterPointCFlex(getPoint,3) motorCPosY(3)], wireColor, ...
[masterPointCFlex(getPoint,1) motorCNegY(1)], [masterPointCFlex(getPoint,2) motorCNegY(2)], [masterPointCFlex(getPoint,3) motorCNegY(3)], wireColor, ...
[masterPointCFlex(getPoint,1) motorCX(1)], [masterPointCFlex(getPoint,2) motorCX(2)], [masterPointCFlex(getPoint,3) motorCX(3)], wireColor, ...
[masterPointDFlex(getPoint,1) motorD(1)], [masterPointDFlex(getPoint,2) motorD(2)], [masterPointDFlex(getPoint,3) motorD(3)], wireColor, ...
[masterPointEFlex(getPoint,1) motorE(1)], [masterPointEFlex(getPoint,2) motorE(2)], [masterPointEFlex(getPoint,3) motorE(3)], wireColor, ...
[masterPointFFlex(getPoint,1) motorFPosY(1)], [masterPointFFlex(getPoint,2) motorFPosY(2)], [masterPointFFlex(getPoint,3) motorFPosY(3)], wireColor, ...
[masterPointFFlex(getPoint,1) motorFNegY(1)], [masterPointFFlex(getPoint,2) motorFNegY(2)], [masterPointFFlex(getPoint,3) motorFNegY(3)], wireColor, ...
[masterPointFFlex(getPoint,1) motorFZ(1)], [masterPointFFlex(getPoint,2) motorFZ(2)], [masterPointFFlex(getPoint,3) motorFZ(3)], wireColor,...
[xCoordsPositionMatrixFlex(getPoint,1) xCoordsPositionMatrixFlex(getPoint,2) xCoordsPositionMatrixFlex(getPoint,5) xCoordsPositionMatrixFlex(getPoint,3) xCoordsPositionMatrixFlex(getPoint,4) xCoordsPositionMatrixFlex(getPoint,1)], ...
[yCoordsPositionMatrixFlex(getPoint,1) yCoordsPositionMatrixFlex(getPoint,2) yCoordsPositionMatrixFlex(getPoint,5) yCoordsPositionMatrixFlex(getPoint,3) yCoordsPositionMatrixFlex(getPoint,4) yCoordsPositionMatrixFlex(getPoint,1)], ...
[zCoordsPositionMatrixFlex(getPoint,1) zCoordsPositionMatrixFlex(getPoint,2) zCoordsPositionMatrixFlex(getPoint,5) zCoordsPositionMatrixFlex(getPoint,3) zCoordsPositionMatrixFlex(getPoint,4) zCoordsPositionMatrixFlex(getPoint,1)],'.k-', ...
[xCoordsPositionMatrixFlex(getPoint,2) FMasterXFlex(getPoint,1)], [yCoordsPositionMatrixFlex(getPoint,2) FMasterYFlex(getPoint,1)], [zCoordsPositionMatrixFlex(getPoint,2) FMasterZFlex(getPoint,1)], '.k-')

app.UIAxesSide.CameraPosition = [-1,0,-0.075];
app.UIAxesSide.CameraTarget = [0,0,-0.075];

plot3(app.UIAxesTop,[masterPointAFlex(getPoint,1) motorA(1)], [masterPointAFlex(getPoint,2) motorA(2)], [masterPointAFlex(getPoint,3) motorA(3)], wireColor, ...
[masterPointBFlex(getPoint,1) motorB(1)], [masterPointBFlex(getPoint,2) motorB(2)], [masterPointBFlex(getPoint,3) motorB(3)], wireColor, ...
[masterPointCFlex(getPoint,1) motorCPosY(1)], [masterPointCFlex(getPoint,2) motorCPosY(2)], [masterPointCFlex(getPoint,3) motorCPosY(3)], wireColor, ...
[masterPointCFlex(getPoint,1) motorCNegY(1)], [masterPointCFlex(getPoint,2) motorCNegY(2)], [masterPointCFlex(getPoint,3) motorCNegY(3)], wireColor, ...
[masterPointCFlex(getPoint,1) motorCX(1)], [masterPointCFlex(getPoint,2) motorCX(2)], [masterPointCFlex(getPoint,3) motorCX(3)], wireColor, ...
[masterPointDFlex(getPoint,1) motorD(1)], [masterPointDFlex(getPoint,2) motorD(2)], [masterPointDFlex(getPoint,3) motorD(3)], wireColor, ...
[masterPointEFlex(getPoint,1) motorE(1)], [masterPointEFlex(getPoint,2) motorE(2)], [masterPointEFlex(getPoint,3) motorE(3)], wireColor, ...
[masterPointFFlex(getPoint,1) motorFPosY(1)], [masterPointFFlex(getPoint,2) motorFPosY(2)], [masterPointFFlex(getPoint,3) motorFPosY(3)], wireColor, ...
[masterPointFFlex(getPoint,1) motorFNegY(1)], [masterPointFFlex(getPoint,2) motorFNegY(2)], [masterPointFFlex(getPoint,3) motorFNegY(3)], wireColor, ...
[masterPointFFlex(getPoint,1) motorFZ(1)], [masterPointFFlex(getPoint,2) motorFZ(2)], [masterPointFFlex(getPoint,3) motorFZ(3)], wireColor,...
[xCoordsPositionMatrixFlex(getPoint,1) xCoordsPositionMatrixFlex(getPoint,2) xCoordsPositionMatrixFlex(getPoint,5) xCoordsPositionMatrixFlex(getPoint,3) xCoordsPositionMatrixFlex(getPoint,4) xCoordsPositionMatrixFlex(getPoint,1)], ...
[yCoordsPositionMatrixFlex(getPoint,1) yCoordsPositionMatrixFlex(getPoint,2) yCoordsPositionMatrixFlex(getPoint,5) yCoordsPositionMatrixFlex(getPoint,3) yCoordsPositionMatrixFlex(getPoint,4) yCoordsPositionMatrixFlex(getPoint,1)], ...
[zCoordsPositionMatrixFlex(getPoint,1) zCoordsPositionMatrixFlex(getPoint,2) zCoordsPositionMatrixFlex(getPoint,5) zCoordsPositionMatrixFlex(getPoint,3) zCoordsPositionMatrixFlex(getPoint,4) zCoordsPositionMatrixFlex(getPoint,1)],'.k-', ...
[xCoordsPositionMatrixFlex(getPoint,2) FMasterXFlex(getPoint,1)], [yCoordsPositionMatrixFlex(getPoint,2) FMasterYFlex(getPoint,1)], [zCoordsPositionMatrixFlex(getPoint,2) FMasterZFlex(getPoint,1)], '.k-')

app.UIAxesTop.CameraPosition = [0.05,0,1];
app.UIAxesTop.CameraTarget = [0.05,0,0];
```

```
global currentAbIndex

if currentAbIndex > 1
    abCode(1)
    flexCode(getPoint(1))
else
    flexCode(getPoint(1))
end


alue changed function: ShowWiresCheckBox
ction ShowWiresCheckBoxValueChanged(app, event)
global xCoordsPositionMatrixFlex
global yCoordsPositionMatrixFlex
global zCoordsPositionMatrixFlex
global FMasterXFlex
global FMasterYFlex
global FMasterZFlex


global masterPointAFlex
global masterPointBFlex
global masterPointCFlex
global masterPointDFlex
global masterPointEFlex
global masterPointFFlex

global xCoordsPositionMatrixAb
global yCoordsPositionMatrixAb
global zCoordsPositionMatrixAb
global FMasterXAb
global FMasterYAb
global FMasterZAb

global masterPointAAb
global masterPointBAb
global masterPointCAb
global masterPointDAb
global masterPointEAb
global masterPointFAb


global motorA
```

```
global motorA
global motorB
global motorCPosY
global motorCNegY
global motorCX
global motorD
global motorE
global motorFZ
global motorFPosY
global motorFNegY

value = app.ShowWiresCheckBox.Value;
if app.HumeralAbductionSlider.Value > 0
    getPoint = round((round(app.HumeralAbductionSlider.Value,1)/0.1)+1);

    if value == 1
        wireColor = 'r.--';
    elseif value == 0
        wireColor = 'w:square';
    end

    plot3(app.UIAxesAngled,[masterPointAAb(getPoint,1) motorA(1)], [masterPointAAb(getPoint,2) motorA(2)], [masterPointAAb(getPoint,3) motorA(3)], wireColor, ...
    [masterPointBAb(getPoint,1) motorB(1)], [masterPointBAb(getPoint,2) motorB(2)], [masterPointBAb(getPoint,3) motorB(3)], wireColor, ...
    [masterPointCAb(getPoint,1) motorCPosY(1)], [masterPointCAb(getPoint,2) motorCPosY(2)], [masterPointCAb(getPoint,3) motorCPosY(3)], wireColor, ...
    [masterPointCAb(getPoint,1) motorCNegY(1)], [masterPointCAb(getPoint,2) motorCNegY(2)], [masterPointCAb(getPoint,3) motorCNegY(3)], wireColor, ...
    [masterPointCAb(getPoint,1) motorCX(1)], [masterPointCAb(getPoint,2) motorCX(2)], [masterPointCAb(getPoint,3) motorCX(3)], wireColor, ...
    [masterPointDAb(getPoint,1) motorD(1)], [masterPointDAb(getPoint,2) motorD(2)], [masterPointDAb(getPoint,3) motorD(3)], wireColor, ...
    [masterPointEAb(getPoint,1) motorE(1)], [masterPointEAb(getPoint,2) motorE(2)], [masterPointEAb(getPoint,3) motorE(3)], wireColor, ...
    [masterPointFAb(getPoint,1) motorFPosY(1)], [masterPointFAb(getPoint,2) motorFPosY(2)], [masterPointFAb(getPoint,3) motorFPosY(3)], wireColor, ...
    [masterPointFAb(getPoint,1) motorFNegY(1)], [masterPointFAb(getPoint,2) motorFNegY(2)], [masterPointFAb(getPoint,3) motorFNegY(3)], wireColor, ...
    [masterPointFAb(getPoint,1) motorFZ(1)], [masterPointFAb(getPoint,2) motorFZ(2)], [masterPointFAb(getPoint,3) motorFZ(3)], wireColor,...
    [xCoordsPositionMatrixAb(getPoint,1) xCoordsPositionMatrixAb(getPoint,2) xCoordsPositionMatrixAb(getPoint,5) xCoordsPositionMatrixAb(getPoint,3) xCoordsPositionMatrixAb(getPoint,4) xCoordsPositionMatrixAb(getPoint,1)], ...
    [yCoordsPositionMatrixAb(getPoint,1) yCoordsPositionMatrixAb(getPoint,2) yCoordsPositionMatrixAb(getPoint,5) yCoordsPositionMatrixAb(getPoint,3) yCoordsPositionMatrixAb(getPoint,4) yCoordsPositionMatrixAb(getPoint,1)], ...
    [zCoordsPositionMatrixAb(getPoint,1) zCoordsPositionMatrixAb(getPoint,2) zCoordsPositionMatrixAb(getPoint,5) zCoordsPositionMatrixAb(getPoint,3) zCoordsPositionMatrixAb(getPoint,4) zCoordsPositionMatrixAb(getPoint,1)],'.k-', ...
    [xCoordsPositionMatrixAb(getPoint,2) FMasterXAb(getPoint,1)], [yCoordsPositionMatrixAb(getPoint,2) FMasterYAb(getPoint,1)], [zCoordsPositionMatrixAb(getPoint,2) FMasterZAb(getPoint,1)], '.k-')


    plot3(app.UIAxesFront,[masterPointAAb(getPoint,1) motorA(1)], [masterPointAAb(getPoint,2) motorA(2)], [masterPointAAb(getPoint,3) motorA(3)], wireColor, ...
    [masterPointBAb(getPoint,1) motorB(1)], [masterPointBAb(getPoint,2) motorB(2)], [masterPointBAb(getPoint,3) motorB(3)], wireColor, ...
    [masterPointCAb(getPoint,1) motorCPosY(1)], [masterPointCAb(getPoint,2) motorCPosY(2)], [masterPointCAb(getPoint,3) motorCPosY(3)], wireColor, ...
    [masterPointCAb(getPoint,1) motorCNegY(1)], [masterPointCAb(getPoint,2) motorCNegY(2)], [masterPointCAb(getPoint,3) motorCNegY(3)], wireColor, ...
    [masterPointCAb(getPoint,1) motorCX(1)], [masterPointCAb(getPoint,2) motorCX(2)], [masterPointCAb(getPoint,3) motorCX(3)], wireColor, ...
    [masterPointDAb(getPoint,1) motorD(1)], [masterPointDAb(getPoint,2) motorD(2)], [masterPointDAb(getPoint,3) motorD(3)], wireColor, ...
```

```
    [masterPointEAb(getPoint,1) motorE(1)], [masterPointEAb(getPoint,2) motorE(2)], [masterPointEAb(getPoint,3) motorE(3)], wireColor, ...
    [masterPointFAb(getPoint,1) motorFPosY(1)], [masterPointFAb(getPoint,2) motorFPosY(2)], [masterPointFAb(getPoint,3) motorFPosY(3)], wireColor, ...
    [masterPointFAb(getPoint,1) motorFNegY(1)], [masterPointFAb(getPoint,2) motorFNegY(2)], [masterPointFAb(getPoint,3) motorFNegY(3)], wireColor, ...
    [masterPointFAb(getPoint,1) motorFZ(1)], [masterPointFAb(getPoint,2) motorFZ(2)], [masterPointFAb(getPoint,3) motorFZ(3)], wireColor,...
    [xCoordsPositionMatrixAb(getPoint,1) xCoordsPositionMatrixAb(getPoint,2) xCoordsPositionMatrixAb(getPoint,5) xCoordsPositionMatrixAb(getPoint,3) xCoordsPositionMatrixAb(getPoint,4) xCoordsPositionMatrixAb(getPoint,1)], ...
    [yCoordsPositionMatrixAb(getPoint,1) yCoordsPositionMatrixAb(getPoint,2) yCoordsPositionMatrixAb(getPoint,5) yCoordsPositionMatrixAb(getPoint,3) yCoordsPositionMatrixAb(getPoint,4) yCoordsPositionMatrixAb(getPoint,1)], ...
    [zCoordsPositionMatrixAb(getPoint,1) zCoordsPositionMatrixAb(getPoint,2) zCoordsPositionMatrixAb(getPoint,5) zCoordsPositionMatrixAb(getPoint,3) zCoordsPositionMatrixAb(getPoint,4) zCoordsPositionMatrixAb(getPoint,1)],'.k-', ...
    [xCoordsPositionMatrixAb(getPoint,2) FMasterXAb(getPoint,1)], [yCoordsPositionMatrixAb(getPoint,2) FMasterYAb(getPoint,1)], [zCoordsPositionMatrixAb(getPoint,2) FMasterZAb(getPoint,1)], '.k-')


    app.UIAxesFront.CameraPosition = [0.05,1,-0.075];
    app.UIAxesFront.CameraTarget = [0.05,0,-0.075];

    plot3(app.UIAxesSide,[masterPointAAb(getPoint,1) motorA(1)], [masterPointAAb(getPoint,2) motorA(2)], [masterPointAAb(getPoint,3) motorA(3)], wireColor, ...
    [masterPointBAb(getPoint,1) motorB(1)], [masterPointBAb(getPoint,2) motorB(2)], [masterPointBAb(getPoint,3) motorB(3)], wireColor, ...
    [masterPointCAb(getPoint,1) motorCPosY(1)], [masterPointCAb(getPoint,2) motorCPosY(2)], [masterPointCAb(getPoint,3) motorCPosY(3)], wireColor, ...
    [masterPointCAb(getPoint,1) motorCNegY(1)], [masterPointCAb(getPoint,2) motorCNegY(2)], [masterPointCAb(getPoint,3) motorCNegY(3)], wireColor, ...
    [masterPointCAb(getPoint,1) motorCX(1)], [masterPointCAb(getPoint,2) motorCX(2)], [masterPointCAb(getPoint,3) motorCX(3)], wireColor, ...
    [masterPointDAb(getPoint,1) motorD(1)], [masterPointDAb(getPoint,2) motorD(2)], [masterPointDAb(getPoint,3) motorD(3)], wireColor, ...
    [masterPointEAb(getPoint,1) motorE(1)], [masterPointEAb(getPoint,2) motorE(2)], [masterPointEAb(getPoint,3) motorE(3)], wireColor, ...
    [masterPointFAb(getPoint,1) motorFPosY(1)], [masterPointFAb(getPoint,2) motorFPosY(2)], [masterPointFAb(getPoint,3) motorFPosY(3)], wireColor, ...
    [masterPointFAb(getPoint,1) motorFNegY(1)], [masterPointFAb(getPoint,2) motorFNegY(2)], [masterPointFAb(getPoint,3) motorFNegY(3)], wireColor, ...
    [masterPointFAb(getPoint,1) motorFZ(1)], [masterPointFAb(getPoint,2) motorFZ(2)], [masterPointFAb(getPoint,3) motorFZ(3)], wireColor,...
    [xCoordsPositionMatrixAb(getPoint,1) xCoordsPositionMatrixAb(getPoint,2) xCoordsPositionMatrixAb(getPoint,5) xCoordsPositionMatrixAb(getPoint,3) xCoordsPositionMatrixAb(getPoint,4) xCoordsPositionMatrixAb(getPoint,1)], ...
    [yCoordsPositionMatrixAb(getPoint,1) yCoordsPositionMatrixAb(getPoint,2) yCoordsPositionMatrixAb(getPoint,5) yCoordsPositionMatrixAb(getPoint,3) yCoordsPositionMatrixAb(getPoint,4) yCoordsPositionMatrixAb(getPoint,1)], ...
    [zCoordsPositionMatrixAb(getPoint,1) zCoordsPositionMatrixAb(getPoint,2) zCoordsPositionMatrixAb(getPoint,5) zCoordsPositionMatrixAb(getPoint,3) zCoordsPositionMatrixAb(getPoint,4) zCoordsPositionMatrixAb(getPoint,1)],'.k-', ...
    [xCoordsPositionMatrixAb(getPoint,2) FMasterXAb(getPoint,1)], [yCoordsPositionMatrixAb(getPoint,2) FMasterYAb(getPoint,1)], [zCoordsPositionMatrixAb(getPoint,2) FMasterZAb(getPoint,1)], '.k-')


    app.UIAxesSide.CameraPosition = [-1,0,-0.075];
    app.UIAxesSide.CameraTarget = [0,0,-0.075];

    plot3(app.UIAxesTop,[masterPointAAb(getPoint,1) motorA(1)], [masterPointAAb(getPoint,2) motorA(2)], [masterPointAAb(getPoint,3) motorA(3)], wireColor, ...
    [masterPointBAb(getPoint,1) motorB(1)], [masterPointBAb(getPoint,2) motorB(2)], [masterPointBAb(getPoint,3) motorB(3)], wireColor, ...
    [masterPointCAb(getPoint,1) motorCPosY(1)], [masterPointCAb(getPoint,2) motorCPosY(2)], [masterPointCAb(getPoint,3) motorCPosY(3)], wireColor, ...
    [masterPointCAb(getPoint,1) motorCNegY(1)], [masterPointCAb(getPoint,2) motorCNegY(2)], [masterPointCAb(getPoint,3) motorCNegY(3)], wireColor, ...
    [masterPointCAb(getPoint,1) motorCX(1)], [masterPointCAb(getPoint,2) motorCX(2)], [masterPointCAb(getPoint,3) motorCX(3)], wireColor, ...
    [masterPointDAb(getPoint,1) motorD(1)], [masterPointDAb(getPoint,2) motorD(2)], [masterPointDAb(getPoint,3) motorD(3)], wireColor, ...
    [masterPointEAb(getPoint,1) motorE(1)], [masterPointEAb(getPoint,2) motorE(2)], [masterPointEAb(getPoint,3) motorE(3)], wireColor, ...
    [masterPointFAb(getPoint,1) motorFPosY(1)], [masterPointFAb(getPoint,2) motorFPosY(2)], [masterPointFAb(getPoint,3) motorFPosY(3)], wireColor, ...
    [masterPointFAb(getPoint,1) motorFNegY(1)], [masterPointFAb(getPoint,2) motorFNegY(2)], [masterPointFAb(getPoint,3) motorFNegY(3)], wireColor, ...
    [masterPointFAb(getPoint,1) motorFZ(1)], [masterPointFAb(getPoint,2) motorFZ(2)], [masterPointFAb(getPoint,3) motorFZ(3)], wireColor,...
    [xCoordsPositionMatrixAb(getPoint,1) xCoordsPositionMatrixAb(getPoint,2) xCoordsPositionMatrixAb(getPoint,5) xCoordsPositionMatrixAb(getPoint,3) xCoordsPositionMatrixAb(getPoint,4) xCoordsPositionMatrixAb(getPoint,1)], ...
```

```
    [yCoordsPositionMatrixAb(getPoint,1) yCoordsPositionMatrixAb(getPoint,2) yCoordsPositionMatrixAb(getPoint,5) yCoordsPositionMatrixAb(getPoint,3) yCoordsPositionMatrixAb(getPoint,4) yCoordsPositionMatrixAb(getPoint,1)], ...
    [zCoordsPositionMatrixAb(getPoint,1) zCoordsPositionMatrixAb(getPoint,2) zCoordsPositionMatrixAb(getPoint,5) zCoordsPositionMatrixAb(getPoint,3) zCoordsPositionMatrixAb(getPoint,4) zCoordsPositionMatrixAb(getPoint,1)],'.k-', ...
    [xCoordsPositionMatrixAb(getPoint,2) FMasterXAb(getPoint,1)], [yCoordsPositionMatrixAb(getPoint,2) FMasterYAb(getPoint,1)], [zCoordsPositionMatrixAb(getPoint,2) FMasterZAb(getPoint,1)], '.k-')


    app.UIAxesTop.CameraPosition = [0.05,0,1];
    app.UIAxesTop.CameraTarget = [0.05,0,0];

elseif app.HumeralFlexionSlider.Value > 0
    getPoint = round((round(app.HumeralFlexionSlider.Value,1)/0.1)+1);

    if value == 1
        wireColor = 'r.--';
    elseif value == 0
        wireColor = 'w:square';
    end


    plot3(app.UIAxesAngled,[masterPointAFlex(getPoint,1) motorA(1)], [masterPointAFlex(getPoint,2) motorA(2)], [masterPointAFlex(getPoint,3) motorA(3)], wireColor, ...
    [masterPointBFlex(getPoint,1) motorB(1)], [masterPointBFlex(getPoint,2) motorB(2)], [masterPointBFlex(getPoint,3) motorB(3)], wireColor, ...
    [masterPointCFlex(getPoint,1) motorCPosY(1)], [masterPointCFlex(getPoint,2) motorCPosY(2)], [masterPointCFlex(getPoint,3) motorCPosY(3)], wireColor, ...
    [masterPointCFlex(getPoint,1) motorCNegY(1)], [masterPointCFlex(getPoint,2) motorCNegY(2)], [masterPointCFlex(getPoint,3) motorCNegY(3)], wireColor, ...
    [masterPointCFlex(getPoint,1) motorCX(1)], [masterPointCFlex(getPoint,2) motorCX(2)], [masterPointCFlex(getPoint,3) motorCX(3)], wireColor, ...
    [masterPointDFlex(getPoint,1) motorD(1)], [masterPointDFlex(getPoint,2) motorD(2)], [masterPointDFlex(getPoint,3) motorD(3)], wireColor, ...
    [masterPointEFlex(getPoint,1) motorE(1)], [masterPointEFlex(getPoint,2) motorE(2)], [masterPointEFlex(getPoint,3) motorE(3)], wireColor, ...
    [masterPointFFlex(getPoint,1) motorFPosY(1)], [masterPointFFlex(getPoint,2) motorFPosY(2)], [masterPointFFlex(getPoint,3) motorFPosY(3)], wireColor, ...
    [masterPointFFlex(getPoint,1) motorFNegY(1)], [masterPointFFlex(getPoint,2) motorFNegY(2)], [masterPointFFlex(getPoint,3) motorFNegY(3)], wireColor, ...
    [masterPointFFlex(getPoint,1) motorFZ(1)], [masterPointFFlex(getPoint,2) motorFZ(2)], [masterPointFFlex(getPoint,3) motorFZ(3)], wireColor,...
    [xCoordsPositionMatrixFlex(getPoint,1) xCoordsPositionMatrixFlex(getPoint,2) xCoordsPositionMatrixFlex(getPoint,5) xCoordsPositionMatrixFlex(getPoint,3) xCoordsPositionMatrixFlex(getPoint,4) xCoordsPositionMatrixFlex(getPoint,1)], ...
    [yCoordsPositionMatrixFlex(getPoint,1) yCoordsPositionMatrixFlex(getPoint,2) yCoordsPositionMatrixFlex(getPoint,5) yCoordsPositionMatrixFlex(getPoint,3) yCoordsPositionMatrixFlex(getPoint,4) yCoordsPositionMatrixFlex(getPoint,1)], ...
    [zCoordsPositionMatrixFlex(getPoint,1) zCoordsPositionMatrixFlex(getPoint,2) zCoordsPositionMatrixFlex(getPoint,5) zCoordsPositionMatrixFlex(getPoint,3) zCoordsPositionMatrixFlex(getPoint,4) zCoordsPositionMatrixFlex(getPoint,1)],'.k-',
    [xCoordsPositionMatrixFlex(getPoint,2) FMasterXFlex(getPoint,1)], [yCoordsPositionMatrixFlex(getPoint,2) FMasterYFlex(getPoint,1)], [zCoordsPositionMatrixFlex(getPoint,2) FMasterZFlex(getPoint,1)], '.k-')


    plot3(app.UIAxesFront,[masterPointAFlex(getPoint,1) motorA(1)], [masterPointAFlex(getPoint,2) motorA(2)], [masterPointAFlex(getPoint,3) motorA(3)], wireColor, ...
    [masterPointBFlex(getPoint,1) motorB(1)], [masterPointBFlex(getPoint,2) motorB(2)], [masterPointBFlex(getPoint,3) motorB(3)], wireColor, ...
    [masterPointCFlex(getPoint,1) motorCPosY(1)], [masterPointCFlex(getPoint,2) motorCPosY(2)], [masterPointCFlex(getPoint,3) motorCPosY(3)], wireColor, ...
    [masterPointCFlex(getPoint,1) motorCNegY(1)], [masterPointCFlex(getPoint,2) motorCNegY(2)], [masterPointCFlex(getPoint,3) motorCNegY(3)], wireColor, ...
    [masterPointCFlex(getPoint,1) motorCX(1)], [masterPointCFlex(getPoint,2) motorCX(2)], [masterPointCFlex(getPoint,3) motorCX(3)], wireColor, ...
    [masterPointDFlex(getPoint,1) motorD(1)], [masterPointDFlex(getPoint,2) motorD(2)], [masterPointDFlex(getPoint,3) motorD(3)], wireColor, ...
    [masterPointEFlex(getPoint,1) motorE(1)], [masterPointEFlex(getPoint,2) motorE(2)], [masterPointEFlex(getPoint,3) motorE(3)], wireColor, ...
    [masterPointFFlex(getPoint,1) motorFPosY(1)], [masterPointFFlex(getPoint,2) motorFPosY(2)], [masterPointFFlex(getPoint,3) motorFPosY(3)], wireColor, ...
    [masterPointFFlex(getPoint,1) motorFNegY(1)], [masterPointFFlex(getPoint,2) motorFNegY(2)], [masterPointFFlex(getPoint,3) motorFNegY(3)], wireColor, ...
```

```
        [masterPointFFlex(getPoint,1) motorFZ(1)], [masterPointFFlex(getPoint,2) motorFZ(2)], [masterPointFFlex(getPoint,3) motorFZ(3)], wireColor,...
        [xCoordsPositionMatrixFlex(getPoint,1) xCoordsPositionMatrixFlex(getPoint,2) xCoordsPositionMatrixFlex(getPoint,5) xCoordsPositionMatrixFlex(getPoint,3) xCoordsPositionMatrixFlex(getPoint,4) xCoordsPositionMatrixFlex(getPoint,1)], ...
        [yCoordsPositionMatrixFlex(getPoint,1) yCoordsPositionMatrixFlex(getPoint,2) yCoordsPositionMatrixFlex(getPoint,5) yCoordsPositionMatrixFlex(getPoint,3) yCoordsPositionMatrixFlex(getPoint,4) yCoordsPositionMatrixFlex(getPoint,1)], ...
        [zCoordsPositionMatrixFlex(getPoint,1) zCoordsPositionMatrixFlex(getPoint,2) zCoordsPositionMatrixFlex(getPoint,5) zCoordsPositionMatrixFlex(getPoint,3) zCoordsPositionMatrixFlex(getPoint,4) zCoordsPositionMatrixFlex(getPoint,1)],'.k-',
        [xCoordsPositionMatrixFlex(getPoint,2) FMasterXFlex(getPoint,1)], [yCoordsPositionMatrixFlex(getPoint,2) FMasterYFlex(getPoint,1)], [zCoordsPositionMatrixFlex(getPoint,2) FMasterZFlex(getPoint,1)], '.k-')

        app.UIAxesFront.CameraPosition = [0.05,1,-0.075];
        app.UIAxesFront.CameraTarget = [0.05,0,-0.075];

        plot3(app.UIAxesSide,[masterPointAFlex(getPoint,1) motorA(1)], [masterPointAFlex(getPoint,2) motorA(2)], [masterPointAFlex(getPoint,3) motorA(3)], wireColor, ...
        [masterPointBFlex(getPoint,1) motorB(1)], [masterPointBFlex(getPoint,2) motorB(2)], [masterPointBFlex(getPoint,3) motorB(3)], wireColor, ...
        [masterPointCFlex(getPoint,1) motorCPosY(1)], [masterPointCFlex(getPoint,2) motorCPosY(2)], [masterPointCFlex(getPoint,3) motorCPosY(3)], wireColor, ...
        [masterPointCFlex(getPoint,1) motorCNegY(1)], [masterPointCFlex(getPoint,2) motorCNegY(2)], [masterPointCFlex(getPoint,3) motorCNegY(3)], wireColor, ...
        [masterPointCFlex(getPoint,1) motorCX(1)], [masterPointCFlex(getPoint,2) motorCX(2)], [masterPointCFlex(getPoint,3) motorCX(3)], wireColor, ...
        [masterPointDFlex(getPoint,1) motorD(1)], [masterPointDFlex(getPoint,2) motorD(2)], [masterPointDFlex(getPoint,3) motorD(3)], wireColor, ...
        [masterPointEFlex(getPoint,1) motorE(1)], [masterPointEFlex(getPoint,2) motorE(2)], [masterPointEFlex(getPoint,3) motorE(3)], wireColor, ...
        [masterPointFFlex(getPoint,1) motorFPosY(1)], [masterPointFFlex(getPoint,2) motorFPosY(2)], [masterPointFFlex(getPoint,3) motorFPosY(3)], wireColor, ...
        [masterPointFFlex(getPoint,1) motorFNegY(1)], [masterPointFFlex(getPoint,2) motorFNegY(2)], [masterPointFFlex(getPoint,3) motorFNegY(3)], wireColor, ...
        [masterPointFFlex(getPoint,1) motorFZ(1)], [masterPointFFlex(getPoint,2) motorFZ(2)], [masterPointFFlex(getPoint,3) motorFZ(3)], wireColor,...
        [xCoordsPositionMatrixFlex(getPoint,1) xCoordsPositionMatrixFlex(getPoint,2) xCoordsPositionMatrixFlex(getPoint,5) xCoordsPositionMatrixFlex(getPoint,3) xCoordsPositionMatrixFlex(getPoint,4) xCoordsPositionMatrixFlex(getPoint,1)], ...
        [yCoordsPositionMatrixFlex(getPoint,1) yCoordsPositionMatrixFlex(getPoint,2) yCoordsPositionMatrixFlex(getPoint,5) yCoordsPositionMatrixFlex(getPoint,3) yCoordsPositionMatrixFlex(getPoint,4) yCoordsPositionMatrixFlex(getPoint,1)], ...
        [zCoordsPositionMatrixFlex(getPoint,1) zCoordsPositionMatrixFlex(getPoint,2) zCoordsPositionMatrixFlex(getPoint,5) zCoordsPositionMatrixFlex(getPoint,3) zCoordsPositionMatrixFlex(getPoint,4) zCoordsPositionMatrixFlex(getPoint,1)],'.k-',
        [xCoordsPositionMatrixFlex(getPoint,2) FMasterXFlex(getPoint,1)], [yCoordsPositionMatrixFlex(getPoint,2) FMasterYFlex(getPoint,1)], [zCoordsPositionMatrixFlex(getPoint,2) FMasterZFlex(getPoint,1)], '.k-')

        app.UIAxesSide.CameraPosition = [-1,0,-0.075];
        app.UIAxesSide.CameraTarget = [0,0,-0.075];

        plot3(app.UIAxesTop,[masterPointAFlex(getPoint,1) motorA(1)], [masterPointAFlex(getPoint,2) motorA(2)], [masterPointAFlex(getPoint,3) motorA(3)], wireColor, ...
        [masterPointBFlex(getPoint,1) motorB(1)], [masterPointBFlex(getPoint,2) motorB(2)], [masterPointBFlex(getPoint,3) motorB(3)], wireColor, ...
        [masterPointCFlex(getPoint,1) motorCPosY(1)], [masterPointCFlex(getPoint,2) motorCPosY(2)], [masterPointCFlex(getPoint,3) motorCPosY(3)], wireColor, ...
        [masterPointCFlex(getPoint,1) motorCNegY(1)], [masterPointCFlex(getPoint,2) motorCNegY(2)], [masterPointCFlex(getPoint,3) motorCNegY(3)], wireColor, ...
        [masterPointCFlex(getPoint,1) motorCX(1)], [masterPointCFlex(getPoint,2) motorCX(2)], [masterPointCFlex(getPoint,3) motorCX(3)], wireColor, ...
        [masterPointDFlex(getPoint,1) motorD(1)], [masterPointDFlex(getPoint,2) motorD(2)], [masterPointDFlex(getPoint,3) motorD(3)], wireColor, ...
        [masterPointEFlex(getPoint,1) motorE(1)], [masterPointEFlex(getPoint,2) motorE(2)], [masterPointEFlex(getPoint,3) motorE(3)], wireColor, ...
        [masterPointFFlex(getPoint,1) motorFPosY(1)], [masterPointFFlex(getPoint,2) motorFPosY(2)], [masterPointFFlex(getPoint,3) motorFPosY(3)], wireColor, ...
        [masterPointFFlex(getPoint,1) motorFNegY(1)], [masterPointFFlex(getPoint,2) motorFNegY(2)], [masterPointFFlex(getPoint,3) motorFNegY(3)], wireColor, ...
        [masterPointFFlex(getPoint,1) motorFZ(1)], [masterPointFFlex(getPoint,2) motorFZ(2)], [masterPointFFlex(getPoint,3) motorFZ(3)], wireColor,...
        [xCoordsPositionMatrixFlex(getPoint,1) xCoordsPositionMatrixFlex(getPoint,2) xCoordsPositionMatrixFlex(getPoint,5) xCoordsPositionMatrixFlex(getPoint,3) xCoordsPositionMatrixFlex(getPoint,4) xCoordsPositionMatrixFlex(getPoint,1)], ...
        [yCoordsPositionMatrixFlex(getPoint,1) yCoordsPositionMatrixFlex(getPoint,2) yCoordsPositionMatrixFlex(getPoint,5) yCoordsPositionMatrixFlex(getPoint,3) yCoordsPositionMatrixFlex(getPoint,4) yCoordsPositionMatrixFlex(getPoint,1)], ...
        [zCoordsPositionMatrixFlex(getPoint,1) zCoordsPositionMatrixFlex(getPoint,2) zCoordsPositionMatrixFlex(getPoint,5) zCoordsPositionMatrixFlex(getPoint,3) zCoordsPositionMatrixFlex(getPoint,4) zCoordsPositionMatrixFlex(getPoint,1)],'.k-',
        [xCoordsPositionMatrixFlex(getPoint,2) FMasterXFlex(getPoint,1)], [yCoordsPositionMatrixFlex(getPoint,2) FMasterYFlex(getPoint,1)], [zCoordsPositionMatrixFlex(getPoint,2) FMasterZFlex(getPoint,1)], '.k-')

        app.UIAxesTop.CameraPosition = [0.05,0,1];
        app.UIAxesTop.CameraTarget = [0.05,0,0];

    elseif app.HumeralFlexionSlider.Value == 0 && app.HumeralAbductionSlider.Value==0
        getPoint = round((round(app.HumeralFlexionSlider.Value,1)/0.1)+1);

        if value == 1
            wireColor = 'r.--';
        elseif value == 0
            wireColor = 'w:square';
        end


        plot3(app.UIAxesAngled,[masterPointAFlex(getPoint,1) motorA(1)], [masterPointAFlex(getPoint,2) motorA(2)], [masterPointAFlex(getPoint,3) motorA(3)], wireColor, ...
        [masterPointBFlex(getPoint,1) motorB(1)], [masterPointBFlex(getPoint,2) motorB(2)], [masterPointBFlex(getPoint,3) motorB(3)], wireColor, ...
        [masterPointCFlex(getPoint,1) motorCPosY(1)], [masterPointCFlex(getPoint,2) motorCPosY(2)], [masterPointCFlex(getPoint,3) motorCPosY(3)], wireColor, ...
        [masterPointCFlex(getPoint,1) motorCNegY(1)], [masterPointCFlex(getPoint,2) motorCNegY(2)], [masterPointCFlex(getPoint,3) motorCNegY(3)], wireColor, ...
        [masterPointCFlex(getPoint,1) motorCX(1)], [masterPointCFlex(getPoint,2) motorCX(2)], [masterPointCFlex(getPoint,3) motorCX(3)], wireColor, ...
        [masterPointDFlex(getPoint,1) motorD(1)], [masterPointDFlex(getPoint,2) motorD(2)], [masterPointDFlex(getPoint,3) motorD(3)], wireColor, ...
        [masterPointEFlex(getPoint,1) motorE(1)], [masterPointEFlex(getPoint,2) motorE(2)], [masterPointEFlex(getPoint,3) motorE(3)], wireColor, ...
        [masterPointFFlex(getPoint,1) motorFPosY(1)], [masterPointFFlex(getPoint,2) motorFPosY(2)], [masterPointFFlex(getPoint,3) motorFPosY(3)], wireColor, ...
        [masterPointFFlex(getPoint,1) motorFNegY(1)], [masterPointFFlex(getPoint,2) motorFNegY(2)], [masterPointFFlex(getPoint,3) motorFNegY(3)], wireColor, ...
        [masterPointFFlex(getPoint,1) motorFZ(1)], [masterPointFFlex(getPoint,2) motorFZ(2)], [masterPointFFlex(getPoint,3) motorFZ(3)], wireColor,...
        [xCoordsPositionMatrixFlex(getPoint,1) xCoordsPositionMatrixFlex(getPoint,2) xCoordsPositionMatrixFlex(getPoint,5) xCoordsPositionMatrixFlex(getPoint,3) xCoordsPositionMatrixFlex(getPoint,4) xCoordsPositionMatrixFlex(getPoint,1)], ...
        [yCoordsPositionMatrixFlex(getPoint,1) yCoordsPositionMatrixFlex(getPoint,2) yCoordsPositionMatrixFlex(getPoint,5) yCoordsPositionMatrixFlex(getPoint,3) yCoordsPositionMatrixFlex(getPoint,4) yCoordsPositionMatrixFlex(getPoint,1)], ...
        [zCoordsPositionMatrixFlex(getPoint,1) zCoordsPositionMatrixFlex(getPoint,2) zCoordsPositionMatrixFlex(getPoint,5) zCoordsPositionMatrixFlex(getPoint,3) zCoordsPositionMatrixFlex(getPoint,4) zCoordsPositionMatrixFlex(getPoint,1)],'.k-'
        [xCoordsPositionMatrixFlex(getPoint,2) FMasterXFlex(getPoint,1)], [yCoordsPositionMatrixFlex(getPoint,2) FMasterYFlex(getPoint,1)], [zCoordsPositionMatrixFlex(getPoint,2) FMasterZFlex(getPoint,1)], '.k-')


        plot3(app.UIAxesFront,[masterPointAFlex(getPoint,1) motorA(1)], [masterPointAFlex(getPoint,2) motorA(2)], [masterPointAFlex(getPoint,3) motorA(3)], wireColor, ...
        [masterPointBFlex(getPoint,1) motorB(1)], [masterPointBFlex(getPoint,2) motorB(2)], [masterPointBFlex(getPoint,3) motorB(3)], wireColor, ...
        [masterPointCFlex(getPoint,1) motorCPosY(1)], [masterPointCFlex(getPoint,2) motorCPosY(2)], [masterPointCFlex(getPoint,3) motorCPosY(3)], wireColor, ...
        [masterPointCFlex(getPoint,1) motorCNegY(1)], [masterPointCFlex(getPoint,2) motorCNegY(2)], [masterPointCFlex(getPoint,3) motorCNegY(3)], wireColor, ...
        [masterPointCFlex(getPoint,1) motorCX(1)], [masterPointCFlex(getPoint,2) motorCX(2)], [masterPointCFlex(getPoint,3) motorCX(3)], wireColor, ...
        [masterPointDFlex(getPoint,1) motorD(1)], [masterPointDFlex(getPoint,2) motorD(2)], [masterPointDFlex(getPoint,3) motorD(3)], wireColor, ...
        [masterPointEFlex(getPoint,1) motorE(1)], [masterPointEFlex(getPoint,2) motorE(2)], [masterPointEFlex(getPoint,3) motorE(3)], wireColor, ...
        [masterPointFFlex(getPoint,1) motorFPosY(1)], [masterPointFFlex(getPoint,2) motorFPosY(2)], [masterPointFFlex(getPoint,3) motorFPosY(3)], wireColor, ...
        [masterPointFFlex(getPoint,1) motorFNegY(1)], [masterPointFFlex(getPoint,2) motorFNegY(2)], [masterPointFFlex(getPoint,3) motorFNegY(3)], wireColor, ...
        [masterPointFFlex(getPoint,1) motorFZ(1)], [masterPointFFlex(getPoint,2) motorFZ(2)], [masterPointFFlex(getPoint,3) motorFZ(3)], wireColor,...
        [xCoordsPositionMatrixFlex(getPoint,1) xCoordsPositionMatrixFlex(getPoint,2) xCoordsPositionMatrixFlex(getPoint,5) xCoordsPositionMatrixFlex(getPoint,3) xCoordsPositionMatrixFlex(getPoint,4) xCoordsPositionMatrixFlex(getPoint,1)], ...
        [yCoordsPositionMatrixFlex(getPoint,1) yCoordsPositionMatrixFlex(getPoint,2) yCoordsPositionMatrixFlex(getPoint,5) yCoordsPositionMatrixFlex(getPoint,3) yCoordsPositionMatrixFlex(getPoint,4) yCoordsPositionMatrixFlex(getPoint,1)], ...
        [zCoordsPositionMatrixFlex(getPoint,1) zCoordsPositionMatrixFlex(getPoint,2) zCoordsPositionMatrixFlex(getPoint,5) zCoordsPositionMatrixFlex(getPoint,3) zCoordsPositionMatrixFlex(getPoint,4) zCoordsPositionMatrixFlex(getPoint,1)],'.k-'
        [xCoordsPositionMatrixFlex(getPoint,2) FMasterXFlex(getPoint,1)], [yCoordsPositionMatrixFlex(getPoint,2) FMasterYFlex(getPoint,1)], [zCoordsPositionMatrixFlex(getPoint,2) FMasterZFlex(getPoint,1)], '.k-')

        app.UIAxesFront.CameraPosition = [0.05,1,-0.075];
        app.UIAxesFront.CameraTarget = [0.05,0,-0.075];

        plot3(app.UIAxesSide,[masterPointAFlex(getPoint,1) motorA(1)], [masterPointAFlex(getPoint,2) motorA(2)], [masterPointAFlex(getPoint,3) motorA(3)], wireColor, ...
        [masterPointBFlex(getPoint,1) motorB(1)], [masterPointBFlex(getPoint,2) motorB(2)], [masterPointBFlex(getPoint,3) motorB(3)], wireColor, ...
        [masterPointCFlex(getPoint,1) motorCPosY(1)], [masterPointCFlex(getPoint,2) motorCPosY(2)], [masterPointCFlex(getPoint,3) motorCPosY(3)], wireColor, ...
        [masterPointCFlex(getPoint,1) motorCNegY(1)], [masterPointCFlex(getPoint,2) motorCNegY(2)], [masterPointCFlex(getPoint,3) motorCNegY(3)], wireColor, ...
        [masterPointCFlex(getPoint,1) motorCX(1)], [masterPointCFlex(getPoint,2) motorCX(2)], [masterPointCFlex(getPoint,3) motorCX(3)], wireColor, ...
        [masterPointDFlex(getPoint,1) motorD(1)], [masterPointDFlex(getPoint,2) motorD(2)], [masterPointDFlex(getPoint,3) motorD(3)], wireColor, ...
        [masterPointEFlex(getPoint,1) motorE(1)], [masterPointEFlex(getPoint,2) motorE(2)], [masterPointEFlex(getPoint,3) motorE(3)], wireColor, ...
        [masterPointFFlex(getPoint,1) motorFPosY(1)], [masterPointFFlex(getPoint,2) motorFPosY(2)], [masterPointFFlex(getPoint,3) motorFPosY(3)], wireColor, ...
        [masterPointFFlex(getPoint,1) motorFNegY(1)], [masterPointFFlex(getPoint,2) motorFNegY(2)], [masterPointFFlex(getPoint,3) motorFNegY(3)], wireColor, ...
        [masterPointFFlex(getPoint,1) motorFZ(1)], [masterPointFFlex(getPoint,2) motorFZ(2)], [masterPointFFlex(getPoint,3) motorFZ(3)], wireColor,...
        [xCoordsPositionMatrixFlex(getPoint,1) xCoordsPositionMatrixFlex(getPoint,2) xCoordsPositionMatrixFlex(getPoint,5) xCoordsPositionMatrixFlex(getPoint,3) xCoordsPositionMatrixFlex(getPoint,4) xCoordsPositionMatrixFlex(getPoint,1)], ...
        [yCoordsPositionMatrixFlex(getPoint,1) yCoordsPositionMatrixFlex(getPoint,2) yCoordsPositionMatrixFlex(getPoint,5) yCoordsPositionMatrixFlex(getPoint,3) yCoordsPositionMatrixFlex(getPoint,4) yCoordsPositionMatrixFlex(getPoint,1)], ...
        [zCoordsPositionMatrixFlex(getPoint,1) zCoordsPositionMatrixFlex(getPoint,2) zCoordsPositionMatrixFlex(getPoint,5) zCoordsPositionMatrixFlex(getPoint,3) zCoordsPositionMatrixFlex(getPoint,4) zCoordsPositionMatrixFlex(getPoint,1)],'.k-', ...
        [xCoordsPositionMatrixFlex(getPoint,2) FMasterXFlex(getPoint,1)], [yCoordsPositionMatrixFlex(getPoint,2) FMasterYFlex(getPoint,1)], [zCoordsPositionMatrixFlex(getPoint,2) FMasterZFlex(getPoint,1)], '.k-')

        app.UIAxesSide.CameraPosition = [-1,0,-0.075];
        app.UIAxesSide.CameraTarget = [0,0,-0.075];

        plot3(app.UIAxesTop,[masterPointAFlex(getPoint,1) motorA(1)], [masterPointAFlex(getPoint,2) motorA(2)], [masterPointAFlex(getPoint,3) motorA(3)], wireColor, ...
        [masterPointBFlex(getPoint,1) motorB(1)], [masterPointBFlex(getPoint,2) motorB(2)], [masterPointBFlex(getPoint,3) motorB(3)], wireColor, ...
        [masterPointCFlex(getPoint,1) motorCPosY(1)], [masterPointCFlex(getPoint,2) motorCPosY(2)], [masterPointCFlex(getPoint,3) motorCPosY(3)], wireColor, ...
        [masterPointCFlex(getPoint,1) motorCNegY(1)], [masterPointCFlex(getPoint,2) motorCNegY(2)], [masterPointCFlex(getPoint,3) motorCNegY(3)], wireColor, ...
        [masterPointCFlex(getPoint,1) motorCX(1)], [masterPointCFlex(getPoint,2) motorCX(2)], [masterPointCFlex(getPoint,3) motorCX(3)], wireColor, ...
        [masterPointDFlex(getPoint,1) motorD(1)], [masterPointDFlex(getPoint,2) motorD(2)], [masterPointDFlex(getPoint,3) motorD(3)], wireColor, ...
        [masterPointEFlex(getPoint,1) motorE(1)], [masterPointEFlex(getPoint,2) motorE(2)], [masterPointEFlex(getPoint,3) motorE(3)], wireColor, ...
        [masterPointFFlex(getPoint,1) motorFPosY(1)], [masterPointFFlex(getPoint,2) motorFPosY(2)], [masterPointFFlex(getPoint,3) motorFPosY(3)], wireColor, ...
        [masterPointFFlex(getPoint,1) motorFNegY(1)], [masterPointFFlex(getPoint,2) motorFNegY(2)], [masterPointFFlex(getPoint,3) motorFNegY(3)], wireColor, ...
        [masterPointFFlex(getPoint,1) motorFZ(1)], [masterPointFFlex(getPoint,2) motorFZ(2)], [masterPointFFlex(getPoint,3) motorFZ(3)], wireColor,...
        [xCoordsPositionMatrixFlex(getPoint,1) xCoordsPositionMatrixFlex(getPoint,2) xCoordsPositionMatrixFlex(getPoint,5) xCoordsPositionMatrixFlex(getPoint,3) xCoordsPositionMatrixFlex(getPoint,4) xCoordsPositionMatrixFlex(getPoint,1)], ...
        [yCoordsPositionMatrixFlex(getPoint,1) yCoordsPositionMatrixFlex(getPoint,2) yCoordsPositionMatrixFlex(getPoint,5) yCoordsPositionMatrixFlex(getPoint,3) yCoordsPositionMatrixFlex(getPoint,4) yCoordsPositionMatrixFlex(getPoint,1)], ...
        [zCoordsPositionMatrixFlex(getPoint,1) zCoordsPositionMatrixFlex(getPoint,2) zCoordsPositionMatrixFlex(getPoint,5) zCoordsPositionMatrixFlex(getPoint,3) zCoordsPositionMatrixFlex(getPoint,4) zCoordsPositionMatrixFlex(getPoint,1)],'.k-', ...
        [xCoordsPositionMatrixFlex(getPoint,2) FMasterXFlex(getPoint,1)], [yCoordsPositionMatrixFlex(getPoint,2) FMasterYFlex(getPoint,1)], [zCoordsPositionMatrixFlex(getPoint,2) FMasterZFlex(getPoint,1)], '.k-')

        app.UIAxesTop.CameraPosition = [0.05,0,1];
        app.UIAxesTop.CameraTarget = [0.05,0,0];
```

```
    % Component initialization
    methods (Access = private)

        % Create UIFigure and components
        function createComponents(app)

            % Create UIFigure and hide until all components are created
            app.UIFigure = uifigure('Visible', 'off');
            app.UIFigure.Position = [100 100 786 588];
            app.UIFigure.Name = 'MATLAB App';

            % Create UIAxesAngled
            app.UIAxesAngled = uiaxes(app.UIFigure);
            title(app.UIAxesAngled, 'Angled View')
            app.UIAxesAngled.XLim = [-0.1 0.574];
            app.UIAxesAngled.YLim = [-0.49 0.49];
            app.UIAxesAngled.ZLim = [-0.48 0.48];
            app.UIAxesAngled.Position = [1 326 399 239];

            % Create UIAxesFront
            app.UIAxesFront = uiaxes(app.UIFigure);
            title(app.UIAxesFront, 'Front View')
            app.UIAxesFront.XLim = [-0.1 0.574];
            app.UIAxesFront.YLim = [-0.49 0.49];
            app.UIAxesFront.ZLim = [-0.48 0.48];
            app.UIAxesFront.Position = [399 326 388 239];

            % Create UIAxesSide
            app.UIAxesSide = uiaxes(app.UIFigure);
            title(app.UIAxesSide, 'Side View')
            app.UIAxesSide.View = [0 0];
            app.UIAxesSide.XLim = [-0.1 0.574];
            app.UIAxesSide.YLim = [-0.49 0.49];
            app.UIAxesSide.ZLim = [-0.48 0.48];
            app.UIAxesSide.Position = [20 107 380 220];

            % Create UIAxesTop
            app.UIAxesTop = uiaxes(app.UIFigure);
            title(app.UIAxesTop, 'Top View')
            app.UIAxesTop.CameraPosition = [0 0 1];
            app.UIAxesTop.CameraTarget = [0 0 0];
            app.UIAxesTop.XLim = [-0.1 0.574];
            app.UIAxesTop.YLim = [-0.49 0.49];
```


```
            app.UIAxesTop.YLim = [-0.49 0.49];
```


```
            app.UIAxesTop.ZLim = [-0.48 0.48];
            app.UIAxesTop.Position = [370 107 387 220];

            % Create HumeralAbductionSliderLabel
            app.HumeralAbductionSliderLabel = uilabel(app.UIFigure);
            app.HumeralAbductionSliderLabel.HorizontalAlignment = 'right';
            app.HumeralAbductionSliderLabel.Position = [209 88 106 22];
            app.HumeralAbductionSliderLabel.Text = 'Humeral Abduction';

            % Create HumeralAbductionSlider
            app.HumeralAbductionSlider = uislider(app.UIFigure);
            app.HumeralAbductionSlider.Limits = [0 120];
            app.HumeralAbductionSlider.ValueChangedFcn = createCallbackFcn(app, @HumeralAbductionSliderValueChanging, true);
            app.HumeralAbductionSlider.Position = [336 97 382 3];

            % Create HumeralFlexionSliderLabel
            app.HumeralFlexionSliderLabel = uilabel(app.UIFigure);
            app.HumeralFlexionSliderLabel.HorizontalAlignment = 'right';
            app.HumeralFlexionSliderLabel.Position = [224 45 92 22];
            app.HumeralFlexionSliderLabel.Text = 'Humeral Flexion';

            % Create HumeralFlexionSlider
            app.HumeralFlexionSlider = uislider(app.UIFigure);
            app.HumeralFlexionSlider.Limits = [0 120];
            app.HumeralFlexionSlider.ValueChangedFcn = createCallbackFcn(app, @HumeralFlexionSliderValueChanging, true);
            app.HumeralFlexionSlider.Position = [337 54 381 3];

            % Create ShowWiresCheckBox
            app.ShowWiresCheckBox = uicheckbox(app.UIFigure);
            app.ShowWiresCheckBox.ValueChangedFcn = createCallbackFcn(app, @ShowWiresCheckBoxValueChanged, true);
            app.ShowWiresCheckBox.Text = 'Show Wires';
            app.ShowWiresCheckBox.Position = [104 45 86 43];

            % Show the figure after all components are created
            app.UIFigure.Visible = 'on';
        end
    end

    % App creation and deletion
    methods (Access = public)

        % Construct app
```


```
            app.HumeralFlexionSlider.Limits = [0 120];
            app.HumeralFlexionSlider.ValueChangedFcn = createCallbackFcn(app, @HumeralFlexionSliderValueChanging, true);
            app.HumeralFlexionSlider.Position = [337 54 381 3];

            % Create ShowWiresCheckBox
            app.ShowWiresCheckBox = uicheckbox(app.UIFigure);
            app.ShowWiresCheckBox.ValueChangedFcn = createCallbackFcn(app, @ShowWiresCheckBoxValueChanged, true);
            app.ShowWiresCheckBox.Text = 'Show Wires';
            app.ShowWiresCheckBox.Position = [104 45 86 43];

            % Show the figure after all components are created
            app.UIFigure.Visible = 'on';
        end
    end

    % App creation and deletion
    methods (Access = public)

        % Construct app
```


```
        function app = shoulderRotationApp

            % Create UIFigure and components
            createComponents(app)

            % Register the app with App Designer
            registerApp(app, app.UIFigure)

            % Execute the startup function
            runStartupFcn(app, @startupFcn)

            if nargout == 0
                clear app
            end
        end

        % Code that executes before app deletion
        function delete(app)

            % Delete UIFigure when app is deleted
            delete(app.UIFigure)
        end
    end
end
```

8.0 References

Anatomical Terminology. Anatomical Terminology | SEER Training. (n.d.). Retrieved March 14, 2024, from https://training.seer.cancer.gov/anatomy/body/terminology.html

Avin KG, Bloomfield SA, Gross TS, Warden SJ. Biomechanical aspects of the muscle-bone interaction. Curr Osteoporos Rep. 2015 Feb;13(1):1-8. Doi: 10.1007/s11914-014-0244-x. PMID: 25515697; PMCID: PMC4306629.

Bagg, S. D., & Forrest, W. J. (2016, March 4). A biomechanical analysis of scapula rotation during arm abduction in the scapula plane. Academia.edu. Retrieved March 14, 2024, from https://www.academia.edu/22798706/A_biomechanical_analysis_of_scapula_rotation_during_arm_abduction_in_the_scapula_Plane?bulkDownload=thisPaper-topRelated-sameAuthor-citingThis-citedByThis-secondOrderCitations&from=cover_page

Bolsterlee, B., Veeger, D. H. E. J., & Chadwick, E. K. (2013). Clinical applications of musculoskeletal modelling for the shoulder and upper limb. Medical & Biological Engineering & Computing, 51(9), 953–963. https://doi.org/10.1007/s11517-013-1099-5

Bones of the shoulder. jointspecialists.org. (2017, March 27). Retrieved March 14, 2024, from https://jointspecialists.org/bones-of-the-shoulder/

Chua, Chee Kai, and Wai Yee Yeong. "Impact Toughness." *Impact Toughness - an Overview | ScienceDirect Topics*, www.sciencedirect.com/topics/engineering/impact-toughness. Accessed 14 Mar. 2024.

Deane, Ella, et al. Worcester, Massachusetts, 2022, *Anatomically Accurate Motorized Shoulder Model with Scapula Movement*.

Engelhardt, C., Camine , M., V., Ingram D., Müllhaupt P., Farron A., Pioletti D., Terrier A. (April 4, 2014). Comparison of an EMG-based and a stress-based method to predict shoulder muscle forces, Computer Methods in Biomechanics and Biomedical Engineering. Taylor& Francis Online Journal, 1272-1279, DOI: 10.1080/10255842.2014.899587

Garofalo, Pietro. "Clavicular Component of Scapulo-Humeral Rhythm." *ResearchGate*, www.researchgate.net/figure/Clavicular-Component-of-Scapulo-humeral-Rhythm-Third-III-top-phase-of-scapulohumeral_fig15_24019863. Accessed 14 Mar. 2024.

Igoshin, Nikita. Worcester, Massachusetts, 2022, *Producing Accurate Relative Motion in a Shoulder Model*.

Mathoma. (July 26, 2016). "3D Rotations in General: Rodrigues Rotation Formula and Quaternion Exponentials" *YouTube*, https://www.youtube.com/watch?v=q-ESzg03mQc. Accessed 12 Mar. 2024.

McEvilly, Fiona, et al. Worcester, Massachusetts, 2023, *Realistic Shoulder Model with Soft Tissue Attachments*.

O'Leary S, Christensen SW, Verouhis A, Pape M, Nilsen O, McPhail SM. Agreement between physiotherapists rating scapular posture in multiple planes in patients with neck pain: Reliability study. Physiotherapy. 2015 Dec;101(4):381-8. doi: 10.1016/j.physio.2015.01.005. Epub 2015 Jan 25. PMID: 25749493.

"Online Learning for Physiotherapists." *Physiotutors*, 21 Feb. 2024, Physiotutors.com/.

OpenStax. "University Physics Volume 1." *10.6 Torque | University Physics Volume 1*, 3 Aug. 2016, courses.lumenlearning.com/suny-osuniversityphysics/chapter/10-6-torque/.

Physics LibreTexts, Libretexts, 12 Mar. 2024, phys.libretexts.org/Bookshelves/Conceptual_Physics/Introduction_to_Physics_(Park)/03%3A_Unit_2-_Mechanics_II_-_Energy_and_Momentum_Oscillations_and_Waves_Rotation_and_Fluids/06%3A_Rotation/6.03%3A_Dynamics_of_Rotational_Motion-_Rotational_Inertia.

"Pololu Robotics and Electronics." *Pololu Robotics & Electronics*, www.pololu.com/. Accessed 14 Mar. 2024.

“Programming with Matlab.” *MATLAB & Simulink*, www.mathworks.com/products/matlab/programming-with-matlab.html. Accessed 14 Mar. 2024.

Shelton, Stephen M., and William H. Swanger. *Fatigue Properties of Steel Wire*, nvlpubs.nist.gov/nistpubs/jres/14/jresv14n1p17_A1b.pdf. Accessed 14 Mar. 2024.

Shojaei, Amir Mohammad. “Connect Two Arduino Boards Using I2C Communication Protocol.” *Electropeak*, 28 May 2023, electropeak.com/learn/connect-two-arduino-boards-using-i2c-communication-protocol/.

Singh, A. P. (n.d.) Muscles of Shoulder Region. Bones and Spine. https://boneandspine.com/muscles-of-shoulder/

“StepperOnline.” *STEPPERONLINE*, StepperOnline.com/. Accessed 14 Mar. 2024.

“Tutorials.” *Arduino*, www.arduino.cc/en/Tutorial/HomePage. Accessed 14 Mar. 2024.

Ultimaker Nylon: Technical Data Sheet. (2022, 20 Apr). Ultimaker. Retrieved March 14, 2024, from https://ultimaker.com/materials/nylon

Ultimaker TPU 95A: Technical Data Sheet. (2022, 29 Apr). Ultimaker. March 14, 2024, from https://ultimaker.com/materials/tpu-95a

“The Ultimate Resource for Healthcare Professionals & Medical Students.” *TeachMeAnatomy*, teachmeanatomy.info/. Accessed 14 Mar. 2024.

“Unleashing the Power of Programming.” *An Introduction To Coding*, www.arduino.cc/education/an-introduction-to-coding/. Accessed 14 Mar. 2024.

Yabata, Kazuyuki, and Tsutomu Fukui. “Characteristics of the Scapula Movement during Shoulder Elevation Depend on Posture.” *Journal of Physical Therapy Science*, U.S. National Library of Medicine, July 2022, www.ncbi.nlm.nih.gov/pmc/articles/PMC9246406/.