

Detailed Design Report

Trey Dufrene Zack Johnson David Orcutt Alan Wallingford Ryan Warner

Submitted in Partial Fulfillment of the Requirements of:
ME420 Detailed Design of Robotic Systems – Spring 2020



Meiosis

College of Engineering
Embry-Riddle Aeronautical University
Prescott, AZ

Abstract

The Manipulator for Educational Institutions with Open Source Integrated Systems (MEIOSIS) aims to increase the accessibility of robotics to secondary educational institutions and hobbyists. Accordantly, the manipulator is 3D printed in PLA with aluminum tube supports and costs the end-user less than \$1000. The manipulator has six links and a base. The base houses a Raspberry Pi 3B and power supply. The Raspberry Pi controls seven Dynamixel smart servos with position feedback and proportional derivative control. Six MX-12W servos actuate six rotational joints, while one AX-12A servo actuates the removable end-effector. They provide the manipulator a position repeatability within 2mm of the previous pose. The manipulator can draw as well as perform pick and place operations within its dexterous workspace which is a hemispherical sub-shell of the reachable workspace of 280 mm thickness. The manipulator's operation is controlled by open-source software.

Contents

1	Introduction	1
2	Conceptual Design	1
3	Specifications	8
4	Preliminary and Detailed Design	13
5	Electrical System	13
6	Mechanical System	13
7	Software	25
8	Testing	30
9	Introduction	30
10	Specifications and Tests	30
11	Further Testing	42
12	Conclusion	43
13	Project Management	44
14	Conclusion	44

List of Figures

1	Overall System Conceptual Design	1
2	Manipulator Base with Call-outs	2
3	Drawing Showing Key Features of Design	3
4	Drawing Showing Link Cross Section	3
5	Electrical System Block Diagram	4
6	Software Flowchart	5
7	Overview of Physical System	8
8	Elbow Manipulator Configuration with Link Offset	9
9	Kinematic Model Representing Zeroed Configuration	11
10	Harmonic Gearbox Locations	13
11	Exploded View of the Harmonic Gearbox	14
12	Motor Locations and Orientations	17
13	Closed-Loop Control Simulation Animation Snapshots	19
14	Joint Angles vs Time in Closed-Loop Simulation	19
15	T-Bar ANSYS FEA	20
16	ANSYS Simulated Forces Image Capture	21
17	ANSYS FEA of Dynamical Loading Scenario	21
18	ANSYS Fatigue Test	22
19	Software Flowchart	25
20	Labeled Image of Manipulator Rendering	31
21	Measurement Test for Link Offset	32
22	Manipulator in its Zeroed Configuration	33
23	Solidworks Measurement of Reachable Workspace	34
24	Test of Link Length for Specification 1.5.a.	35
25	Matlab Plot of Invalid Dexterity Check	36
26	Matlab Plot of Successful Dexterity Check	37
27	Dynamixel Parallel Gripper with Servo	38
28	Dynamixel Gripper with and without Foam Padding at Minimum and Maximum Opening	38
29	Dynamixel Gripper Rotational to Linear Motion Mechanism [5]	39
30	Dynamixel Parallel Gripper with Servo	40
31	GUI Template	41

List of Tables

1	Configuration Decision Matrix	6
2	Material Choice Decision Matrix	6
3	Computing Choice Decision Matrix	7
4	End Effector Attachment Design Decision Matrix	7
5	MEIOSIS Bill of Materials with Costs	23
6	End-User Bill of Materials with Costs	24
7	MEIOSIS Bill of Materials with Costs	31
8	Measurements of Manipulator Link Lengths	35
9	Results of Angle Measurement Test for Specification 2.1.6.a	37
10	Summary of Test Results	43

List Of Acronyms and Abbreviations

FK : Forward Kinematics

IK : Inverse Kinematics

PD : Proportional Derivative

Notation

$r_{\text{From Frame To}}$: Direction Vectors

$T_{\text{From To}}$: Direction Cosine (Transformation) Matrices

$c_{\theta_{nm}}$: $\cos(\theta_n + \theta_m)$

$s_{\theta_{nm}}$: $\sin(\theta_n + \theta_m)$

1 Introduction

Robotics is a rapidly growing field, yet numerous potential innovators for the discipline are never exposed to it. While many industrial style manipulators are available for commercial applications, their cost is so high that they are not practical for educational purposes. With the intention of making robotics education more accessible to classrooms, the Manipulator for Educational Institutions with Open Source Integrated Systems (MEIOSIS) aims to provide robotics classes with an accurate and precise manipulator for cost lower than traditional manipulators. MEIOSIS is designed to be 3D printed by the end-user to increase accessibility of the manipulator and will be modifiable to help create an increased understanding of robotics. The following document details the entirety of the design process, from conceptual design and definition of requirements to final implementation and testing.

2 Conceptual Design

The terminator T-2000 is a science-fiction spectacle of a robot – until you see the price. Channelling the inspiration many high school students may have for robotics, MEIOSIS robotics aims to provide an affordable manipulator to educators and enthusiasts. MEIOSIS uses primarily 3-D printed components and easily accessible materials. Among these materials are a Raspberry PI, smart servos and metal tubing. These features create an open-source manipulator accessible to the public to further robotics education.

2.1 Physical System Overview

The physical design of the robotic manipulator is shown in Figures 1, 2, 3, and 4.

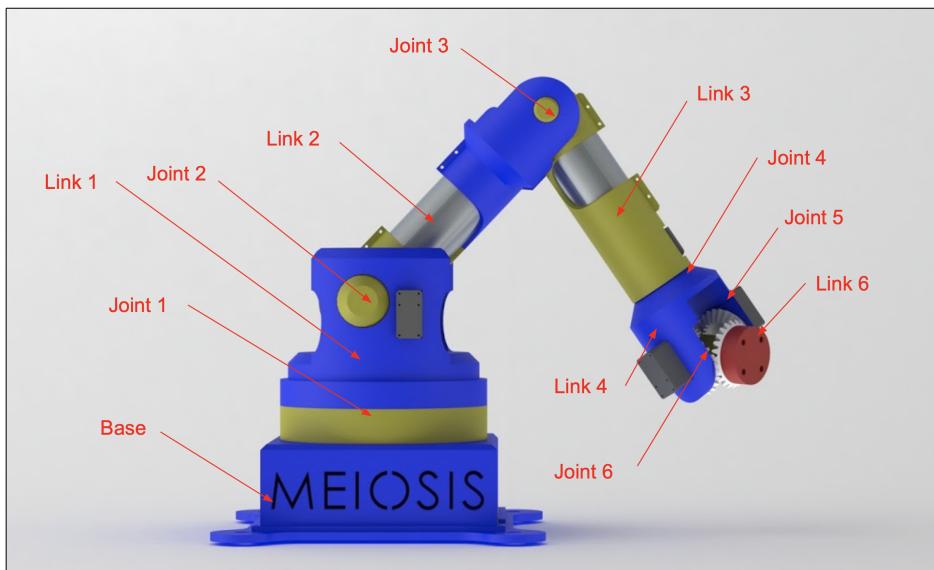


Figure 1: Overall System Conceptual Design

The colored links in *Figure 1* distinguish the different joints and links of the manipulator. The overall reach of the robot is 582.5 mm. This length was chosen to decrease material

cost and weight while still satisfying requirement 2.1.2 and 2.1.5, allowing the manipulating to pick and place objects to perform basic tasks. The base of the robot is made to contain the Raspberry Pi and other electrical components.

2.1.1 Base

The base of the manipulator will house several of the electronic components, such as the computational system, power supply, and motor controller. A cross section of the base can be seen in *Figure 2*.

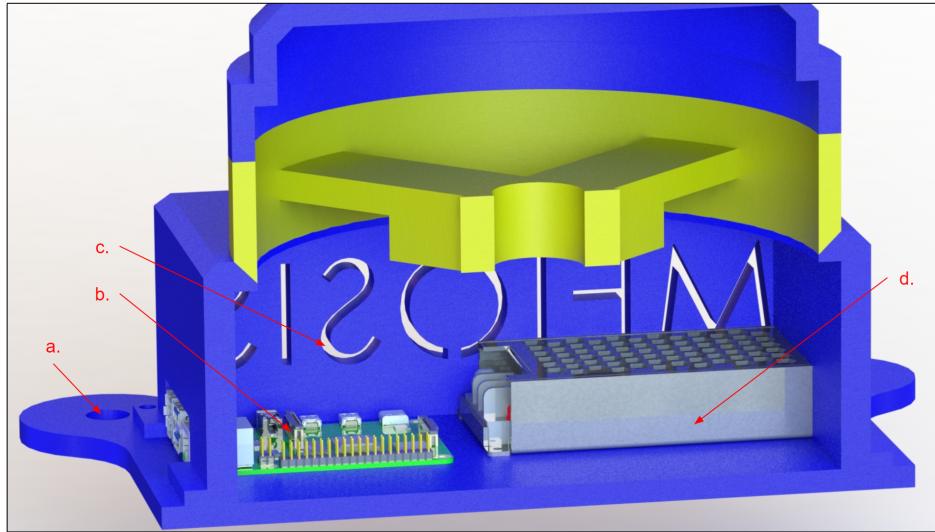


Figure 2: Manipulator Base with Call-outs

From *Figure 2*,

a. Base Supports: The base supports are located at each corner of the base and allow the base of the manipulator to be securely attached to a variety of surfaces with either standard fasteners or suction cups.

b. Computational System: The computational system is a Raspberry Pi; it is housed in the base, which allows the Raspberry Pi to be more easily accessible. The primary reason for this system being chosen is to fulfill the budget requirement, 2.1.1. The Raspberry Pi computes the manipulator's inverse kinematics and sends the angle commands to the servos.

c. Airflow Cutouts: The side of the base has cutouts to allow for airflow through the base; since the power supply is housed inside of the base as well as the computational system, the temperature must be regulated to prevent overheating.

d. Power Supply: The power supply is housed in the base as well, which allows the power supply to be more accessible and therefore more modifiable, so the end-user can easily expand the system to fulfill their needs.

2.1.2 Links

Figure 3 shows the links and their key features.

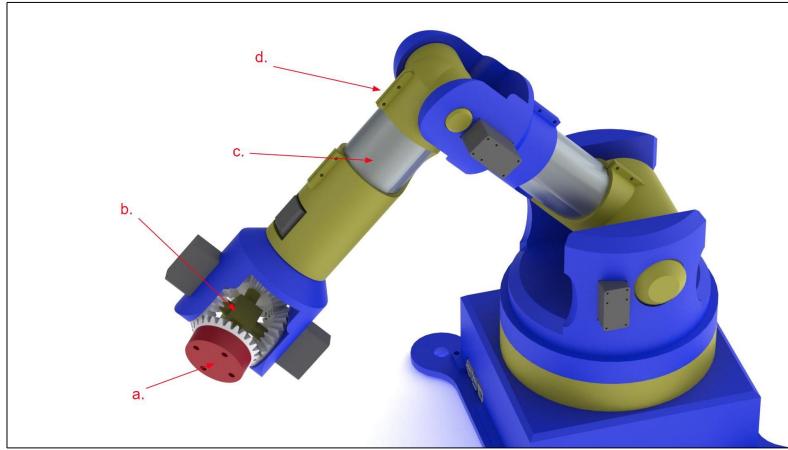


Figure 3: Drawing Showing Key Features of Design

In *Figure 3*, call-out (a) shows the connection point for the end effector. The mounting layout is the standard used by the Sawyer manipulator. The end effector mounting layout may be adjusted to accommodate lower cost, more accessible end effectors. Call-out (b) shows the differential gearbox that is used in the manipulator's wrist, saving space and weight. The manipulator has aluminum tubing as support in the links (c) and are attached to the 3D printed portion of the robot using clamp joints (d) tightened by screws.

Figure 4 depicts the cross section of link 2 for the manipulator.

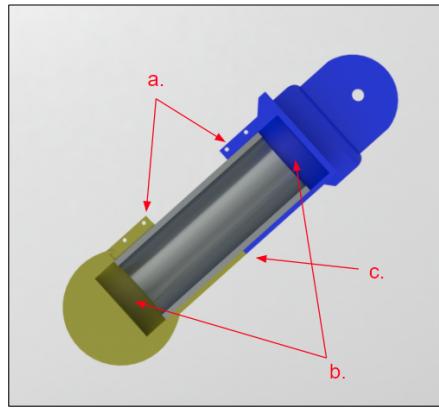


Figure 4: Drawing Showing Link Cross Section

The cross section seen in *Figure 4* shows the internal design for links two and three. It features two clamps that hold a hollow aluminum bar in place (a) and allows for gaps between the aluminum tube and the 3D printed call-out (b). The proper length is dictated

by the 3D printed guides lining up at call-out (c). This design allows for imprecision in the manufacturing of the aluminum tube.

2.2 System Functions

The system can be divided into two subsystems: the electrical and software systems. The electrical subsystem includes the wiring and hardware computational components, power system, actuators with drivers, and sensors. The software subsystem includes the algorithm for the computational system.

2.3 Electrical

Figure 5 is the block diagram for the electrical system of the manipulator.

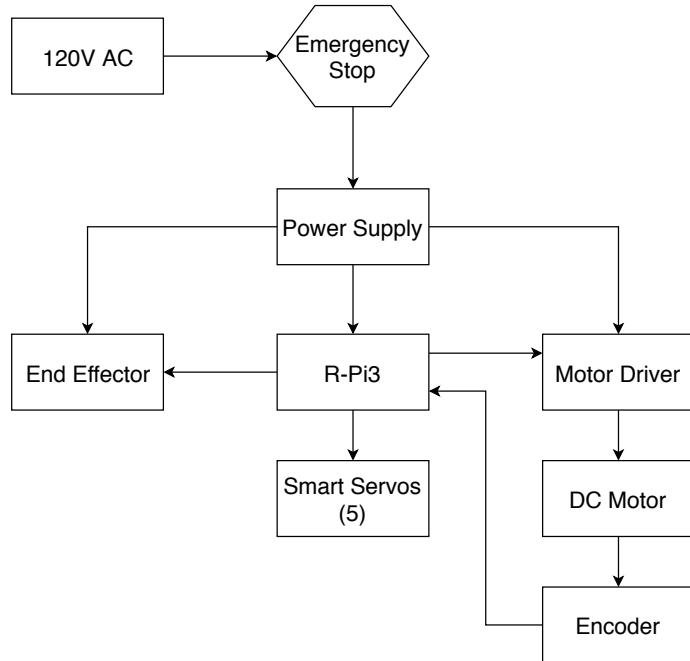


Figure 5: Electrical System Block Diagram

Figure 5 shows that the electrical systems of the manipulator consists of three main components: the power supply, the Raspberry Pi, and the servos. Power is supplied by the 120V AC from standard wall outlets. A power supply converts the AC voltage to the required voltages for each component. One component is the Raspberry Pi, which performs calculations for motor control. The Raspberry Pi sends signals to the DC motor driver and the five smart servos. The smart servos have an on-board controller, so no feedback will be necessary. However, the first joint, between the base and the first link, is actuated by a DC motor with an encoder to minimize cost.

2.4 Software

Figure 6 shows the software flowchart for the system.

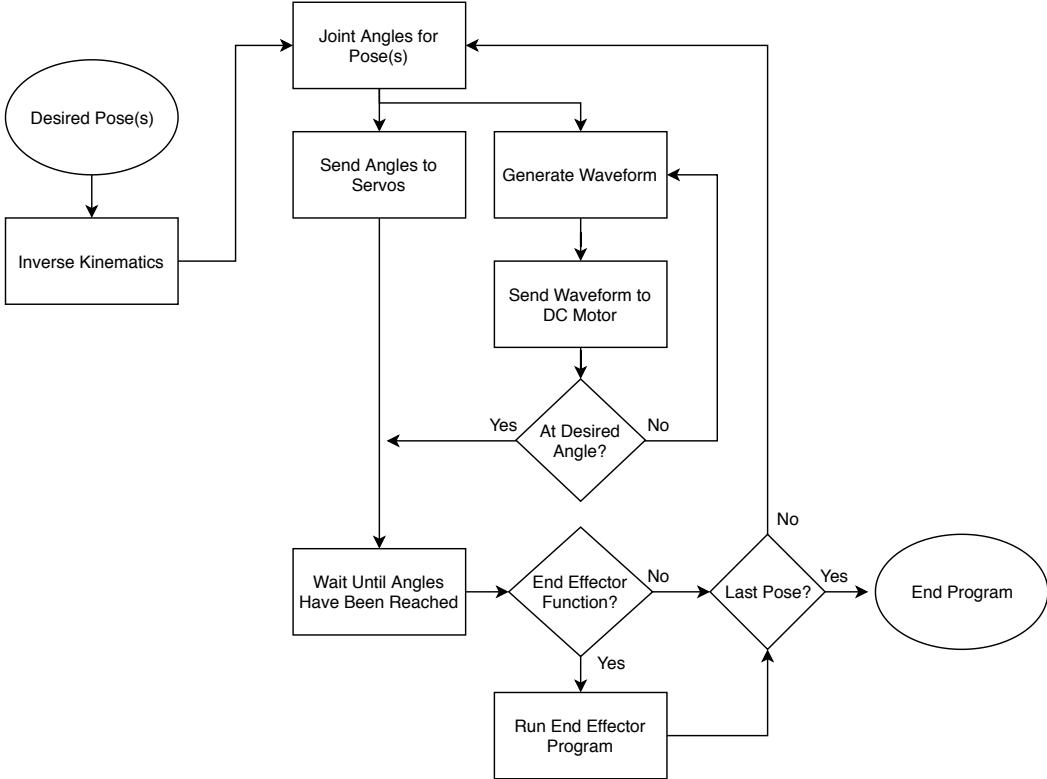


Figure 6: Software Flowchart

Figure 6 shows that the software receives the desired pose or poses that the user would like the manipulator to reach. Next, the Raspberry Pi uses inverse kinematics to calculate the necessary joint angles. The wave-forms/desired angles are sent to the respective drivers/motors, and positional information is be sent back to the Raspberry Pi to adjust the DC motor angle. When the motors have reached their desired pose, the Raspberry Pi actuates the end effector if it is specified by the user. The system then checks to see if there are any more poses to reach and either repeats the steps in the motor control section given the desired angles of the new pose or ends the program if the last pose has been reached.

2.5 Decision Matrices

Each team member's conceptual design provides a variety of different options for select issues, and in order to narrow down the options into the best choice decision matrices are used to make unobjective decisions about each individual topic. The weights of each criteria are defined by the average of our votes on how important the criteria is on a scale of one to ten. The same method is used to rate the different options on each criteria. The most important decision that needs to be made is what configuration the manipulator will have as the configuration has an affect on almost all the design choices. There are five potential configurations, and the scores of each option can be seen in *Table 1*.

The results from *Table 1* show that the solely rotational configuration is the best option,

Table 1: Configuration Decision Matrix

Configuration Weighting	Cost	Educational Value	Ease of Use	Ease to Program	Ease of Manufacture	Total
	10	8	7	2	6	
Cartesian w/ Spherical Wrist	5	9	10	8	4	232
RRR w/ Spherical Wrist	8	9	7	7	8	263
Cylindrical	7	10	7	8	7	257
SCARA	6	9	8	8	6	240
Spherical	7	10	7	7	6	249

followed closely by the cylindrical. Having translational joints is too be problematic for manufacturing and more expensive, and that is enough of an issue that the rotational manipulator has the advantage over the rest of the options.

Another design choice that needs to be made is what material the manipulator will be constructed out of. The main choices are to 3D print everything, manufacture the manipulator out of aluminum, or use a combination of the two. Cost and accessibility are the two highest weighted criteria as we deem them to be the most important aspects, followed by weight and manufacturability. Durability has the lowest weight because the manipulator should not have enough forces acting on it so that the strength of PLA compared to aluminum should not necessarily affect the manipulator. The grades of each material are shown in *Table 2*.

Table 2: Material Choice Decision Matrix

Material Weighting	Cost	Weight	Accessibility	Manufacturability	Durability	Total
	9.2	6.6	8	6.4	5.4	
3D Printed	8	9	8	9.4	5	284.16
Aluminum	4	5.8	5.8	6.6	9.2	213.4
Combination	8	7.8	7	9	9.2	288.36

As seen in *Table 2*, the winning material is a combination of 3D printing and aluminum. The combination allows the manipulator to have 3D printed parts that would be hard to manufacture out of aluminum or that would not necessarily need to be extremely durable, and allows the use of aluminum for the aspects of the manipulator that need to be strong and durable. This option allows for the best aspects of both materials to be utilized.

The internal computing system for the manipulator is another decision that needs to be made, so another decision matrix is used to decide from the six options available. Because the manipulator is intended to be low cost and used by primary education institutions, cost and ease of use are the two highest weighted criteria. Having a lot of capabilities and being able to function independently are factored in, but are deemed of secondary importance which can be seen in *Table 3*.

Table 3: Computing Choice Decision Matrix

Computing System	Cost	Capabilities	Ease of Use	Independence	Total
Weights	10	4.2	7.6	4.2	
RPi	7.9	6	9	10	214.6
Jetson	1	7	7	9	130.4
RPi + Arduino	4.8	7	8	10	180.2
Arduino	9	4	9	3	187.8
B Black	1.2	7	7	10	136.6
B Green	5	7	6	9	162.8

Table 3 shows that the Raspberry Pi is the winner by a significant margin, with only the Arduino coming close to it. While the Pi is not the most capable, its cost is very good and its ease of use and ability to independent functionality edge it over the other more capable competition. The arduino is the most cost effective, but its lack of functionality and low independence are detrimental to its score.

One of the other designs that needs to be selected was the way that the end effectors are attached to the end of the manipulator, and the different categories that are used to grade each option are ease of use, manufacturability, and durability, as seen in *Table 4*.

Table 4: End Effector Attachment Design Decision Matrix

EE Attachment	Ease of Use	Manufacturability	Durability	Total
Weighting	3.6	5	7	
Screw Connections	6.8	8.8	9.8	137.08
Snap Fit Joint	8.6	5	2.2	71.36
Threaded End Effector	6.3	5.4	7.3	100.78

From *Table 4*, the optimal design turns out to be screw connections in the end effector. The snap fit joint would be easier to use, but to create a functional snap fit connection would be difficult and would likely not be too durable. The threaded end effector is difficult to manufacture and not durable as well, so having connections for screw on the end effector is the clear choice.

3 Specifications

With the intention of making robotics education more accessible, The Manipulator for Educational Institutions with Open Source Integrated Systems (MEIOSIS) intends to provide high school educators and robot enthusiasts with a low cost manipulator. The system should be usable by novice students. It should also be modifiable to create a sustainably increased understanding of robotics. While MEIOSIS may not fully emulate industrial manipulators, it aims to provide more students with access to robotics education.

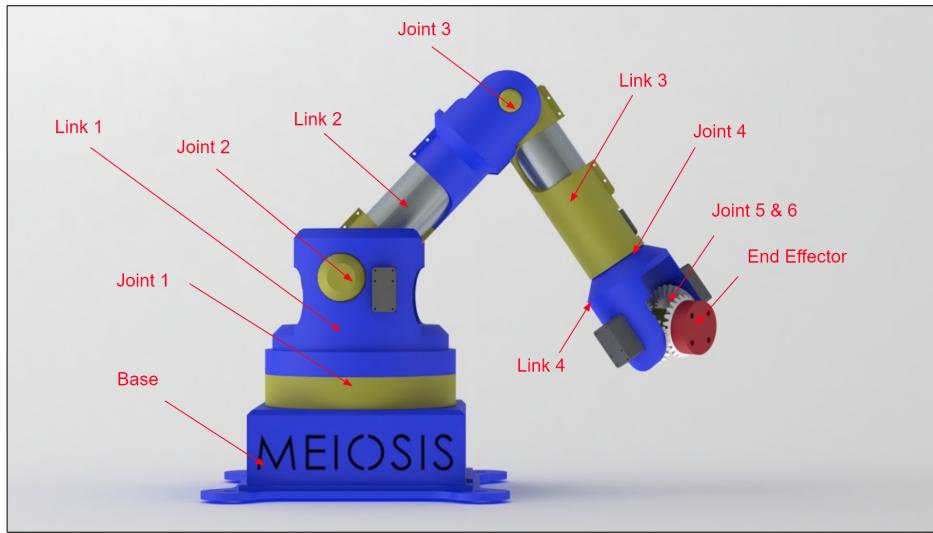


Figure 7: Overview of Physical System

The design seen in *Figure 7* is based on our conceptual design. It features four links and six joints for rotation and will be referenced throughout this document. The base of the manipulator and end-effector can also be seen in the figure.

3.1 Design Requirements

The specifications of the system are strictly based on the requirements defined previously. The requirements are divided into two primary categories, hardware and software.

3.2 Hardware

The following requirements and specifications are hardware specific and dictate the physical constraints the system must adhere to.

3.2.1 The system shall cost the end-user no more than \$1000.

3.2.1.a *The cost for the MEIOSIS team to develop the manipulator shall cost no more than \$800.*

3.2.2 The system shall be fully dexterous without being kinematically redundant.

3.2.2.a The system shall consist of six rotational joints connected by four links. The last three joints will create a spherical wrist.

As defined [9], “A manipulator having more than six DOF is referred to as a kinematically redundant manipulator (5).” A manipulator with less than six degrees of freedom will not be fully dexterous within it’s workspace. *Figure 22* (see subsection 3.2.6, p. 33) shows a six degree-of-freedom rotary manipulator with it’s coordinate frames in zeroed positions. The joint and link locations are seen in *Figure 7* (see section 9, p. 30).

3.2.2.b The system shall have no link offsets.

Link offsets as seen in *Figure 8* are commonly used to avoid singularities. However, having a link offset prevents the manipulator’s dexterous workspace from being a complete hemispherical shell.

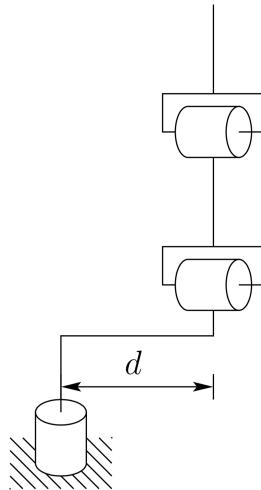


Figure 8: Elbow Manipulator Configuration with Link Offset [9]

As shown in *Figure 8*, the line directly above the first joint of the manipulator is offset such that the axes of the other joints are unable to become collinear with the base axis; this prevents singularity but causes a void in the dexterous workspace.

3.2.3 The system end effector shall maintain a positional accuracy magnitude of ± 1 mm and an orientation accuracy of $\pm 5^\circ$ eigen angle from the base frame.

To ensure that the robot has educational value, the accuracy must be defined so that any desired positions and movements are achieved with sufficient accuracy.

3.2.3.a The system shall accommodate a process in which the end user can calibrate the

end effector position and orientation to within 0.5 mm and 1 degree of the manipulator's precision.

The addition of a calibration process allows the removal of any systematic errors, such as drift. The theoretical limit of the calibration process is the difference between the precision and accuracy metrics of the system.

3.2.4 The system end effector shall maintain a pose repeatability magnitude between 0.1—1.5 mm for the position and $\pm 4^\circ$ eigen angle from the base frame for the orientation.

3.2.4.a Joint one and two of the system shall possess an angle error of no more than .025 degrees.

Being that joint one and two are the first two rotational elements in the system, their error will propagate the most to the end effector's position.

3.2.4.b Joint three of the system shall possess an angle error of no more than .03 degrees.

Since joint three is closer to the end effector it's error will not propagate as severely throughout the system.

3.2.4.c Joints four, five, and six shall possess an angle error of no more than .29 degrees.

The spherical wrist is the closest to the end effector's final position and therefore has the least error propagation.

3.2.5 The system's reachable workspace shall be a hemisphere with a radius of 300-700 mm.

This workspace will provide enough movement to manipulate objects in order to perform basic tasks.

3.2.5.a The length of link one, two, three, four, and the wrist shall be 220.8 mm, 250 mm, 200 mm, 80 mm, and 52.5 mm respectively.

This results in a total height of 220.8 mm with a total reach of 582.5 mm in the zeroed configuration as shown in the configuration represented in *Figure 22*.

3.2.6 The system's dexterous workspace shall contain a hemispherical shell within the reachable workspace with a thickness of 280 mm.

This workspace will provide enough movement to manipulate objects in order to perform basic tasks. 280mm is slightly greater than the length of letter paper.

3.2.6.a The rotational limit of joint one, two, three, four, five, and six shall be $\pm 180^\circ$, -9.7°

to 177.5° , -150.6° to -19.3° , $\pm 180^\circ$, -180° to -1.6° , and $\pm 180^\circ$ respectively.

The angles stated are with respect to the kinematic model shown in *Figure 22*. To be fully dexterous within our 280 mm dexterous workspace the manipulator must have the joint angles specified above. The joint limitations were calculated by iteratively verifying the orientation about every point within the quarter hemisphere cross section seen in *Figure ??* (see Appendix, p. 46).

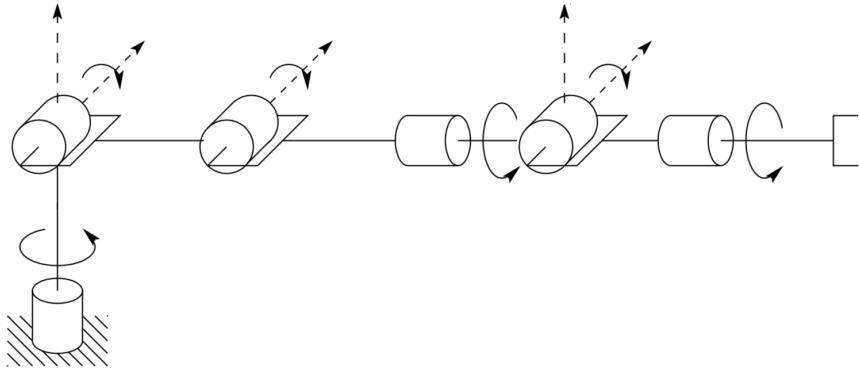


Figure 9: Kinematic Model Representing Zeroed Configuration [9]

3.2.7 The system shall have a removable end effector capable of picking and placing a low-odor chisel tip Expo dry erase marker.

This creates a robot capable of performing a variety of basic tasks, which enhances its educational value.

3.2.7.a The system shall use a parallel gripper that can close to 18mm.

The diameter of a low-odor chisel tip Expo dry erase marker is approximately 18 mm.

3.2.7.b The end effector shall attach to the manipulator using screws configured in a pattern that can accommodate a Dynamixel AX-12A servo.

It is expected that a majority of end effector styles will have to accommodate for a servo to facilitate actuation, therefore a pattern was chosen to standardize the mounting.

3.2.8 The system shall be able to write with a low-odor chisel tip Expo dry erase marker.

3.2.8.a The end effector shall be able to support 0.004 Newton meter moments about the axes normal to its gripping surfaces.

The coefficient of friction between the Expo marker and paper can be approximated and given the weight of an Expo marker the approximate grip strength of the end effector can

be calculated.

3.3 Software

The following requirements and specifications are software specific and determine the attributes of the operating system.

3.3.1 The system shall be open source.

This will create an easily obtainable, low cost method of distributing the system's source code, which may be modified for personal use.

3.3.1.a The software shall be hosted publicly on an online repository and maintain an MIT license for distribution.

This allows the end-user to freely download and modify the code without licensing. The MIT license disregards any legal obligation to code upkeep and documentation by the original author.

3.3.2 The system shall be capable of operating given only desired end effector cartesian coordinates specified with respect to the base frame.

3.3.2.a The system shall have a user interface capable of accepting the end-effector's desired cartesian position and Euler angle orientation as a six element row vector.

The system software interface facilitates an untrained user to operate without the advanced knowledge of the system's kinematics.

3.3.2.b The system shall be capable of performing floating point arithmetic.

The solution for the inverse kinematics requires the ability to perform high level arithmetic with little error.

4 Preliminary and Detailed Design

5 Electrical System

6 Mechanical System

Description of Mechanical Design

One of the main differences between the conceptual mechanical design and the final design is the inclusion of 3D-printed harmonic gearboxes. The previous designs utilized a combination of standard spur gears and timing belts. It was determined that neither of these options provided the necessary precision without also introducing backlash into the system, which is nearly impossible to correct for with our feedback configuration. The solution was to design and implement a harmonic drive system. The harmonic gearbox locations are shown below in Figure 10

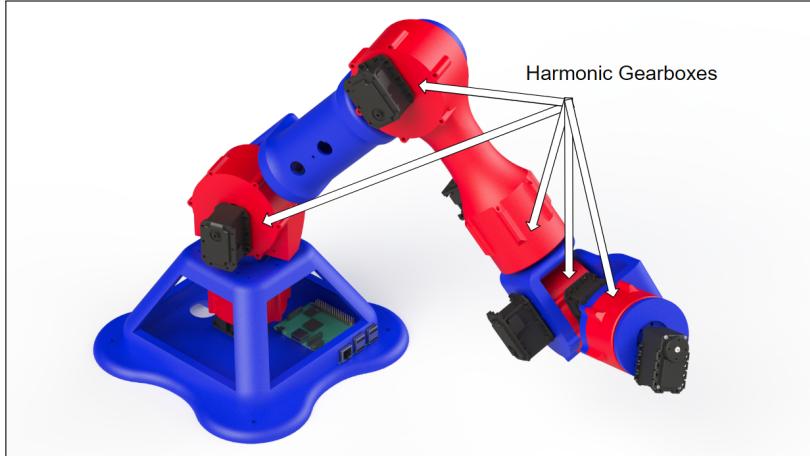


Figure 10: Harmonic Gearbox Locations

As seen in Figure 10, each joint only requires a single actuator, which is a configuration not seen in the previous design. This is possible because harmonic drives allow for very high gear ratios with minimal backlash. The first prototype harmonic drive was provided by a posting on Thingiverse [citation]. This design was meant to be driven by a stepper motor and provided a gear ratio of 39:1. However, the gearboxes implemented in the manipulator need to be driven by a smart servo. Because the smart servo cannot drive as fast as a stepper motor, the ratio of 39:1 resulted in joint movement that was extremely slow. Additionally, given the link lengths and precision of the servos, only a gear ratio of 20:1 was necessary to meet the precision requirements of the system. For these reasons, a second harmonic drive was designed to provide the proper gear ratio of 20:1. Figure 11 provides an exploded view of the main structural components of the newest harmonic drive.

As seen in Figure 11, the basic harmonic drive design features an outer housing, a stiff output shaft, a flexible gear, and a bearing holder assembly. When fully assembled, the

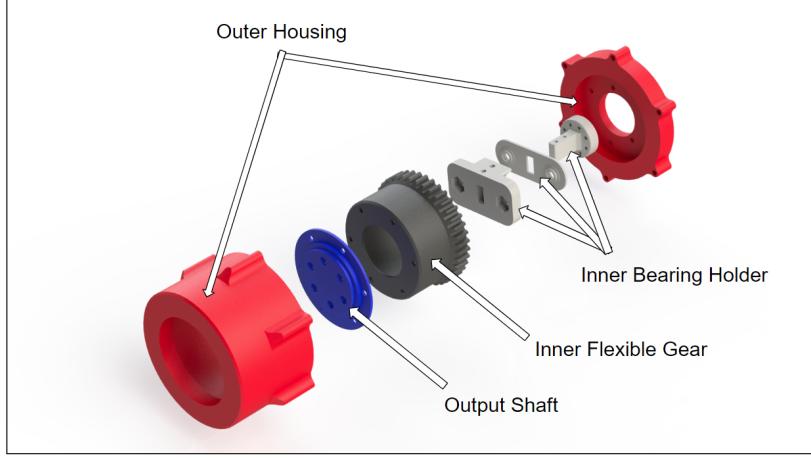


Figure 11: Exploded View of the Harmonic Gearbox

bearing holder sits on the inside of the flexible gear. As the servo drives the inner bearing holder, a continuous meshing of teeth occurs between the flexible gear and the set of teeth that are built into the outer housing. Because the inner set of teeth has two less teeth than the outer set (the inside has 40 while the outside has 42), the inner gear will move forward by two teeth per rotation of the bearing holder. This results in 1/20th of a rotation of the flexible gear per full rotation of the bearing holder. With the output shaft being attached to the flexible gear, the whole harmonic drive provides a ratio of 20:1.

Equations of Motion

Given robot dynamics described by $H(\gamma)\ddot{\gamma} + d(\gamma, \dot{\gamma}) + G(\gamma) = F_\gamma$, the equations of motion for the manipulator can be determined. Solving this equation for the acceleration, $\ddot{\gamma}$, gives:

$$\ddot{\gamma} = H(\gamma)^{-1} (F_\gamma - d(\gamma, \dot{\gamma}) - G(\gamma)) \quad (1)$$

Where H is the system mass matrix, F_γ is the vector of generalized forces, d is the vector of centripital and coriolis effects, and G is gravitational effects.

$$H(\gamma) = \sum_B^N J_B(\gamma)^T \begin{bmatrix} {}_B^B J & \mathring{S}({}_B^B \Gamma)^I T_B^T \\ {}^I T_B \mathring{S}({}_B^B \Gamma)^T & m_B I \end{bmatrix} J_B(\gamma) , \quad {}_B^B \Gamma = {}_B^B r_{cm} m_b , \quad \mathring{S}(\omega)r = (\omega \times r)$$

$$d(\gamma, \dot{\gamma}) = \sum_B^N J_B(\gamma)^T \begin{bmatrix} {}_B^B J & \mathring{S}({}_B^B \Gamma)^I T_B^T \\ {}^I T_B \mathring{S}({}_B^B \Gamma)^T & m_B I \end{bmatrix} J_B(\gamma, \dot{\gamma}) + J_B(\gamma)^T \begin{bmatrix} {}_B^B \omega_I \times {}_B^B J_B^B \omega_I \\ {}^I T_B \left({}_B^B \omega_I \times ({}_B^B \omega_I \times {}_B^B \Gamma) \right) \end{bmatrix}$$

$$G(\gamma) = \left(\frac{\partial U({}^I r(\gamma))}{\partial \gamma} \right)^T , \quad U_B = [0 \ 0 \ g] \left({}_B^B r_B m_B + {}^I T_B {}_B^B \Gamma \right)$$

Where J_B is the jacobian of the body, Γ is the vector of first mass moments, m_B is the mass of the body, and ${}_B^B \omega_I$ is the rotational velocity of the body relative to the inertial frame.

Actuator Dynamics

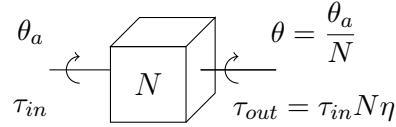
Given robot dynamics described by $H(\gamma)\ddot{\gamma} + n(\gamma, \dot{\gamma}) = \tau$, the torque, τ , provided by the servo motors is necessary to solve the closed loop dynamics of the system. Assuming the servo is driven by a D.C. motor with proportional derivative control,

$$\tau_a = Ki_a = J_a\ddot{\theta}_a + b_a\dot{\theta}_a + \tau_L \quad (2)$$

Where τ_a is the actuator torque, K is the back-EMF constant, i_a is the motor current, J_a is the armature inertia, θ_a , $\dot{\theta}_a$, $\ddot{\theta}_a$ is the motor position and it's first and second time derivatives respectively, b_a is the viscous friction coefficient, and τ_L is the torque available for the actuator to do work. The basic equation for a motor is known to be:

$$V_a = i_a R_a + K\dot{\theta}_a \quad (3)$$

Where V_a is the voltage applied to the actuator and R_a is the armature resistance. Given a gearbox with in/out ratio N and efficiency η ,



The motor equation (2) can be expressed in the output coordinates:

$$Ki_a = J_a N \ddot{\theta} + b_a N \dot{\theta} + \frac{\tau}{N\eta}$$

Substituting into equation (3) and solving for i_a :

$$\begin{aligned} i_a &= \frac{J_a N}{K} \ddot{\theta} + b_a N \dot{\theta} + \frac{\tau}{N\eta} \\ V_a &= \frac{R_a J_a N}{K} \ddot{\theta} + \frac{R_a b_a N}{K} \dot{\theta} + \frac{R_a}{KN\eta} \tau + KN\dot{\theta} \end{aligned} \quad (4)$$

Assuming PD control, $V_a = K_p(\theta - \theta_d) + K_d\dot{\theta}$, where θ_d is the desired orientation of the actuator, the following solution is found by setting the PD solution equal to equation (4). After collecting like terms:

$$\frac{R_a J_a N}{K} \ddot{\theta} + \left(\frac{R_a J_a N}{K} - K_d + KN \right) \dot{\theta} - K_p \theta = -K_p \theta_d - \frac{R_a}{KN\eta} \tau \quad (5)$$

The following parameters of the system can be obtained by applying a step input to the system with $\tau = 0$ and measuring the characteristics of it's response. Denoting ζ as the damping ratio and ω_n as the natural frequency of the system,

$$\% \text{ Overshoot} = \left(\frac{\theta_{max} - \theta_{ss}}{\theta_{ss}} \right) \times 100, \quad \zeta = \frac{-\ln(\% \text{OS}/100)}{\sqrt{\pi^2 + \ln^2(\% \text{OS}/100)}}, \quad \omega_n = \frac{\pi}{T_p \sqrt{1 - \zeta^2}}$$

Given θ_{max} , θ_{ss} , and T_p as measured parameters of the system's max output, steady state, and time to peak, respectively.

Refactoring equation (5) and equating with the general solution for a second order system given by $\ddot{\theta} + 2\zeta\omega_n\dot{\theta} + \omega_n^2\theta = \omega_n^2\theta_d$, the following solutions are found:

$$2\zeta\omega_n = \frac{b_a}{J_a} - \frac{KK_d}{R_a J_a N} + \frac{K}{R_a J_a} \quad (6) \qquad \omega_n^2 = \frac{-KK_p}{R_a J_a N} \quad (7)$$

Performing a similar experiment as previously described, except with a known inertial load $\tau = J_m \ddot{\theta}$, the following parameters can be found:

$$\alpha_m \equiv 2\zeta\omega_n = \frac{R_a b_a N^2 \eta - KK_d N \eta + K^2 N^2 \eta}{R_a J_a N^2 \eta + R_a J_m} , \quad \beta_m \equiv \omega_n = -\frac{KK_p N \eta}{R_a J_a N^2 \eta + R_a J_m}$$

$$\begin{bmatrix} 1 & -(\alpha_1 J_1 + \beta_1 J_1) \\ 1 & -(\alpha_2 J_2 + \beta_2 J_2) \\ \vdots & \vdots \end{bmatrix} \begin{bmatrix} \frac{R_a b_a N^2 \eta - KK_d N \eta + K^2 N^2 \eta - KK_p N \eta}{R_a J_a N^2 \eta} \\ \frac{1}{J_a N^2 \eta} \end{bmatrix} = \begin{bmatrix} \alpha_1 + \beta_1 \\ \alpha_2 + \beta_2 \\ \vdots \end{bmatrix} \quad (8)$$

With multiple datasets (varying inertial loads, J_m), the solutions of equation (8) can be found using the least-squares method, yeilding

$$\frac{R_a b_a N - KK_d + K^2 N \eta - KK_p}{R_a J_a N} \quad (9) \qquad \frac{1}{J_a N^2 \eta} \quad (10)$$

Finally, the coefficients of the second order system (11) are known:

$$\underbrace{\left(J_a N^2 \eta \right)}_{1/(10) = C_1} \ddot{\theta} + \underbrace{\left(\frac{R_a b_a N^2 \eta - KK_d N \eta + K^2 N^2 \eta}{R_a} \right)}_{(6)/(10) = C_2} \dot{\theta} - \underbrace{\left(\frac{KK_p N \eta}{R_a} \right)}_{(7)/(10) = C_3} \theta + \underbrace{\left(\frac{KK_p N \eta}{R_a} \right)}_{(7)/(10) = C_3} \theta_d = -\tau \quad (11)$$

The MATLAB code implementing this process can be found in the Appendix (see section 14, p. 46, *Listing 3*). The torque provided by the servo can now be solved for, given the current position (θ), velocity ($\dot{\theta}$), angular acceleration ($\ddot{\theta}$), and desired position (θ_d) are known.

Given the equation of motion for the dynamical response of the system (1), substituting in the solution obtained for the motor dynamics and solving for the acceleration,

$$\left(H + J_a N^2 \eta \right)^{-1} \left[\left(B - \frac{R_a b_a N^2 \eta - KK_d N \eta + K^2 N^2 \eta}{R_a} \right) \dot{\gamma} - \left(\frac{KK_p N \eta}{R_a} \right) (\gamma_d - \gamma) - n \right] = \ddot{\gamma}$$

Where

$$n(\gamma, \dot{\gamma}) = d(\gamma, \dot{\gamma}) + G(\gamma) + C \text{sgn}(\dot{\gamma})$$

System Dynamics

The motor equation (11) gives an expression for the motor torques, however the system dynamics are defined in terms of geometric joint angles. The inclusion of differential drive systems means that the joint angles, γ , do not directly correspond to motor rotations, θ , as shown in *Figure 12*.

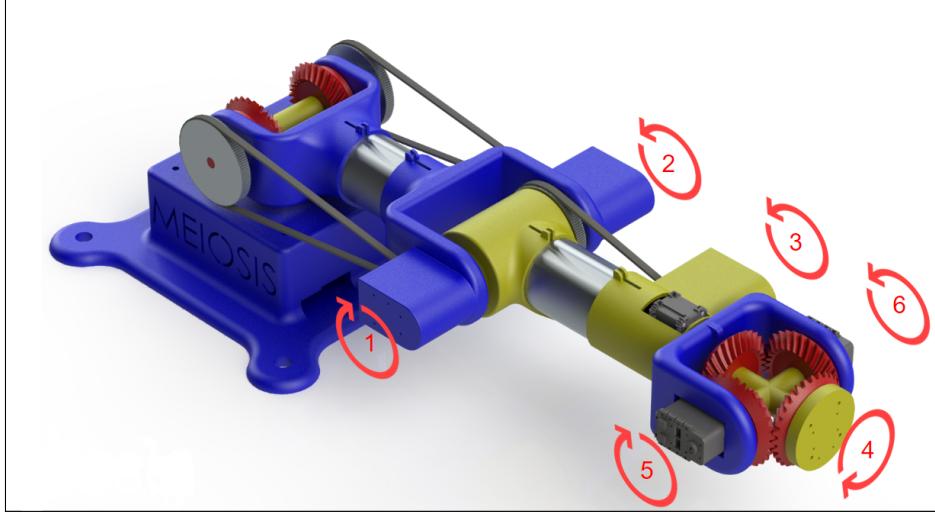


Figure 12: Motor Locations and Orientations

Figure 12 shows the motor positions and relative orientations. This layout was used to define a linear relation between the joint angles and the motor rotations, described in equation (12).

$$\gamma = A\theta \quad \text{where} \quad A = \begin{bmatrix} 1/(2N) & 1/(2N) & 0 & 0 & 0 & 0 \\ 1/(2N) & -1/(2N) & 0 & 0 & 0 & 0 \\ 0 & 0 & -1/N & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1/2 & 1/2 \\ 0 & 0 & 0 & 0 & 1/2 & 1/2 \end{bmatrix} \quad (12)$$

Equation (12) can be used to map the joint angles to the motor angles. The gear ratio of 1:10 is represented by the variable N. Similarly, the motor angles can be determined by multiplying both sides of equation (12) by the inverse of matrix A, giving the following relation.

$$\theta = A^{-1}\gamma \quad (13)$$

It is important to note that the virtual work done by the joint torques (F_γ) and the virtual work done by the motor torques (F_θ) are equal. Using equation (13), a linear relation between

the joint torques and motor torques can be determined.

$$\begin{aligned}
\delta W &= F_\theta^T \delta \theta = F_\gamma^T \delta \gamma, \text{ where } \delta \gamma = A \delta \theta \\
F_\theta^T \delta \theta &= F_\gamma^T (A \delta \theta) \\
F_\theta^T &= F_\gamma^T A \\
(F_\theta^T)^T &= (F_\gamma^T A)^T \\
F_\theta &= A^T F_\gamma \Leftrightarrow F_\gamma = A^{-T} F_\theta
\end{aligned}$$

Using this equation, a relation can be determined between the motor dynamics and the system dynamics given in equation (11) and equation (1) respectively.

$$\begin{aligned}
H(\gamma) \ddot{\gamma} + d(\gamma, \dot{\gamma}) + G(\gamma) &= -A^{-T} (C_1 A^{-1} \ddot{\gamma} + C_2 A^{-1} \dot{\gamma} + C_3 \theta_d - C_3 A^{-1} \gamma) \\
\ddot{\gamma} &= H(\gamma)^{-1} (-A^{-T} (C_1 A^{-1} \ddot{\gamma} + C_2 A^{-1} \dot{\gamma} + C_3 \theta_d - C_3 A^{-1} \gamma) - d(\gamma, \dot{\gamma}) - G(\gamma)) \quad (14)
\end{aligned}$$

Because this equation includes the motor model, which in turn includes an internal PD controller, this equation can be integrated to solve for the system response given a desired motor angle input, θ_d . However, doing so will not result in the desired system response. This control scheme does not have any compensation for the inertia of the links, and it is also lacking gravity compensation. This can be remedied by modifying the input to the motors, θ_d . A new input, u , is defined such that gravity can be compensated. Thus, the motor input term in equation (14) must include both compensation for gravity and the desired motor angle.

$$\begin{aligned}
A^{-T} C_3 u &= G(\gamma) + d(\gamma, \dot{\gamma}) + A^{-T} C_3 \theta_d \\
u &= (A^{-T} C_3)^{-1} (G(\gamma) + d(\gamma, \dot{\gamma})) + \theta_d \quad (15)
\end{aligned}$$

With this new motor input, the closed loop control system equations of motion are given as:

$$\ddot{\gamma} = H(\gamma)^{-1} (-A^{-T} (C_1 A^{-1} \ddot{\gamma} + C_2 A^{-1} \dot{\gamma} + C_3 u - C_3 A^{-1} \gamma) - d(\gamma, \dot{\gamma}) - G(\gamma)) \quad (16)$$

Equation (16) can then be integrated to solve for the system response given desired motor angles.

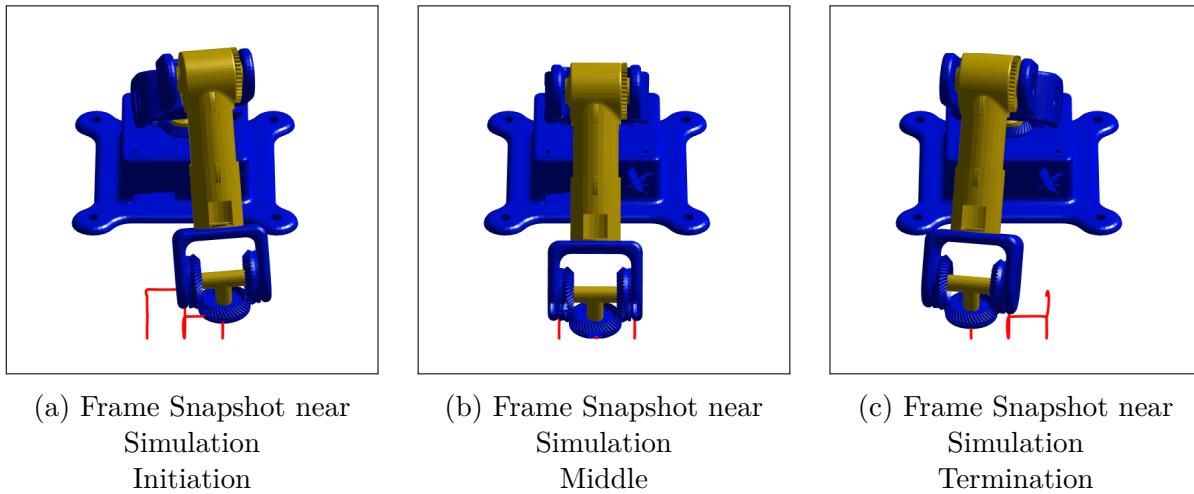


Figure 13: Closed-Loop Control Simulation Animation Snapshots

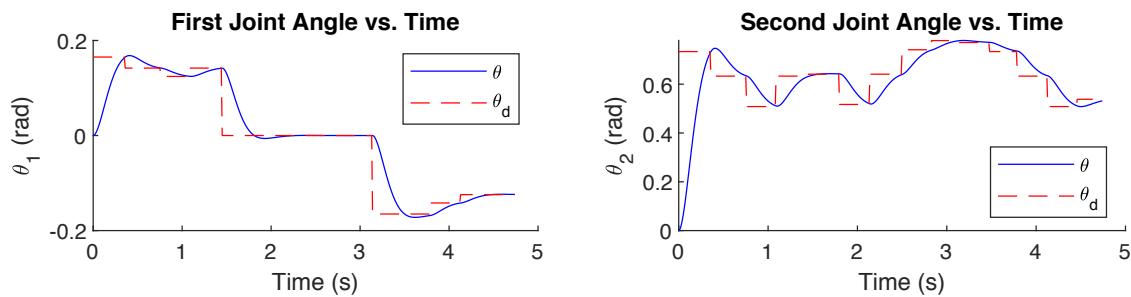


Figure 14: Joint Angles vs Time in Closed-Loop Simulation

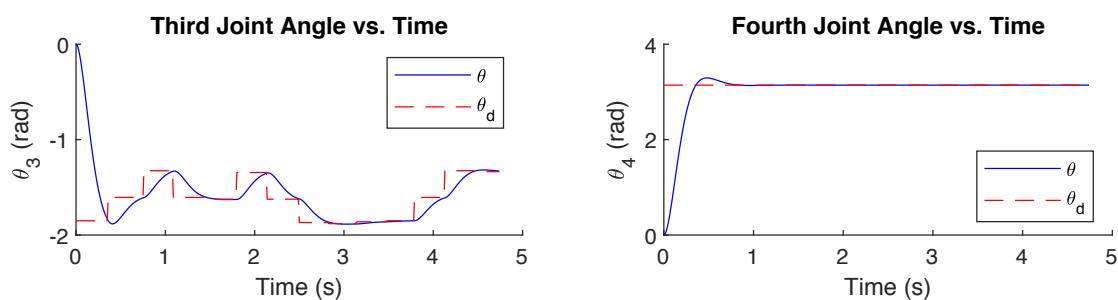


Figure 14 (cont.): Joint Angles vs Time in Closed-Loop Simulation

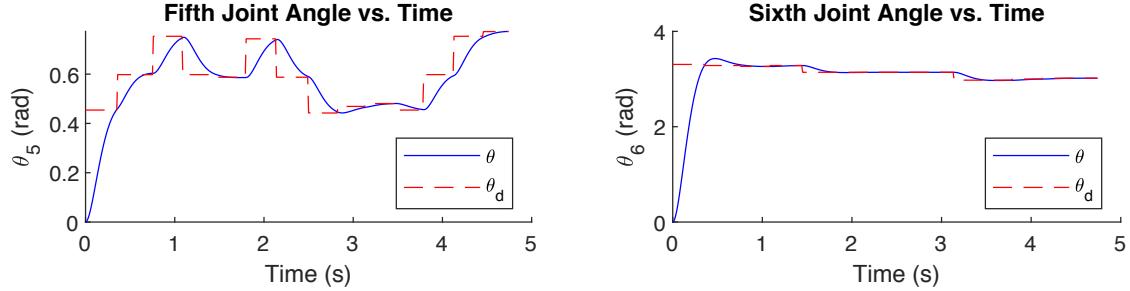


Figure 14 (cont.): Joint Angles vs Time in Closed-Loop Simulation

ANSYS

With 100% infill, 3D printed PLA has a maximum shear stress of 13.6 kpsi. The manipulator applies a load of 13N in the negative y-direction. Without gears in the base differential, the differential support would bear the load on its bearing mounts. *Figure 15* shows the manipulator's differential support could experience up to 97 kPa or 0.014 kpsi of shear stress, which is less than the maximum shear stress of PLA with 100% infill. Since some of the manipulator's mass is supported by the gears, the actual shear experienced by the differential support will be less.

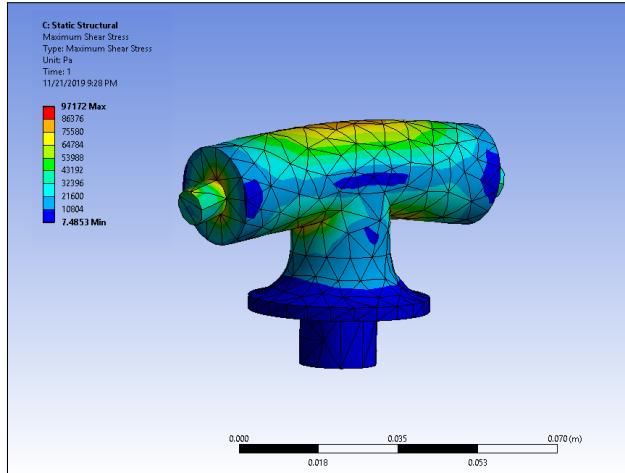


Figure 15: T-Bar ANSYS FEA

To simulate a dynamical loading situation where the manipulator would be under the largest amount of stress, gravitational forces and an outward force (parallel to the arm direction in it's zeroed configuration) were applied to the structure. This situation represents the worst-case loading scenario, such as the manipulator swinging while outstretched. The supports and simulated forces can be seen in the ANSYS image capture shown in *Figure 16*.

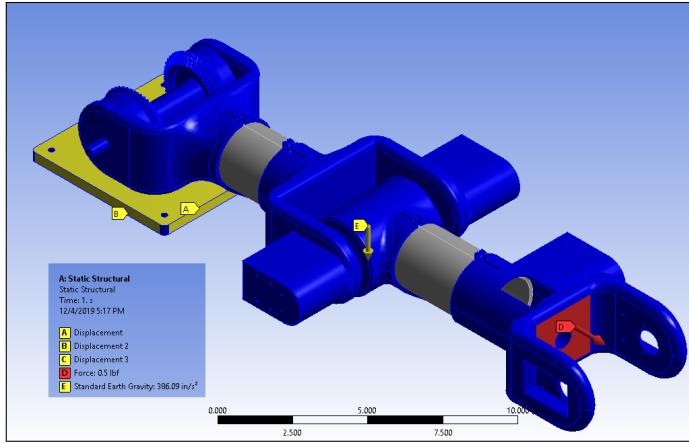
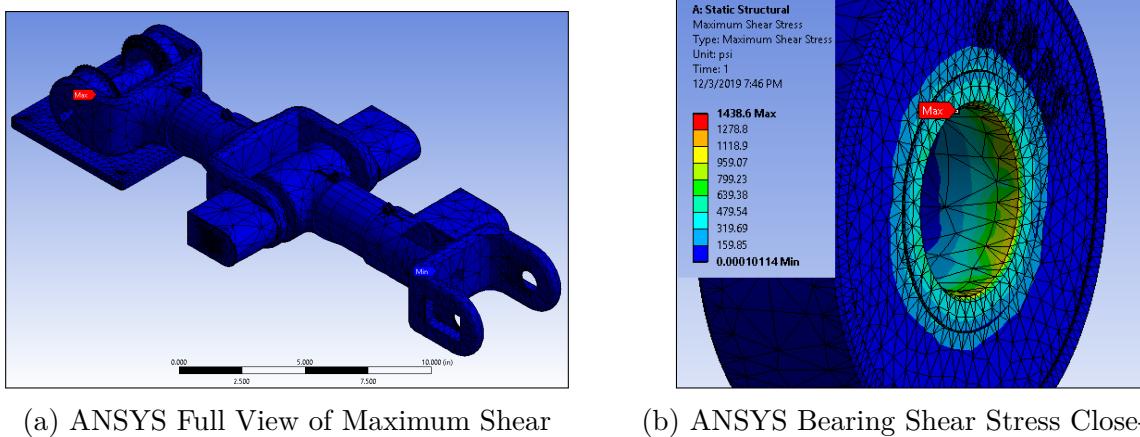


Figure 16: ANSYS Simulated Forces Image Capture

As shown in *Figure 16*, the red arrow is the outward force simulating centrifugal forces, the yellow arrow represents gravity acting at the manipulator's center of mass, and the yellow highlighted faces show the fixed support at the base.

The dynamical loadings resulted in a maximum shear stress at the shoulder differential bearing, as seen in *Figure 17a*; a close-up image of the bearing analysis can be seen in *Figure 17b*.



(a) ANSYS Full View of Maximum Shear

(b) ANSYS Bearing Shear Stress Close-up

Figure 17: ANSYS FEA of Dynamical Loading Scenario

To further validate that the structure is capable of handling alternating stresses, a fatigue test was also performed showing the life of the manipulator handles a minimum of 1e6 cycles, as seen in *Figure 18*, showing it is unlikely to fail due to material yielding.

As seen in *Figure 18*, the lower bearing of the differential drive on the shoulder of the manipulator would be the most likely component to fail under repeating loadings.

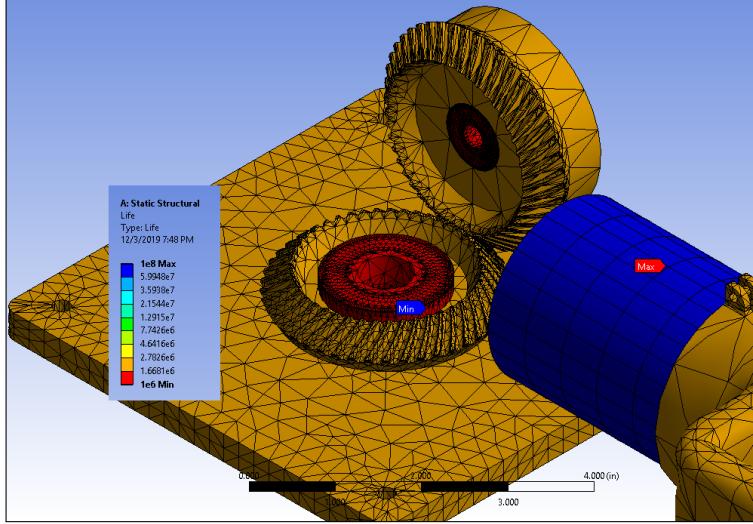


Figure 18: ANSYS Fatigue Test

Parts list and budget

Parts List

Table 5 lists the parts MEIOSIS will require to build the manipulator. The total cost is \$629.22, including shipping. Specification 1.1b requires MEIOSOS's cost to develop the manipulator be less than \$800. In addition to pulley belts for the current configuration, *Table 5* allocates \$20 for two additional belt sizes to increase joint 1/2's and joint 3's torque by a factor of 10. The belts must be purchased in packs of 3 from Automation Direct and two more belt sizes may be required for the two pulley design to allow the servos to be mounted without interfering with the base. *Table 5* also accounts for increased cable lengths of 500 mm to communication bus signals from motor 2 to 3 and motor 3 to 4. And cable lengths of 350 mm to communication bus signals from motor 5 to 6. Motors are identified in [MOTOR ORIENTATION FIGURE]. Aside from electronic hardware, *Table 6* for physical hardware. To allow screw mounting in plastic with metal threads, *Table 6* accounts for 100 threaded press-fit inserts. The manipulator will require between 21 and 46 inserts. The majority of inserts attach MX-12W servos to the manipulator. Mounting hardware accompanies each servo. Since purchasing six sets of pulleys would put MEIOSIS over the \$800 budget, they are printed.

In addition to the costs listed in *Table 5*, *Table 6* shows further costs for the end-user highlighted in blue. Since the Embry Riddle robotics lab has 3D printing available without affecting MEIOSIS's \$800 budget, *Table 6* accounts for outsourced 3D printing costs sufficient to print the entire manipulator with six sets of pulleys. If the end-user owns a 3D printer, the 3D printing cost would effectively reduce to filament cost. Additionally, *Table 6* assumes the end-user does not already possess an AX-12A servo to be used with the end-effector. Further, *Table 6* assumes the manipulator would be more accessible to end-users by using a proprietary U2D2 communication module in lieu of a soldered or bread-board circuit. The

Table 5: MEIOSIS Bill of Materials with Costs

Part	Retailer	Quantity	Unit Cost (USD)	Total Cost (USD)
3 pack, 300 tooth		1	11.5	11.5
3 pack, 208 tooth	Automation Direct	1	9.5	28.5
Base second belts		1	10	10
Link 3 second belts		1	10	10
MX-12W		6	65.9	395.4
500 mm, 1/2 pulleys	Trossen Robotics	2	3.95	7.9
350 mm, 3 pulley		1	2.95	2.95
EE		1	24.95	24.95
Pi 3 B		1	37.99	37.99
Bearings	Amazon	1	8.99	8.99
2 Sch 10 12" Al tube	Industrial Metal Sales	1	2.99	2.99
12 V, 5 V power supply	Digi-Key Electronics	1	43.21	43.21
Automation Direct				0
Trossen Robotics				13.15
Amazon	Shipping	—	—	0
Industrial Metal Sales				26.36
Digi-Key				8.99
			Total	629.22

robotics lab has an AX-12A servo and U2D2 communication module MEIOSIS will use. With the aforementioned additional costs, the MEIOSIS manipulator costs the end-user \$1,007.98 including shipping costs. While \$1,007.98 is slightly above the maximum cost of \$1000 from specification 1.1a, it provides greater accessibility, which may be diminished by assuming end-users possess 3D printers.

Table 6: End-User Bill of Materials with Costs

Part	Retailer	Quantity	Unit Cost (USD)	Total Cost (USD)
Aforementioned Costs	<i>Table 5</i>	—	—	629.22
U2D2	Trossen Robotics	1	49.9	49.9
EE with servo	Trossen Robotics	1	64.95	64.95
3D PLA outsourcing (incl. shipping)	Craft Cloud	1	288.86	288.86
			Total	1007.98

7 Software

Flowcharts

The software the system will need will take in a row vector of position, orientation, path type, and end effector function information for all points to be traveled through and run the manipulator through the desired points following the specified paths requested. To do this, the user will first be asked what the number of points being input will be so that a few data structures can be preallocated. The user will then be asked if the manipulator will be writing or doing pick and place, and store the response. If the user is doing pick and place, the user will be asked for the full point data consisting of the x, y, and z location in millimeters, the phi, theta, and psi angles in degrees, the path type, and the end effector function. If the user is doing writing, only the x, y, and z locations will be requested. The orientation of the point will be defaulted in the “down” orientation since the marker will be held vertically, the path type will be set to cartesian straight line, and the end effector function will be set to stay unchanged. When all the desired points have been input, the software will then create intermediate points every centimeter between points whose paths are specified as cartesian straight line and store the new path points in a different data structure. After the path has been created, each point will be run through inverse kinematics to get the required joint angles to achieve the position and will be stored as the motor data for each point. If the user is writing with the manipulator, the user will be prompted to press a key to close the end effector to grab the marker. The user will then be prompted to press a key to begin, at which point the software will send the motor data to each servo for the first point that the system is trying to reach, wait till the servos are in the desired position, run the end effector function if there is one, and then repeat the process with the next set of motor data until all the points have been traveled through. When the last point has been reached, the program will prompt the user to input the number of desired points and wait for the input to start the process over again. The software the system uses takes in a row vector of position, orientation, path type, and end effector function information for all points to be traveled through and runs the manipulator through the desired points following the specified paths requested. The general overview of the code is shown in *Figure 19*.

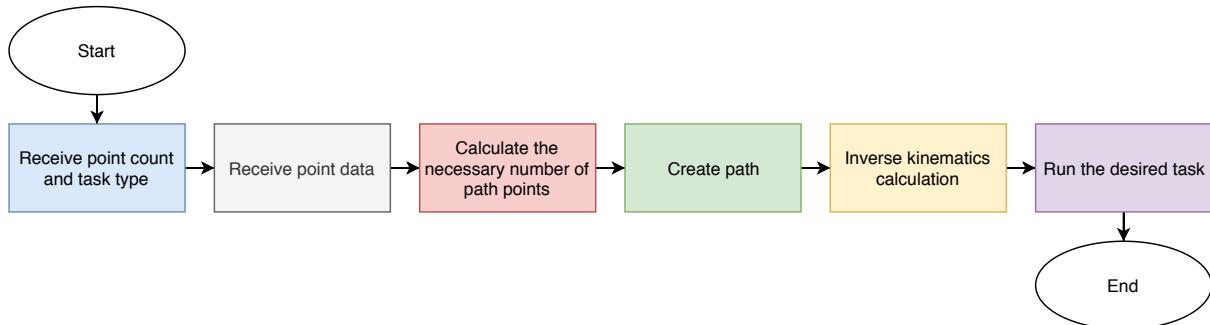


Figure 19: Software Flowchart

Figure 19 shows that the software for the manipulator is broken up into six subsections, two sections that receive data, three that do calculations, and one that runs the specified task.

The first subsection of the software works to receive the number of points the user is inputting as well as the general task the user is completing, shown in *Figure 19a*.

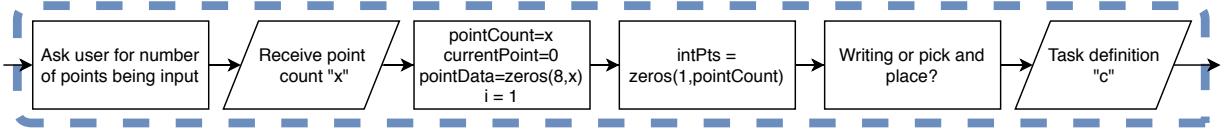


Figure 19a: Software Flowchart Subsection 1

Figure 19a specifies that the software prompts the user for the number of points that the manipulator will travel through and stores the input as a variable, in this case ‘x’. The ‘x’ variable is only used to help preallocate data vectors so that the size of the vector does not change with each input. The software also receives the task specification as either a 0 for cartesian straight line pathing or a 1 for a straight line in the joint space and stores this value in the variable ‘c’.

The second general block in the software flowchart works to receive and store the necessary data for the points the user is inputting depending on the path type as seen in *Figure 19b*.

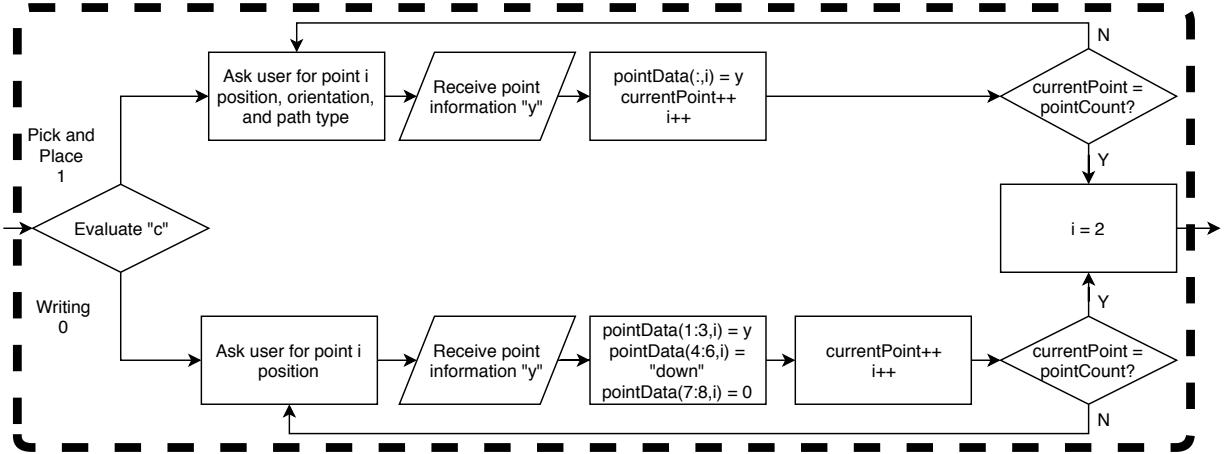


Figure 19b: Software Flowchart Subsection 2

Figure 19b shows that the path type variable ‘c’ is used to determine what information is necessary to collect. If the user is doing a writing task, the software only collects the x, y, and z distances for the point and assumes that the end effector orientation will be facing down so that the marker is vertical. If the user is doing pick and place, the software prompts the user for the x, y, z, phi, theta, psi, path type, and end effector data. The software loops until all the points have been input.

The next block in the software flowchart calculates the total points necessary to complete the task. The overview for this section can be seen in *Figure 19c*.

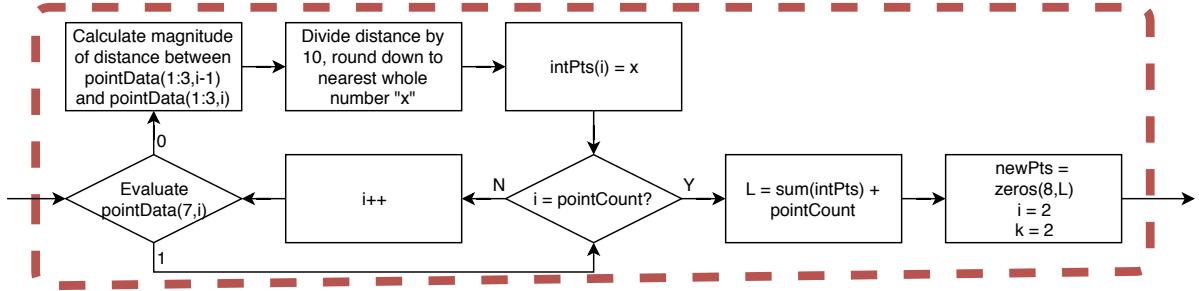


Figure 19c: Software Flowchart Subsection 3

As seen in *Figure 19c*, the section of software calculates the distance between the current point and the previous point if the path type is cartesian straight line and divides the distance by ten to find the number of centimeters between the two points. This value is stored as the necessary number of intermediate points, and the software will loop through every point has been checked. The section of code also stores the total number of points that will be used as the variable level for later use.

The fourth code block in the flowchart creates and stores the necessary intermediate points along the desired path, shown in *Figure 19d*.

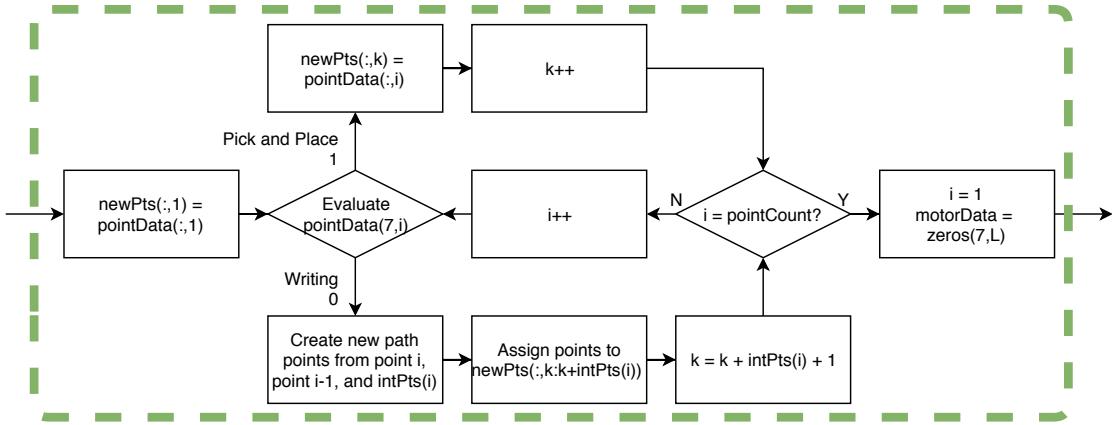


Figure 19d: Software Flowchart Subsection 4

The code shown in *Figure 19d* creates points every centimeter if the path type is cartesian straight line using the number of path points stored for each point from the previous block of software. This ensures that a straight line will be followed between the two user input points. If the path type is a straight line in the joint space, the software does not add any intermediate points since the path seen in the cartesian space does not matter.

The fifth code block in the flowchart calculates inverse kinematics of the points defined in

the previous block of code and stores the angles as counts that can be used by the servos. The overview of this section can be seen in *Figure 19e*.

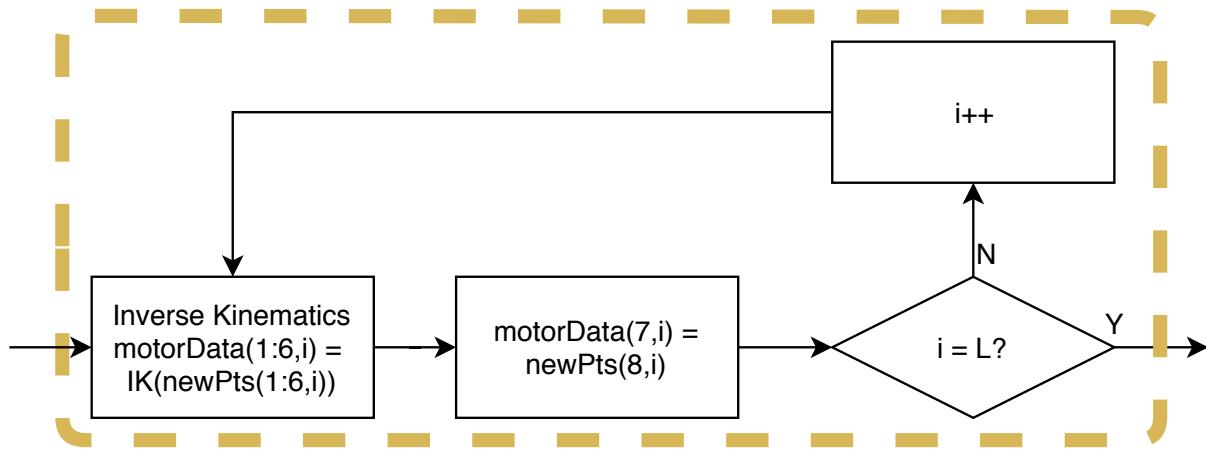


Figure 19e: Software Flowchart Subsection 5

Figure 19e shows that the new points found in the prior section of code are run through an inverse kinematics function that will output the necessary counts the servos can utilize. The code iterates through each point until the inverse kinematics have been calculated for all points.

The final block in the software diagram runs the manipulator through the desired task, with this section of code requiring user input at certain stages depending on the path type, seen in *Figure 19f*.

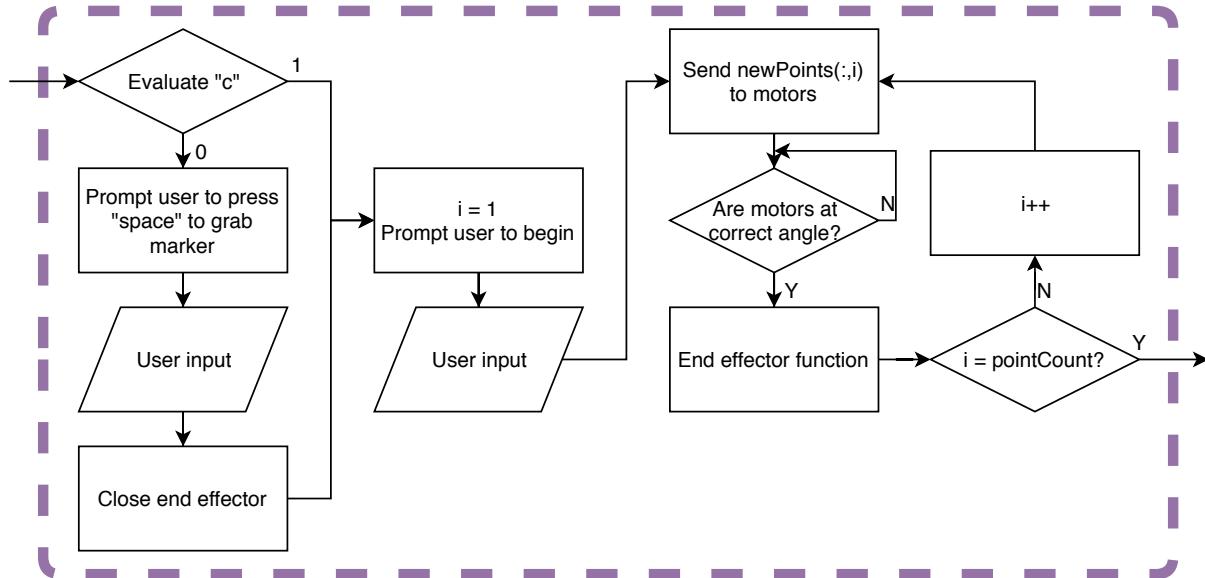


Figure 19f: Software Flowchart Subsection 6

The code in *Figure 19f* prompts the user to press space to close the end effector and grab the marker if drawing was the specified task, otherwise the software jumps straight into prompting the user to begin the task, and when the user begins the task the counts for each position are sent to the servos one at a time. The counts for the next position are not sent to the servos until the servos have reached the desired positions and the end effector function has been completed if there is one.

Description of control system

Implementation

The implementation of the software control algorithm described in the software flowcharts was not able to be finished to completion, although the basic control scheme of the manipulator was established with Python scripts. The MATLAB code written for simulation of the manipulator, such as the inverse kinematics calculations, were converted to Python and successfully implemented. Since only two joints of the manipulator were written, a two link inverse kinematics function was implemented as well as servo control algorithms to ease future programming. Snippets of the servo control scheme as well as the two link arm inverse kinematics can be seen in Listings 1 & 2.

Listing 1: twolink.py

```
def twoLinkIK(x,y):
    l1 = 265.0
    l2 = 165.0
    D = (x**2 + y**2 - l1**2 - l2**2)/(2*l1*l2)
    theta2 = atan2(sqrt(1.0 - D**2),D)
    theta1 = atan2(y,x) - atan2(l2*sin(theta2),l1+l2*cos(theta2))
    print(degrees(theta1),degrees(theta2))
    return [degrees(theta1),degrees(theta2)]
```

The `meiosis_servo.py` file contains several methods composed in an attempt to simplify and expedite future programming.

Listing 2: meiosis_servo.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

"""basic readable servo commands"""

from dynamixel_sdk import *
...
def setJA(self, IDLIST, angle):
    if(type(IDLIST) == int):
        IDLIST = [IDLIST]
    if(type(angle) == int):
```

```

angle = [angle]
for i in range(0,len(IDLIST)):
    self.setPos(IDLIST[i], int(round(angle[i] * gearidx[i]...
        + offset[i])))
    print(int(round(angle[i] * gearidx[i] + offset[i])))

```

The `setJA` method shown above is an example of how a basic function needed for future programming was methodically created in an attempt to ease the burden of the final manipulator implementation. All methods were created with versatility in mind; the testing performed was limited to a two link manipulator since that is all that was physically available, but the underlying programming would remain very similar when all six joints were physically implemented.

Integration

8 Testing

9 Introduction

With the intention of making robotics education more accessible to classrooms, the Manipulator for Educational Institutions with Open Source Integrated Systems (MEIOSIS) aims to provide robotics classes with an accurate and precise manipulator for cost lower than traditional manipulators. MEIOSIS is designed to be 3D printed by the end-user to reduce overall cost of the system and will be modifiable to create an increased understanding of robotics. The following document details the tests performed by the MEIOSIS team and the specific results obtained by each test. This is done in order to ensure that the design requirements are properly met by the finished product.

10 Specifications and Tests

The goal of testing each individual specification is to acquire feedback in order to enhance the design of MEIOSIS. The Specifications and Tests section of the document overviews the tests done and the data/results acquired from the tests.

Specification 1.1.a states: **The cost for the MEIOSIS team to develop the manipulator shall cost no more than \$800.** Team MEIOSIS meets this specification if the cost for the MEIOSIS team to develop the manipulator does not exceed \$800. This specification is tested by adding together the total cost of all parts ordered. This process can be seen in Table 7.

In Table 7, the “Shipping/Tax” row shows the listed shipping fees for the Pi 3 B and associated parts. The “Discounts” row shows the sum total of all the small discounts that were applied to the bulk purchases of multiple items. Table 1 also demonstrates that the total

Table 7: MEIOSIS Bill of Materials with Costs

Product	Quantity	Individual Price	Transaction Cost
M3x30mm Screws	2	\$5.99	\$11.98
M3 Locking nuts	1	\$7.99	\$7.99
TRiREAK AC Bearing	7	\$9.00	\$63.00
Dynamixel MX-12W	3	\$65.90	\$197.70
Thrust Bearing	1	\$7.99	\$7.99
Skateboard Bearings	1	\$5.29	\$5.29
Pi 3 B	1	\$34.95	\$34.95
Power Supply	1	\$9.95	\$9.95
SD Card	1	\$12.95	\$12.95
Shipping/Tax	1	N/A	\$7.95
Discounts	1	N/A	-\$8.17
		Total Cost	\$351.58

cost to develop the manipulator was \$351.58, well under the \$800 allocated for this project, signifying that **Specification 1.1.a was met by the product.**

Specification **1.2.a** states: **The system shall consist of six rotational joints connected by four links. The last three joints will create a spherical wrist.** For this specification to be met, the robot must have six rotational joints and four links. The last three joints must create a spherical wrist where the axis of revolution for the last three joints are intersecting with one another. To perform a test to check for six rotational joints, SolidWorks is required. A rendering of the manipulator with labeled joints can be seen in Figure 20.

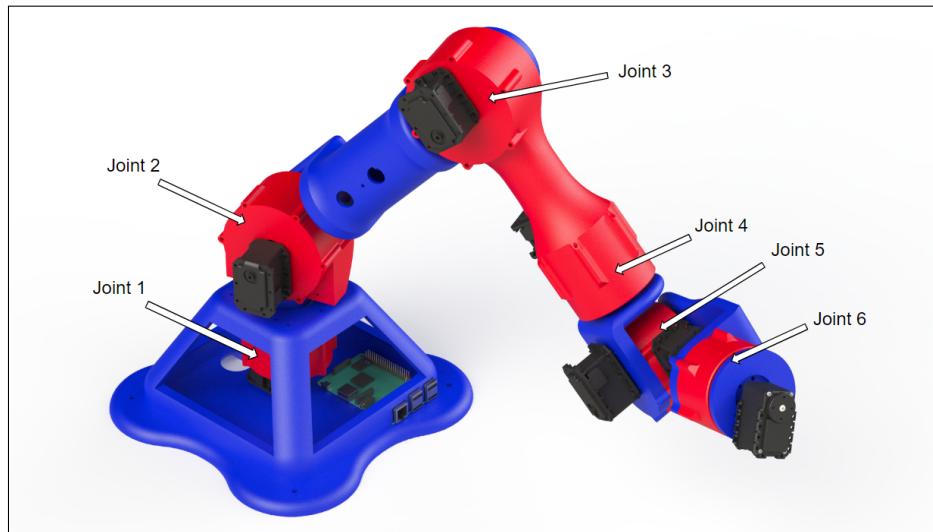


Figure 20: Labeled Image of Manipulator Rendering

As is illustrated in Figure 1, the robot consists of six rotational joints. Joint 1 controls the rotation about the base, joints 2 and 3 control the “shoulder” and “elbow” of the robot,

and joints 4, 5, and 6 control the spherical wrist. The rotational axis of joint 4 intersects with joint 5's and joint 5's intersects with joint 6's rotational axis. To perform this test, the assembly file containing the manipulator was opened and the joints were counted. Using the move tool, each joint can be observed to be exclusively rotational. The data obtained is simulated due to the fact that the robot has not been fully fabricated. **This manipulator meets specification 1.2.a.**

Specification **1.2.b** states: **The system shall have no link offsets.** For this specification to be satisfied, there must be no link offsets present in the design of the robotic manipulator meaning that the link should connect two joints by translating in only one direction. The axes of the joints attached to the link should be collinear with one another when there is no offset present. Testing this specification requires SolidWorks' measurement tool. Using the measurement tool, measure each joint to ensure there is only displacement along one axis. Figure 21 shows an example of a measurement done on the elbow link of the manipulator.

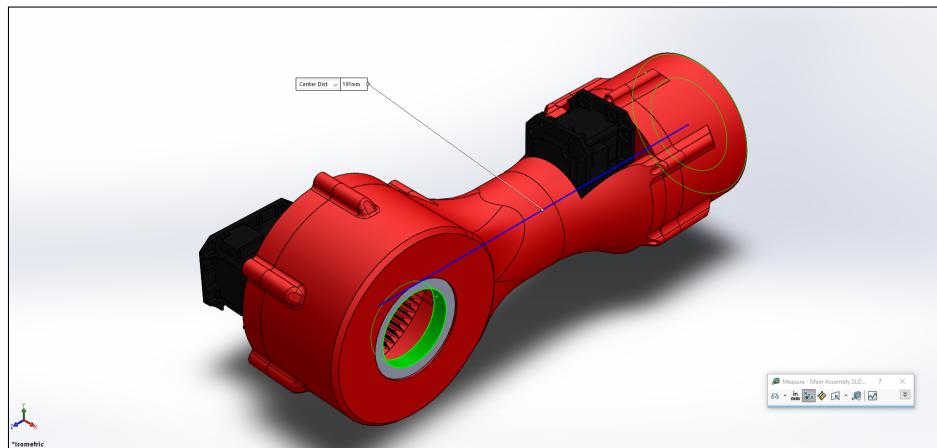


Figure 21: Measurement Test for Link Offset

Figure 21 shows a screen capture of a measurement taken. The blue line, the link only extends across one axis and if there were a link offset present, there would be multiple multi-colored lines to show displacement in multiple directions. These tests were done in simulation on each link of the manipulator and none contained any offset direction. Through this test, it is observed that **the manipulator meets specification 1.2.b.**

Specification **1.3.a** states: **The system shall accommodate a process in which the end user can calibrate the end effector position and orientation to within 0.5 mm and 1 degree of the manipulator's precision.** In order to fulfill specification 1.3.a, an algorithm was developed for calibration. This algorithm must allow the user to manually move the links into a desired zeroed configuration, then accurately retain that zeroed configuration as a reference position during use. Figure 22 depicts the manipulator in a zeroed configuration.

The configuration shown in Figure 22 is what we have defined to be the zeroed configuration. The purpose of the calibration algorithm is to force the software to consider every joint angle

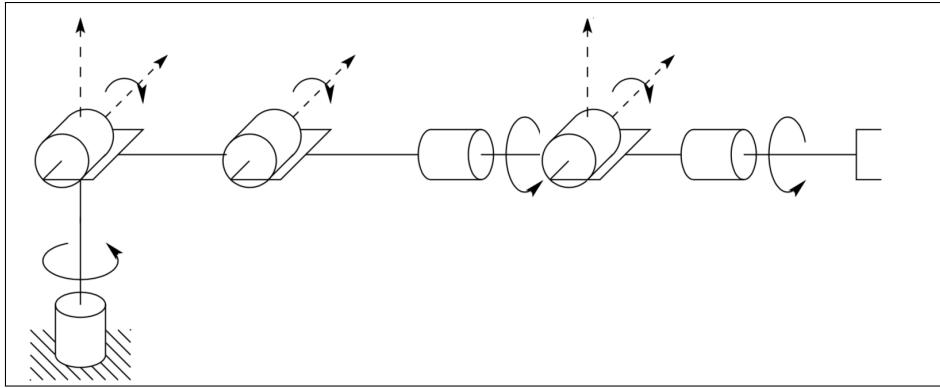


Figure 22: Manipulator in its Zeroed Configuration

to be zero when the manipulator is in this configuration, even if the servos return non-zero angular values. The algorithm developed allows the users to move the joints one by one until the robot is in the desired “zeroed” position. The program then reads the servos positions and stores those values as offset variables. These offsets are subtracted from the desired joint angle values when commanding the servo positions.

In order to measure the calibration accuracy, the PhaseSpace Motion Capture system will be required. The calibration algorithm itself is accurate to the same value as the manipulator, however inaccuracies can arise due to the user not being able to manually put the manipulator into the correct configuration. In completing this testing procedure, it is assumed that the motion capture system will be fully set up, including placing the tracking markers on the manipulator’s base and end-effector. Once the motion capture system is fully set up, the manipulator will be placed in an arbitrary configuration, at least 30° away from zero in each joint. The manipulator will then be restarted, which will erase the system’s memory of its configuration. At this point, the motion capture system will start recording data. Using the calibration software, the manipulator will then be manually zeroed to the best of the operator’s abilities. Once the zeroing is completed, error terms can be determined by comparing the tracking data to the known values of the end-effector’s position and orientation in the zeroed state. Due to circumstances beyond our control, we are not able to perform this test and **adherence to specification 1.3.a cannot be determined**.

Specification 1.4.a states: **Joint one and two of the system shall possess an angle error of no more than 0.025 degrees.** Specification 1.4.b states: **Joint three of the system shall possess an angle error of no more than .03 degrees.** Specification 1.4.c states: **Joints four, five, and six shall possess an angle error of no more than .29 degrees.** In order to verify specifications 1.4.a, 1.4.b, and 1.4.c, the angle accuracy of the system must be measured. To determine the accuracy, the PhaseSpace Motion Capture System is required. Three LED indicators will be placed at the end of each gearbox that is farthest from the previous gearbox in the line to indicate the farthest point directly controlled by the previous gearbox. The three LEDs will be placed at 120° increments around each link/gearbox so that one or more of the LEDs are visible to the motion capture system at

a time. To attain accurate results, small divots will be placed on the manipulator where the LEDs will be located which allows for precise application of the LEDs to the system. Accurate LED placement will allow for more accurate kinematic/angle calculations.

To test the system, the manipulator will be placed in the zeroed configuration and each joint will be tested individually. First, joint 1 will be tested by running from 0° to 90° , and then back to 0° ten times. Each time joint 1 has reached 90° , the motion capture system will collect the location data of the LEDs and store it for error calculation. When this procedure is completed, the data can be used to calculate the angle the link actually achieved, and can be compared to the desired angle. Any error in zeroing will be seen from the collected data. After joint 1 has been tested, joint 2 will be tested following the same procedure, after which joints 3 through 6 will be tested in order. The worst angle error achieved by each joint will be compared to the value defined in the specifications. This test is unable to be performed without lab access and thus **the manipulator's adherence to specifications 1.4a-1.4c cannot be determined**.

Specification **1.5.a** states: **The manipulator shall have a maximum reach between 300 and 700 mm.** In order to satisfy this specification, the robot's links must have a sum length of between 300 mm and 700 mm. Using Solidworks, the total reachable workspace can be measured using the “Measure” tool. This measurement can be seen in Figure 23.

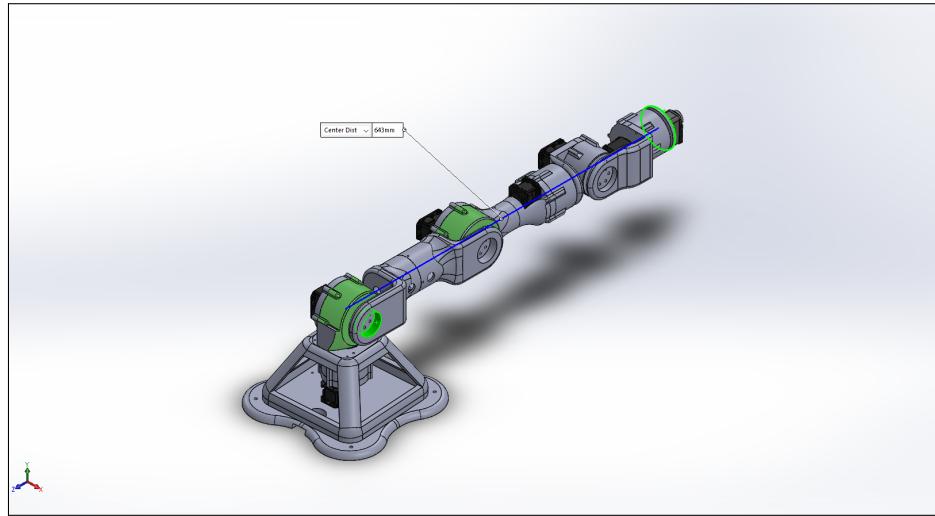


Figure 23: Solidworks Measurement of Reachable Workspace

As can be seen in Figure 23, the total reachable workspace is 643 mm. This measurement is between 300 and 700 millimeters and therefore **the manipulator meets specification 1.5.a.**

Specification **1.5.b** states: **The length of links one, two, three, and the wrist shall be 161.5 mm, 250 mm, 250 mm, and 143 mm respectively.** For the manipulator to meet specification 1.5.a, each link and the wrist simply need to match the lengths provided in the specification. Testing this specification requires the SolidWorks measurement tool.

An example of the measurement tool being used to test this specification can be seen in Figure 24.

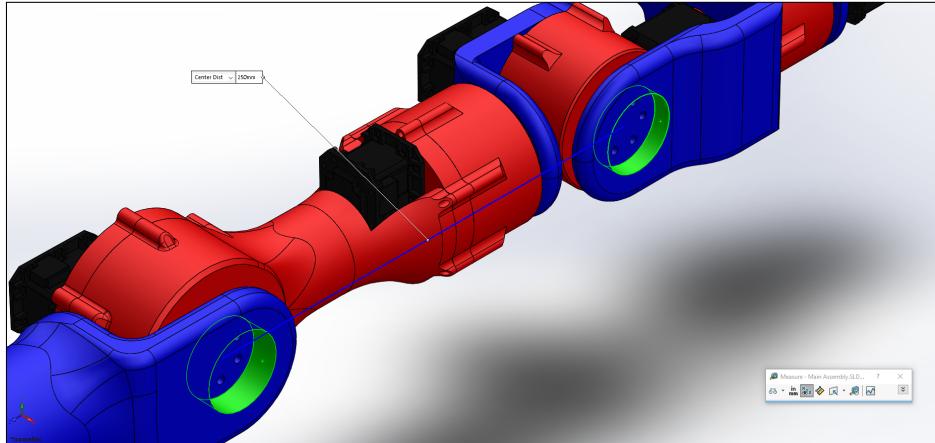


Figure 24: Test of Link Length for Specification 1.5.a.

Once the measurement tool in SolidWorks is selected, a measurement from the first joint in a link to the last joint in a link should be taken. Figure 24 shows a measurement between joints 3 and 5, these joints make up Link 3 or the “elbow” link of the manipulator. The center distance measurement came out to be 250 mm. Once a measurement is done for each link, the center distance values are to be recorded. The variable factors in these measurements are the dimensions of the parts used to make up the link lengths. The lengths of the individual links can be seen in Table 8. These dimensions are simulated in SolidWorks.

Table 8: Measurements of Manipulator Link Lengths

Portion of manipulator measured	Measurement Value
Link 1 (Base to Joint 2)	161.5 mm
Link 2 (Joint 2 to Joint 3)	250 mm
Link 3 (Joint 3 to Joint 5)	250 mm
Wrist (Joint 5 to End Effector)	143 mm

As can be seen by Table 8, the link measurements obtained in SolidWorks match the measurements provided by specification 1.5.a and the **manipulator meets the specification**.

Specification **1.6.a** states: **The system’s workspace must contain a hemispherical shell that has a thickness of 280 mm where the robot is theoretically fully dexterous.** In order to test the dexterity of the manipulator and ensure that it fulfills specification 1.6.a, a program was created that tests a series of points for dexterity. These points are 1 mm apart to match the manipulator’s precision and can be rotated around the z-axis to create a hemispherical shell. The script accomplishes this by calculating the inverse kinematics for the manipulator, given its dimensions. If any point is outside of the manipulator’s workspace it is marked on a MATLAB plot, as can be seen in Figure 25.

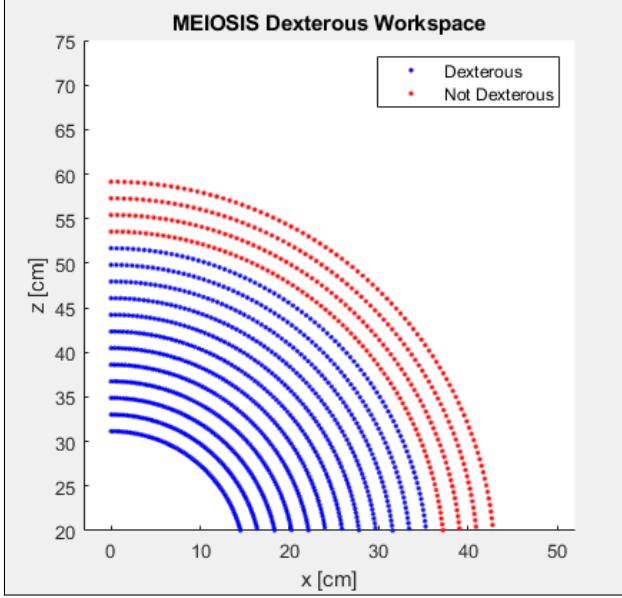


Figure 25: Matlab Plot of Invalid Dexterity Check

Figure 25 shows a failed configuration for our dexterity check. At approximately 35 cm away from the robot it is no longer fully dexterous in this configuration. In order to fix this, changes to the design need to be made, specifically the length of the links. Another fix would be moving the dexterous workspace closer to the base of the robot, however it cannot be moved too close since the base of the robot would be in the way. Figure 26 shows a successful dexterity check.

As can be seen in Figure 26, a successful dexterity check was achieved and the requirement is met. The dexterous workspace spans from 70 mm away from the base to 350 mm away from the base and thus creates a hemispherical workspace with a thickness of 280 mm, thus **the manipulator fulfills specification 1.6.a.**

Specification **1.6.b** states: **The rotational limit of joint one, two, three, four, five, and six shall be -165° to 180° , 13.7° to 179.8° , -179.6° to -19.2° , $\pm 180^\circ$, -180° to 0° , and $\pm 180^\circ$ respectively.** This specification is met if the links can rotate to the maximum and minimum angles provided by the specification. These angles were tested in SolidWorks using the measurement tool. The links to be measured are moved until they collide with one another and then the measurement tool is selected. Once the measurement tool is selected, the two links that form the angle to be measured are selected and the “create sensor” button is pressed. SolidWorks then provides the angle in degrees on the left sidebar. The angles for each joint are then recorded and the results of the measurements can be seen in Table 9.

Table 9 shows that the measurements for joints 3 and 5 do not meet the specification’s individual requirements for them, therefore **the manipulator does not meet specification 1.6.b.** This specification could be met following a major redesign of the system’s link and

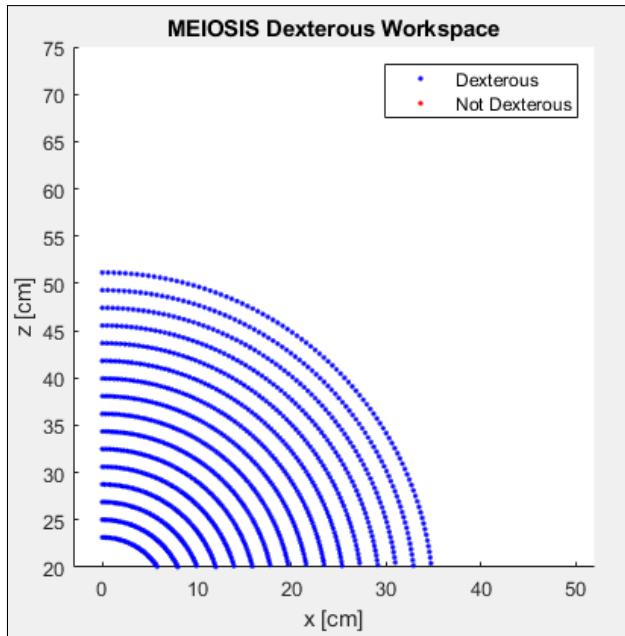


Figure 26: Matlab Plot of Successful Dexterity Check

Table 9: Results of Angle Measurement Test for Specification 2.1.6.a

Joint Number	Minimum Angle	Maximum Angle	Specification Met?
1	-180°	180°	✓
2	-21°	201°	✓
3	-126°	126°	✗
4	-180°	180°	✓
5	-75°	115°	✗
6	-180°	180°	✓

joint system.

Specification **1.7.a** states: **The system shall use a parallel gripper that can close to 18 mm.** To meet specification 1.7.a, a parallel gripper must be designed or purchased that can close to a distance of 18 mm or smaller. Dynamixel offers a parallel gripper actuated by an AX-12A servo as shown in Figure 27. This gripper can be utilized since the manipulator's end effector is designed to attach an AX-12A smart servo.

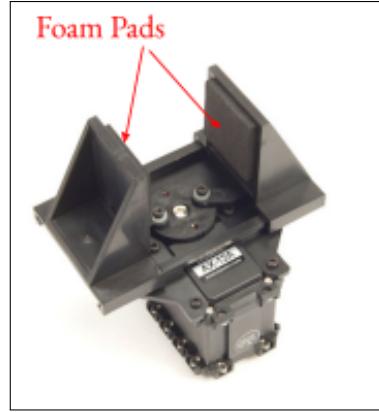


Figure 27: Dynamixel Parallel Gripper with Servo

As depicted in Figure 27, the gripper is open to its widest position. However, the gripper's fingers have removable foam pads. The foam pads allow adjustments in the gripping plasticity and the maximum opening/closure. Figure 28 shows the gripper in various configurations.

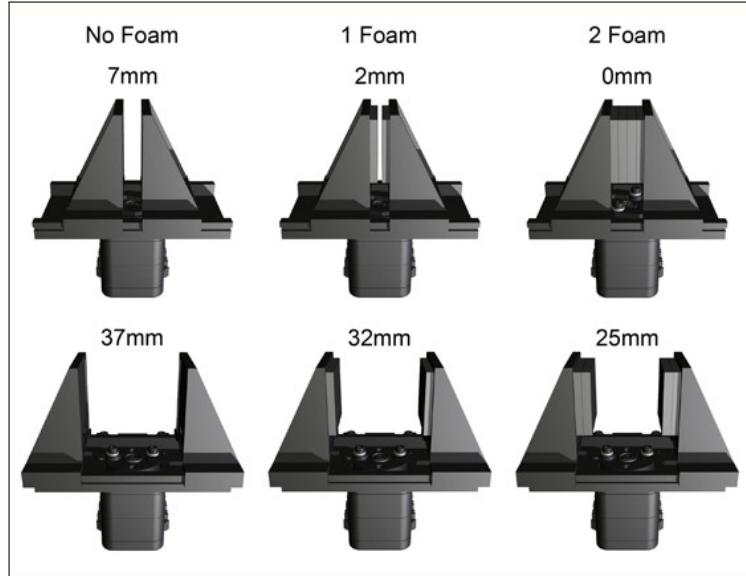


Figure 28: Dynamixel Gripper with and without Foam Padding at Minimum and Maximum Opening

The top three gripper configurations, shown in Figure 28, are fully closed and the bottom

three are fully opened. From left to right, the fingers have no, one, and two pieces of foam. The maximum opening with no foam is 37 mm and the maximum closure is 0 mm with four pieces of foam. While none of these configurations have exactly 18 mm between the fingers the distance between the fingers can be adjusted to accommodate this, as shown in Figure 29.



Figure 29: Dynamixel Gripper Rotational to Linear Motion Mechanism [5]

By sending the servo a position, the two highlighted red half-circular bars seen in Figure 29 pivot about the servo face's edges, translating the fingers either inward or outward. Clockwise rotation would decrease the distance between the fingers, while counterclockwise rotation would increase the distance. The closure's precision would be dictated by the AX-12A's resolution. Since the precision of the 18mm closing distance is not specified, it can be assumed the precision need only be sufficient to prevent the marker from being dislodged during writing, which is addressed in 1.8.a. Therefore, no matter the number of foam pads applied, the gripper can close to 18 mm. With the Dynamixel Parallel gripper being used as an end effector attachment, **the manipulator meets specification 1.7.a**.

Specification **1.7.b** states: **The end effector shall attach to the manipulator using screws configured in a pattern that can accommodate a Dynamixel AX-12A servo.** In order to meet this specification, a Dynamixel AX-12A smart servo must be compatible with the end effector's screw configuration. This specification can simply be tested using screws and a screwdriver. If the smart servo can successfully be screwed onto the end effector, then this specification is met. The design matches measurements acquired in the lab and parts to accommodate an AX-12A were fabricated and assembled in the lab. Since the Dynamixel smart servos have been tested and observed to fit on the parts printed, **the manipulator meets specification 1.7.b**.

Specification **1.8.a** states: **The gripper shall be able to support 0.004 Newton meter moments about the axes normal to its gripping surfaces.** To meet specification 1.8.a, the gripper selected to be attached to the manipulator's end effector should be capable of holding an object that provides 0.004 Newton meter moments about the axes normal to

its gripping surfaces. Using the same Dynamixel gripper seen in Figure 30, a test can be performed

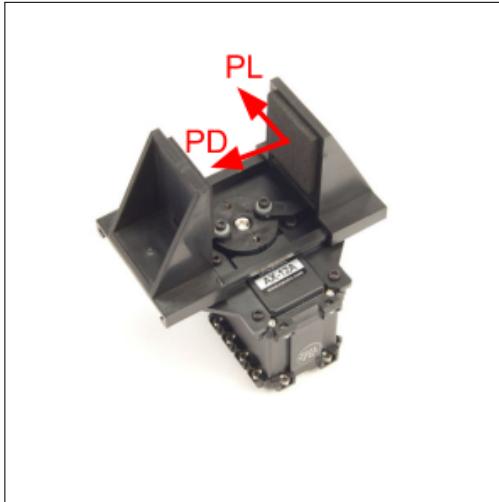


Figure 30: Dynamixel Parallel Gripper with Servo

For testing, we can consider the convenient coordinate frame shown in Figure 30. Specifically, when writing, moments caused by the Expo marker's contact force with paper can be decomposed along the axes of this coordinate frame. In the best case scenario, the marker provides force parallel to and perpendicular to the gripping plane denoted by PL and PD respectively. To test this specification, a set of masses and an Expo marker are required. First, the servo would be commanded to close the gripper on the Expo marker and the masses would be gradually hung from the end of the Expo marker. The moment could be calculated as more mass is added. If the marker is observed to slip or rotate, the moment is too large for the gripper. The gripper would be tested at different angles, grip force, and number of pieces of foam attached, as can be seen in Figure 28. Due to unforeseen circumstances, **adherence to specification 1.8.a cannot be determined.**

Specification **2.1.a** states: **The software shall be hosted publicly on an online repository and maintain an MIT license for distribution.** To satisfy this requirement, the software used to control the robot must be open source and publicly accessible. The use of the code repository GitHub allows team MEIOSIS to easily make the code publicly accessible; by using the MIT licensing, the software can be considered open source and possible legal implications due to misuse can be avoided. The license allows the end-user to modify, redistribute, or do whatever they please with the code: "...without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software..."[10] The MIT license makes it such that any end-user that makes a system with the manipulator, i.e. repurposes the given system for a task, has no limitations with how they wish to redistribute, share, or even sell the system. To implement the use of the license, the source code must contain the proper header, shown by MIT [10]. The header was added and can be observed in the software as well as the code is hosted in a public GitHub repository. Therefore **the system fulfills specification 2.1.a.**

Specification **2.2.a** states: **The system shall have a user interface capable of accepting the end-effector's desired cartesian position and Euler angle orientation as a six element row vector.** In order to satisfy specification 2.2.a, Python scripts have been methodically developed to allow for user input of desired end effector position. These scripts could potentially be used as a backend for a graphical user interface (GUI) to further simplify the process of controlling the manipulator.

In lieu of a Python based GUI directly controlling the manipulator, a GUI could be created to instead control an animated simulation of the manipulator. The simulation is currently generated by a MATLAB script that draws the robot STL files into a figure window. The possibilities of controlling the simulation consist of several options, the most efficient would likely be to develop a user interface entirely with MATLAB. This GUI avoids possible incompatibilities on systems without Python installed and avoids any communication issues that could incur with attempting to create the GUI in another language.

The best option is to have a MATLAB GUI control the MATLAB simulation, therefore a basic interface shall be created in which the user can select which control scheme is desired as well as goal positions for the end effector. A sample GUI can be seen in Figure 31.

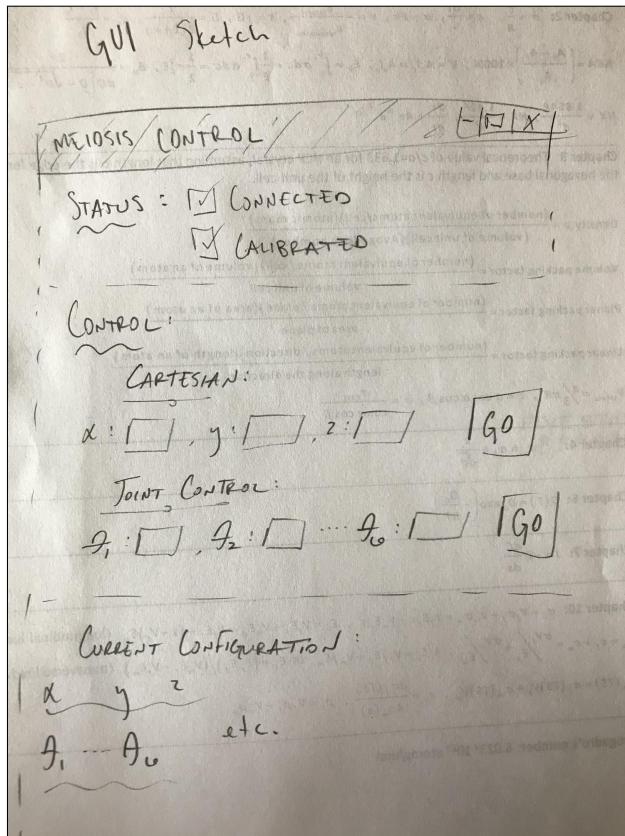


Figure 31: GUI Template

As shown in Figure 31, the GUI will have desired cartesian end effector positions, such as an X, Y, and Z for the end effector position, as well as an end effector orientation input, i.e.,

phi, theta, and psi. The input will closely resemble the input of a six element row vector. The interface will also have user selectable direct joint controls, rather than the system performing the inverse kinematic calculations for the user. The interface will also display status indicators showing whether the robot is calibrated as well as the current manipulator configuration.

In order to test this specification, several points within the workspace will be selected as well as at least three points outside of the workspace for the purpose of error checking. The points (in cartesian XYZ and Euler angles) are chosen randomly. These points are then given to the user interface, which will return joint angle values. The same randomly generated coordinates will be given to the verified kinematics functions, and the outputs will be compared to the output of the UI to check for validity.

Due to uncertainties in the design caused by recent events, development for this user interface has been stopped and verification of its accuracy can not be determined. Therefore, **adherence to specification 2.2.a cannot be determined**.

Specification **2.2.b** states: **The system shall be capable of performing floating point arithmetic.** In order to fulfill specification 2.2.b, a computational system must be selected that can perform floating point arithmetic. The Raspberry Pi 3b+ includes an ARM Cortex A-53 [11], which is capable of performing high-speed floating point arithmetic [1]. Based on this fact, **the manipulator meets specification 2.2b.**

11 Further Testing

In order to determine how useful the harmonic gearboxes are, each gearbox must be fatigue tested. The procedures for both the 1/39 ratio gearbox and the 1/20 gearbox are essentially the same, with the only difference being the load applied to each one. To run the test, each gearbox will be fitted with a load that represents the maximum resting force that the individual gearbox will have to support. For the 1/39 gearbox, the weight applied will be equivalent to the weight the manipulator must support in the shoulder joint. For the 1/20 gearbox, the gearbox will be loaded with the weight the manipulator must support in the elbow joint.

Once the load has been applied, the gearboxes will be set to run repeatedly at full speed from 0° to 180° , then back to 0° to maximize the torque being applied to each gearbox. The test will run for eight hours or until the gearbox breaks, after which data will be collected and analyzed.

In order to collect useful data from the fatigue tests, certain measurements must be taken. Before the fatigue test both the height and width of 4 teeth at 90° angles from each other will be measured and marked, then measured again after the tests to determine the wear the gearbox may go through. A visual inspection of the gearbox will also be conducted to determine if there are any shavings of material in the gearbox.

To test how gearbox fatigue will affect error, the PhaseSpace motion capture system will be utilized to determine any drift from the exact angle. Since the length of each load is known, a PhaseSpace marker can be placed at the end of each load so the motion capture system can see the location of the end of the link. The marker will be placed in a slightly recessed divot on the end of the link in order to make the forward kinematics calculations more exact. Another marker will be placed at the center of the motor cap, again placed within a small divot. These markers will allow us to determine the angle of the load by comparing the positions of the end of the load to the center of rotation. While the fatigue test is running, the motion capture system will capture data at each 0 and 180° position, and the actual angle will be plotted versus time to show any drift or error that occurs.

12 Conclusion

Table 10 summarizes the results of the tests performed throughout the course of this semester. This table marks whether or not each specification was met or if it could not be determined.

Table 10: Summary of Test Results

Specification Tested	Specification Met?
1.1.a	✓
1.2.a	✓
1.2.b	✓
1.3.a	Cannot be Determined
1.4.a	Cannot be Determined
1.4.b	Cannot be Determined
1.4.c	Cannot be Determined
1.5.a	✓
1.5.b	✓
1.6.a	✓
1.6.b	✗
1.7.a	✓
1.7.b	✓
1.8.a	Cannot be Determined
2.1.a	✓
2.2.a	Cannot be Determined
2.2.b	✓

Based on the results of our testing it is not possible to fully determine that the manipulator functions in accordance with the requirements defined. Since specification 1.1.a was met, requirement 1.1, which defines the budget of the robot, is adhered to by the final version of this manipulator. Requirement 1.2 specifies that the manipulator should be fully dexterous without being kinematically redundant. The manipulator is seen in Table 10 to function according to this requirement since both specifications 1.2.a and 1.2.b were tested and met.

Requirements 1.3 and 1.4 both define the necessary precision and accuracy required for this

manipulator to be of educational value. The main goal of our project was a robot that is accurate and precise. Since the tests that would determine whether or not the manipulator adheres to these requirements can only be hypothesized and not actually performed, specifications 1.3.a-1.4.c could not be determined and therefore it is unknown as to whether or not the manipulator functions according to requirements 1.3 and 1.4.

The manipulator functions according to requirement 1.5 since specifications 1.5.a and 1.5.b were both met. This requirement details the reachable workspace that the robot should possess. Requirement 1.6 mentions the dexterous workspace that the robot should have available. This requirement was made to ensure that there is enough movement available for the robot to perform tasks. The manipulator was unable to function according to this requirement. The robot is able to adhere to specification 1.6.a, but without a major redesign it does not meet specification 1.6.b.

The robot was tested to meet specifications 1.7.a and 1.7.b and therefore functions according to requirement 1.7, which specifies that the system's end effector shall be removable and capable of picking and placing a low-odor chisel tip Expo dry erase marker. Since specification 1.8.a could not be determined due to lack of access to the lab it is unsure as to whether the manipulator functions according to requirement 1.8, which mentions that the manipulator should be able to write using the aforementioned Expo marker.

Requirement 2.1, which states that the system shall be open source, has been met since all code and files have been uploaded to a publicly accessible GitHub repository in adherence to specification 2.1.a. However, adherence to requirement 2.2 cannot be determined. Due to the shift out of the lab, there was a large uncertainty as to how the robot would be controlled since there is no longer a physical robot to be moved. This resulted in inconclusive data regarding specification 2.2.a. Despite adhering to specification 2.2.b, it is unknown as to whether or not the manipulator functions according to requirement 2.2.

Overall, since the success of the manipulator largely depended on the accuracy and precision requirements (1.3 and 1.4), it is not possible to say whether or not the manipulator meets the requirements defined in the preliminary design semester.

13 Project Management

14 Conclusion

References

- [1] Cortex-a53 specifications. URL
<https://developer.arm.com/ip-products/processors/cortex-a/cortex-a53>.
- [2] Trossen robotics. URL <https://www.trossenrobotics.com/>.
- [3] Embry-riddle aeronautical university-prescott. URL
<https://www.collegetuitioncompare.com/edu/104586/embry-riddle-aeronautical-university-prescott/>.
- [4] Range of robot cost – robot system cost series. URL
<https://motioncontrolsrobotics.com/range-robot-cost/>.
- [5] PhantomX Parallel Gripper Assembly Guide. URL <https://www.trossenrobotics.com/productdocs/assemblyguides/phantomx-parallel-gripper.html>.
- [6] Sawyer: Rethink robotics unveils new robot. URL <https://spectrum.ieee.org/automaton/robotics/industrial-robots/sawyer-rethink-robotics-new-robot>.
- [7] Pla density. URL
<https://all3dp.com/2/pla-density-what-s-the-density-of-pla-filament-plastic/>.
- [8] Michael Zeltkevic 1. Runge-kutta methods, 04 1998. URL http://web.mit.edu/10.001/Web/Course_Notes/Differential_Equations_Notes/node5.html.
- [9] S. Hutchinson M. Spong and M. Vidyasager. *Robot Modeling and Control*. 2006.
- [10] OSI. The MIT License, 2019. URL <https://opensource.org/licenses/MIT>.
- [11] Raspberry Pi Foundation. Raspberry Pi 3 Model B, 2016. URL
<https://www.raspberrypi.org/products/raspberry-pi-3-model-b/>.
- [12] ROBOTIS. Ax-12a e-manual, 2019. URL
<http://emanual.robotis.com/docs/en/dxl/ax/ax-12a/>.

A Appendix

A.I Salient Code

Listing 3: Actuator Dynamics MATLAB Code

```
1 % dynamixel motor model experiment
2 close all;clear;clc
3
4 % test loads mass moments of inertia
5 m = [.146 .088 0.108]; % mass (kg)
6 b = [.61277 .37227 0.28257]; % length (m)
7 h = [.01915 0.01915 0.0299]; % height (m)
8 J = (1/12)*m.* (h.^2 + b.^2);
9
10 % path to csv files relative to script
11 datapath = 'data/AX12A/';
12 files = dir(strcat(datapath,'*.csv'));
13 numFiles = length(files);
14 % initialize variables
15 [damp, wn, Tp] = deal(zeros(numFiles,1));
16
17 for ii = 1:numFiles
18     % load experimental data, skip 5 header lines
19     M = csvread(strcat(datapath, files(ii).name),5,0);
20     % clean data by removing outliers
21     nani = (find(diff(M(:,1)) > 100));
22     M(nani,:) = [];
23     % show response
24     figure();
25     plot(M(:,1),M(:,2))
26     title('Experimental Data')
27     % find % OS
28     peak = max(M(:,2)); % peak value
29     peaki = find(M(:,2)==peak, 1, 'first'); % peak value index
30     ss = M(end,2); % steady state
31     os = ((peak - ss) / ss) * 100; % % OS
32     % damping ratio
33     damp(ii) = -log(os/100) / sqrt(pi^2 + log(os/100)^2);
34     % find where the motor begins responding
35     start = M(find(diff(M(:,2)) > 1, 1, 'first'), 1);
36     % time to peak
37     Tp(ii) = (M(peaki,1) - start) / 1000;
38     % natural frequency
39     wn(ii) = pi / (sqrt(1 - damp(ii)^2)*Tp(ii));
40 end
```

```

41
42 sol = zeros(4,1);
43 % no load case, 2*zeta*omega_n
44 sol(1) = 2*mean(damp(end-2:end))*mean(wn(end-2:end));
45 % no load case, omega_n^2
46 sol(2) = mean(wn(end-2:end))^2;
47
48 % obtain average damping ratio and natural frequencies for load cases
49 zeta = [mean(damp(1:3));mean(damp(4:6));mean(damp(7:9))];
50 omegan = [mean(wn(1:3));mean(wn(4:6));mean(wn(7:9))];
51
52 alpha = 2.*zeta.*omegan;
53 beta = omegan.^2;
54 A = zeros(3,2);
55 b = zeros(3,1);
56
57 for jj = 1:3
58     A(jj,:) = [1, -(alpha(jj)*J(jj) + beta(jj)*J(jj))];
59     b(jj) = alpha(jj) + beta(jj);
60 end
61
62 sol(3:4) = A \ b;

```

Listing 4: Forward Kinematics MATLAB Function

```

1 function [r6,T6]= MeiosisFK(theta)
2
3 % Mapping between joint space and motor space
4 N = 10; %Gear Ratio
5 A = [ 1/(2*N), 1/(2*N), 0, 0, 0, 0;
6     1/(2*N), -1/(2*N), 0, 0, 0, 0;
7     0, 0,-1/N, 0, 0, 0;
8     0, 0, 0, 1, 0, 0;
9     0, 0, 0, 0,-1/2, 1/2;
10    0, 0, 0, 0, 1/2, 1/2];
11 gamma = A*theta;
12
13 % %Define Constants
14 % LB = 12.275;
15 % L1 = 0;
16 % L2 = 25;
17 % L3 = 20;

```

```

18 % L4 = 7.2;
19 % L5 = 0;
20 % L6 = 5.3;
21
22 %Relative Positions
23 rBfromI = [ 0.00000000; 0.00000000; 0.00000000];
24 r1fromB = [ 0.00000000; 0.00000000; 0.12275000];
25 r2from1 = [ 0.00000000; 0.00000000; 0.00000000];
26 r3from2 = [ 0.00000000; 0.25000000; 0.00000000];
27 r4from3 = [ 0.00000000; 0.20000000; 0.00000000];
28 r5from4 = [ 0.00000000; 0.07000000; 0.00000000];
29 r6from5 = [ 0.00000000; 0.04750000; 0.00000000];
30 %r7from6 = [0; 0; 0]; % dist. from 3rd wrist coor. frame to the end
   ↪ effector is 5.25 cm
31
32 %Orientations wrt I:
33 T1 = rotz(gamma(1));
34 T2 = T1*rotx(gamma(2));
35 T3 = T2*rotx(gamma(3));
36 T4 = T3*roty(gamma(4));
37 T5 = T4*rotx(gamma(5));
38 T6 = T5*roty(gamma(6));
39
40 %Positions wrt I:
41 %rB = rBfromI;
42 r1 = r1fromB;
43 r2 = r1 + T1*r2from1;
44 r3 = r2 + T2*r3from2;
45 r4 = r3 + T3*r4from3;
46 r5 = r4 + T4*r5from4;
47 r6 = r5 + T5*r6from5;
48
49 end

```

Listing 5: Inverse Kinematics MATLAB Function

```

1 function [theta, error] = MeiosisIK(pos,R)
2
3 e0ff = [0;47.5;0];
4 npos = pos - R*e0ff;
5 xc = npos(1);
6 yc = npos(2);

```

```

7      zc = npos(3);
8      L1 = 122.75;
9      d = 0;
10     L2 = 250;
11     L3 = 270;
12
13 % Inverse Position
14 if (xc^2 + yc^2 -d^2) < 0
15     theta = 1000*[1;1;1;1;1;1];
16     error = 1;
17 else
18     t1 = atan2(yc,xc) - atan2(d,sqrt(xc^2 + yc^2 -d^2)) - pi/2;
19     D = (xc^2 + yc^2 - d^2 + (zc - L1)^2 - L2^2 - L3^2)/(2*L2*L3);
20     t3 = atan2(-sqrt(1-D^2),D);
21     t2 = atan2(zc - L1,sqrt(xc^2 + yc^2 - d^2)) - atan2(L3*sin(t3),L2 +
22         ↪ L3*cos(t3));
23
24 % Inverse Orientation
25 T3 = rotz(t1)*rotx(t2)*rotx(t3);
26 T = T3.*R;
27 t6 = atan2(T(2,1),-T(2,3));
28 t4 = atan2(T(1,2),T(3,2));
29 %t4 = atan2(sin(t4),cos(t4));
30
31 if sin(t4) > -10e-6 && sin(t4) < 10e-6
32     t5 = atan2(T(3,2)/cos(t4),T(2,2));
33 else
34     t5 = atan2(T(1,2)/sin(t4),T(2,2));
35 end
36
37 gamma = [t1,t2,t3,t4,t5,t6].';
38
39 % Mapping between joint space and motor space
40 N = 10; %Gear Ratio
41 % B = [ N, N, 0, 0, 0, 0;
42 % N,-N, 0, 0, 0, 0;
43 % 0, 0,-N, 0, 0, 0;
44 % 0, 0, 0, 1, 0, 0;
45 % 0, 0, 0, 0,-1, 1;
46 % 0, 0, 0, 0, 1, 1];
47 A = [ 1/(2*N), 1/(2*N), 0, 0, 0, 0;
48     1/(2*N),-1/(2*N), 0, 0, 0, 0;
49         0, 0,-1/N, 0, 0, 0;
50         0, 0, 0, 1, 0, 0;
51         0, 0, 0, 0,-1/2, 1/2;

```

```

51      0, 0, 0, 0, 1/2, 1/2];
52
53     theta = A\gamma;
54     error = 0;
55 end
56 end

```

Listing 6: Closed Loop Animation

```

1 % Closed Loop Animation
2 % 11/11/2019
3 % Adapted from code written by: Zack Johnson and Edward Pierce
4
5 clear all
6 close all
7 clc
8
9 dt = 0.01; %Integration timestep
10 delta_t = 4; %Animation timestep
11 t = 0:dt:40; %Simulation time array
12 b = zeros(12); %Pre-allocate state variable for all time
13
14 %Define Desired Coordinates and Desired Joint angles
15 [xDes,yDes,zDes,gammad] = Meiosis_Name();
16 r6d = MeiosisFK(gammad(:,1));
17
18 %Define initial conditions
19 b(:,1) = [0;0;0;0;0;0;0;0;0;0;0;0]; %Initial position and velocity
20
21 %Define Control Parameters
22 tolerance = .001*ones(3,1);
23
24 jj = 1;
25
26 for ii = 1:(length(t)-1)
27
28 gamma = b(1:6,ii);
29 r6 = MeiosisFK(gamma);
30 if (abs(r6(3)) < .001)
31     ink(:,ii) = r6; %Keep track of where the marker writes
32     draw(ii) = 1; %Keep track of when the marker writes
33 else

```

```

34     ink(:,ii) = [0;0;0];
35     draw(ii) = 0;
36 end
37 Xe = r6 - r6d;
38
39 if (abs(Xe) <= tolerance)
40     jj = jj + 1;
41     if jj > length(gammad)
42         break
43     end
44     r6d = MeiosisFK(gammad(:,jj));
45 end
46 gammadMat(:,ii) = gammad(:,jj);
47
48 k1 = Meiosis_robot1(b(:,ii),gammad(:,jj));
49 k2 = Meiosis_robot1(b(:,ii) + k1.*dt/2,gammad(:,jj));
50 k3 = Meiosis_robot1(b(:,ii) + k2.*dt/2,gammad(:,jj));
51 k4 = Meiosis_robot1(b(:,ii) + k3.*dt,gammad(:,jj));
52 b(:,ii+1) = b(:,ii) + dt*(k1./6 + k2./3 + k3./3 + k4./6);
53
54
55 end
56
57 % Animation
58 x = ink(1,:);
59 y = ink(2,:);
60 %z = r6Mat(3,:);
61 z = zeros(size(x));
62
63 figure('Name','Closed-Loop Animation','NumberTitle','off')
64 ii = 1;
65 for jj = 1:delta_t:29%(length(b))
66 %for jj = (length(b)-1):length(b)
67 clf(gcf)
68 plot3(x(1:jj),y(1:jj),z(1:jj),'r.')
69 hold on
70 Meiosis_draw2(b(1:6,jj))
71 F(ii) = getframe(gcf);
72 pause(0.001)
73 ii = ii + 1;
74 end
75
76 % % Uncomment the following section to store the animation as a video
77 % writerObj = VideoWriter('DifferentialAnimation.avi');
78 % writerObj.FrameRate = 1/(dt*delta_t);

```

```

79 % open(writerObj);
80 %
81 % % write the frames to the video
82 % for i=1:length(F)
83 % writeVideo(writerObj, F(i));
84 % end
85 %close(writerObj);
86
87 figure('Renderer', 'painters', 'Position', [10 100 700 700], 'Name', 'Joint
    ↪ Angles vs. Time', 'NumberTitle', 'off')
88 Meiosis_Joint_Angle_plot(b(1:6,1:length(gammadMat)), gammadMat, t(1:length(
    ↪ gammadMat)), 'Joint Angles vs. Time')
89 print('JointAnglePlot.pdf', '-dpdf')

```

Listing 7: MEIOSIS Drawing Function

```

1 function Meiosis_draw2(gamma,clr)
2
3 if exist('clr','var')
4     if (clr == 'clear') | (clr == 'Clear')
5         clf(gcf);
6     end
7 end
8
9 Floor_v = [-.3 .4 0;...
10      .3 .4 0;...
11      -.3 .2 0;...
12      .3 .2 0];
13 Floor_f = [1 2 4 3];
14
15 set(gcf, 'Position', [50, 50, 950, 900])
16 hold on
17 patch('Faces',Floor_f,'Vertices',Floor_v,'EdgeColor','None','FaceColor',
    ↪ white');%, 'FaceAlpha',.5);
18
19
20 [Base, Base_f] = stlread('STL/Base.stl');
21 [Link1, Link1_f] = stlread('STL/Link1.stl');
22 [Link2, Link2_f] = stlread('STL/Link2new.stl');
23 [Link3, Link3_f] = stlread('STL/Link3.stl');
24 [Link4, Link4_f] = stlread('STL/Link4.stl');
25 [Link5, Link5_f] = stlread('STL/Link5.stl');

```

```

26 [Link6, Link6_f] = stlread('STL/Link6.stl');
27 [Gear1, Gear1_f] = stlread('STL/Gear1.stl');
28 [Gear2, Gear2_f] = stlread('STL/Gear2.stl');
29 [Gear3, Gear3_f] = stlread('STL/Gear3.stl');
30 [Gear4, Gear4_f] = stlread('STL/Gear4.stl');
31 [Link2tube, Link2tube_f] = stlread('STL/Link2tube.stl');
32 [Link3tube, Link3tube_f] = stlread('STL/Link3tube.stl');
33 [Pulley1, Pulley1_f] = stlread('STL/Pulley1.stl');
34 [Pulley2, Pulley2_f] = stlread('STL/Pulley2.stl');
35 [Link4motors, Link4motors_f] = stlread('STL/Link4motors.stl');
36 [Link6motor, Link6motor_f] = stlread('STL/Link6motor.stl');
37 [Link2belt, Link2belt_f] = stlread('STL/Link2belt.stl');
38 [Link3belt, Link3belt_f] = stlread('STL/Link3belt.stl');

39
40 % Forward kinematics
41 %Relative Positions
42 rBfromI = [ 0.00000000; 0.00000000; 0.00000000];
43 r1fromB = [ 0.00000000; 0.00000000; 0.12275000];
44 r2from1 = [ 0.00000000; 0.00000000; 0.00000000];
45 r3from2 = [ 0.00000000; 0.26000000; 0.00000000];
46 r4from3 = [ 0.00000000; 0.20000000; 0.00000000];
47 r5from4 = [ 0.00000000; 0.07000000; 0.00000000];
48 r6from5 = [ 0.00000000; 0.09600000; 0.00000000];
49 rG1from1 = [ 0.09055000; 0.00000000; 0.00000000];
50 rG2from1 = [-0.09055000; 0.00000000; 0.00000000];
51 rG3from5 = [ 0.04700000; 0.00000000; 0.00000000];
52 rG4from5 = [-0.04700000; 0.00000000; 0.00000000];
53
54 %Orientations wrt I
55 TB = eye(3);
56 T{1} = TB*rotz(gamma(1));
57 T{2} = T{1}*rotx(gamma(2));
58 T{3} = T{2}*rotx(gamma(3));
59 T{4} = T{3}*roty(gamma(4));
60 T{5} = T{4}*rotx(gamma(5));
61 T{6} = T{5}*roty(gamma(6));
62 T{7} = T{1}*rotx(-gamma(1)); %Gear 1
63 T{8} = T{1}*rotx(gamma(1)); %Gear 2
64 T{9} = T{5}*rotx(-gamma(6)); %Gear 3
65 T{10} = T{5}*rotx(gamma(6)); %Gear 4
66
67 %Positions wrt I
68 rB = rBfromI;
69 r{1} = rB + TB*r1fromB;
70 r{2} = r{1} + T{1}*r2from1;

```

```

71 r{3} = r{2} + T{2}*r3from2;
72 r{4} = r{3} + T{3}*r4from3;
73 r{5} = r{4} + T{4}*r5from4;
74 r{6} = r{5} + T{5}*r6from5;
75 r{7} = r{1} + T{1}*rG1from1; %Gear 1
76 r{8} = r{1} + T{1}*rG2from1; %Gear 2
77 r{9} = r{5} + T{5}*rG3from5; %Gear 3
78 r{10}= r{5} + T{5}*rG4from5; %Gear 4
79
80 %Transform the stl coordinates based upon FK
81 Base_v = repmat(rB,1,length(Base)) + Base';
82 Link1_v = repmat(r{1},1,length(Link1)) + T{1}*Link1';
83 Link2_v = repmat(r{2},1,length(Link2)) + T{2}*Link2';
84 Link2tube_v = repmat(r{2},1,length(Link2tube)) + T{2}*Link2tube';
85 Link3_v = repmat(r{3},1,length(Link3)) + T{3}*Link3';
86 Link3tube_v = repmat(r{3},1,length(Link3tube)) + T{3}*Link3tube';
87 Link4_v = repmat(r{4},1,length(Link4)) + T{4}*Link4';
88 Link5_v = repmat(r{5},1,length(Link5)) + T{5}*Link5';
89 Link6_v = repmat(r{6},1,length(Link6)) + T{6}*Link6';
90 Gear1_v = repmat(r{7},1,length(Gear1)) + T{7}*Gear1';
91 Gear2_v = repmat(r{8},1,length(Gear2)) + T{8}*Gear2';
92 Gear3_v = repmat(r{9},1,length(Gear3)) + T{9}*Gear3';
93 Gear4_v = repmat(r{10},1,length(Gear4)) + T{10}*Gear4';
94 Pulley1_v = repmat(r{7},1,length(Pulley1)) + T{7}*Pulley1';
95 Pulley2_v = repmat(r{8},1,length(Pulley2)) + T{8}*Pulley2';
96 Link4motors_v = repmat(r{4},1,length(Link4motors)) + T{4}*Link4motors';
97 Link6motor_v = repmat(r{6},1,length(Link6motor)) + T{6}*Link6motor';
98 Link2belt_v = repmat(r{2},1,length(Link2belt)) + T{2}*Link2belt';
99 Link3belt_v = repmat(r{3},1,length(Link3belt)) + T{3}*Link3belt';
100
101 patch('Faces',Base_f, 'Vertices',Base_v, 'EdgeColor','None','FaceColor'
102     ↪ ,[0 0.082353 1]);
103 patch('Faces',Link1_f,'Vertices',Link1_v,'EdgeColor','None','FaceColor'
104     ↪ ,[0.901961 0.756863 0.035294]);
105 patch('Faces',Link2_f,'Vertices',Link2_v,'EdgeColor','None','FaceColor'
106     ↪ ,[0 0.082353 1]);
107 patch('Faces',Link2tube_f,'Vertices',Link2tube_v,'EdgeColor','None','
108     ↪ FaceColor',[0.5774 0.5774 0.5774]);
patch('Faces',Link3_f,'Vertices',Link3_v,'EdgeColor','None','FaceColor'
    ↪ ,[0.901961 0.756863 0.035294]);
patch('Faces',Link3tube_f,'Vertices',Link3tube_v,'EdgeColor','None','
    ↪ FaceColor',[0.5774 0.5774 0.5774]);
patch('Faces',Link4_f,'Vertices',Link4_v,'EdgeColor','None','FaceColor'
    ↪ ,[0 0.082353 1]);
patch('Faces',Link5_f,'Vertices',Link5_v,'EdgeColor','None','FaceColor'
    ↪ ,[0 0.082353 1]);

```

```

    ↪ ,[0.901961 0.756863 0.035294]);
109 patch('Faces',Link6_f,'Vertices',Link6_v,'EdgeColor','None','FaceColor'
    ↪ ,[1 0.082353 0.082353]);
110 patch('Faces',Gear1_f,'Vertices',Gear1_v,'EdgeColor','None','FaceColor'
    ↪ ,[1 0.082353 0.082353]);
111 patch('Faces',Gear2_f,'Vertices',Gear2_v,'EdgeColor','None','FaceColor'
    ↪ ,[1 0.082353 0.082353]);
112 patch('Faces',Gear3_f,'Vertices',Gear3_v,'EdgeColor','None','FaceColor'
    ↪ ,[1 0.082353 0.082353]);
113 patch('Faces',Gear4_f,'Vertices',Gear4_v,'EdgeColor','None','FaceColor'
    ↪ ,[1 0.082353 0.082353]);
114 patch('Faces',Pulley1_f,'Vertices',Pulley1_v,'EdgeColor','None','
    ↪ FaceColor',[0.5774 0.5774 0.5774]);
115 patch('Faces',Pulley2_f,'Vertices',Pulley2_v,'EdgeColor','None','
    ↪ FaceColor',[0.5774 0.5774 0.5774]);
116 patch('Faces',Link4motors_f,'Vertices',Link4motors_v,'EdgeColor','None',
    ↪ 'FaceColor',[0.5774 0.5774 0.5774]);
117 patch('Faces',Link6motor_f,'Vertices',Link6motor_v,'EdgeColor','None','
    ↪ FaceColor',[0.5774 0.5774 0.5774]);
118 patch('Faces',Link2belt_f,'Vertices',Link2belt_v,'EdgeColor','None','
    ↪ FaceColor',[0 0 0]);
119 patch('Faces',Link3belt_f,'Vertices',Link3belt_v,'EdgeColor','None','
    ↪ FaceColor',[0 0 0]);
120
121 axis equal
122 camlight(180,90)
123 set(gca,'xtick',[])
124 set(gca,'xticklabel',[])
125 set(gca,'XColor','none')
126 set(gca,'ytick',[])
127 set(gca,'yticklabel',[])
128 set(gca,'YColor','none')
129 set(gca,'ztick',[])
130 set(gca,'zticklabel',[])
131 set(gca,'ZColor','none')
132 set(gca,'Color','none')
133
134 set(gca,'projection','perspective')
135 view(140,45)
136 axis([-0.4 0.4 -0.2 0.7 0 0.4])
137 %axis([- .1 .1 -.1 .1 0 .2])
138
139 hold off
140 end

```

Listing 8: H, D, and G Calculations

```

1 function [H, d, G] = Meiosis_HdG(gamma,gammadot)
2
3 % Mass Parameters
4 %Mass(kg)
5 m{1} = 0.13954848; %Link 1
6 m{2} = 1.14358921; %Link 2
7 m{3} = 1.00606970; %Link 3
8 m{4} = 0.26834265; %Link 4
9 m{5} = 0.03933179; %Link 5
10 m{6} = 0.19893937;
11 m{7} = 0.24224704; %Gear 1
12 m{8} = 0.24224704; %Gear 2
13 m{9} = 0.05035795; %Gear 3
14 m{10} = 0.05035795;
15
16 %Center of mass for each link (m)
17 rcm{1} = [ 0.00000000; 0.00000000;-0.02792483]; %Link 1
18 rcm{2} = [ 0.00007758; 0.15623908;-0.00011610]; %Link 2
19 rcm{3} = [-0.01528806; 0.07006082; 0.00110335]; %Link 3
20 rcm{4} = [ 0.00000000; 0.03163485; 0.00000000]; %Link 4
21 rcm{5} = [ 0.00000000; 0.00630115; 0.00000000]; %Link 5
22 rcm{6} = [ 0.00000000;-0.03626349;-0.00034373]; %Link 6
23 rcm{7} = [-0.02271539;-0.00001560; 0.00001565]; %Gear 1
24 rcm{8} = [ 0.02271539;-0.00001560;-0.00001565]; %Gear 2
25 rcm{9} = [-0.00678010; 0.00000000; 0.00000000]; %Gear 3
26 rcm{10}= [ 0.00678010; 0.00000000; 0.00000000]; %Gear 4
27
28 Gam{1} = rcm{1}*m{1};
29 Gam{2} = rcm{2}*m{2};
30 Gam{3} = rcm{3}*m{3};
31 Gam{4} = rcm{4}*m{4};
32 Gam{5} = rcm{5}*m{5};
33 Gam{6} = rcm{6}*m{6};
34 Gam{7} = rcm{7}*m{7};
35 Gam{8} = rcm{8}*m{8};
36 Gam{9} = rcm{9}*m{9};
37 Gam{10} = rcm{10}*m{10};
38
39 %Inertia Matrices
40 J{1} = [ 0.00015462 0.00000000 0.00000000; %Link 1
41           0.00000000 0.00016692 0.00000000;
42           0.00000000 0.00000000 0.00003819];
43 J{2} = [ 0.03894999 -0.00002305 0.00000000; %Link 2

```

```

44      -0.00002305 0.00589287 0.00001585;
45      0.00000000 0.00001585 0.04414151];
46 J{3} = [ 0.01143098 0.00169728 0.00000008; %Link 3
47      0.00169728 0.00185484 -0.00021794;
48      0.00000008 -0.00021794 0.01259229];
49 J{4} = [ 0.00063966 0.00000000 0.00000000; %Link 4
50      0.00000000 0.00069715 0.00000000;
51      0.00000000 0.00000000 0.00108401];
52 J{5} = [ 0.00000851 0.00000000 0.00000000; %Link 5
53      0.00000000 0.00001467 0.00000000;
54      0.00000000 0.00000000 0.00002081];
55 J{6} = [ 0.00036378 0.00000000 0.00000000; %Link 6
56      0.00000000 0.00007212 -0.00000173;
57      0.00000000 -0.00000173 0.00034916];
58 J{7} = [ 0.00014269 -0.00000005 0.00000005; %Gear 1
59      -0.00000005 0.00029726 0.00000000;
60      0.00000005 0.00000000 0.00029726];
61 J{8} = [ 0.00014269 0.00000005 0.00000005; %Gear 2
62      0.00000005 0.00029726 0.00000000;
63      0.00000005 0.00000000 0.00029726];
64 J{9} = [ 0.00003333 0.00000000 0.00000000; %Gear 3
65      0.00000000 0.00001986 0.00000000;
66      0.00000000 0.00000000 0.00001986];
67 J{10}= [ 0.00003333 0.00000000 0.00000000; %Gear 4
68      0.00000000 0.00001986 0.00000000;
69      0.00000000 0.00000000 0.00001986];
70
71
72 % Forward kinematics
73 %Relative Positions
74 rBfromI = [ 0.00000000; 0.00000000; 0.00000000];
75 r1fromB = [ 0.00000000; 0.00000000; 0.12275000];
76 r2from1 = [ 0.00000000; 0.00000000; 0.00000000];
77 r3from2 = [ 0.00000000; 0.26000000; 0.00000000];
78 r4from3 = [ 0.00000000; 0.20000000; 0.00000000];
79 r5from4 = [ 0.00000000; 0.07000000; 0.00000000];
80 r6from5 = [ 0.00000000; 0.09600000; 0.00000000];
81 rG1from1 = [ 0.09055000; 0.00000000; 0.00000000];
82 rG2from1 = [-0.09055000; 0.00000000; 0.00000000];
83 rG3from5 = [ 0.04700000; 0.00000000; 0.00000000];
84 rG4from5 = [-0.04700000; 0.00000000; 0.00000000];
85
86 %Store orientations in cell array
87 rn{1} = r1fromB;
88 rn{2} = r2from1;

```

```

89 rn{3} = r3from2;
90 rn{4} = r4from3;
91 rn{5} = r5from4;
92 rn{6} = r6from5;
93 rn{7} = rG1from1;
94 rn{8} = rG2from1;
95 rn{9} = rG3from5;
96 rn{10} = rG4from5;
97
98 %Orientations wrt I
99 TB = eye(3);
100 T{1} = TB*rotz(gamma(1));
101 T{2} = T{1}*rotx(gamma(2));
102 T{3} = T{2}*rotx(gamma(3));
103 T{4} = T{3}*rot(y(gamma(4)));
104 T{5} = T{4}*rotx(gamma(5));
105 T{6} = T{5}*rot(y(gamma(6));
106 T{7} = T{1}*rotx(gamma(1)); %Gear 1
107 T{8} = T{1}*rotx(-gamma(1)); %Gear 2
108 T{9} = T{5}*rotx(-gamma(6)); %Gear 3
109 T{10} = T{5}*rotx(gamma(6)); %Gear 4
110
111 %Positions wrt I
112 rB = rBfromI;
113 r{1} = rB + TB*r1fromB;
114 r{2} = r{1} + T{1}*r2from1;
115 r{3} = r{2} + T{2}*r3from2;
116 r{4} = r{3} + T{3}*r4from3;
117 r{5} = r{4} + T{4}*r5from4;
118 r{6} = r{5} + T{5}*r6from5;
119 r{7} = r{1} + T{1}*rG1from1; %Gear 1
120 r{8} = r{1} + T{1}*rG2from1; %Gear 2
121 r{9} = r{5} + T{5}*rG3from5; %Gear 3
122 r{10}= r{5} + T{5}*rG4from5; %Gear 4
123
124
125 % Recursive Kinematics
126 Jb0 = zeros(6);
127 Jbdot0 = zeros(6);
128
129 for ii = 1:10
130 if ii == 1
131 Jmat = Jb0;
132 wI = Jmat(1:3,:)*gammadot;
133 [Jb{ii},Jbdot{ii}] = Next_GeoJac_Meiosis(Jb0,Jbdot0,gamma,gammadot,

```

```

    ↪ TB,rn{ii},zeros(3,1),wI,ii);
134 elseif ii == 7
135     Jmat = Jb{1};
136     wI = Jmat(1:3,:)*gammadot;
137     [Jb{ii},Jbdot{ii}] = Next_GeoJac_Meiosis(Jb{1},Jbdot{1},gamma,
138         ↪ gammadot,T{1},rn{ii},zeros(3,1),wI,ii);
139 elseif ii == 8
140     Jmat = Jb{1};
141     wI = Jmat(1:3,:)*gammadot;
142     [Jb{ii},Jbdot{ii}] = Next_GeoJac_Meiosis(Jb{1},Jbdot{1},gamma,
143         ↪ gammadot,T{1},rn{ii},zeros(3,1),wI,ii);
144 elseif ii == 9
145     Jmat = Jb{5};
146     wI = Jmat(1:3,:)*gammadot;
147     [Jb{ii},Jbdot{ii}] = Next_GeoJac_Meiosis(Jb{1},Jbdot{5},gamma,
148         ↪ gammadot,T{5},rn{ii},zeros(3,1),wI,ii);
149 elseif ii == 10
150     Jmat = Jb{5};
151     wI = Jmat(1:3,:)*gammadot;
152     [Jb{ii},Jbdot{ii}] = Next_GeoJac_Meiosis(Jb{1},Jbdot{5},gamma,
153         ↪ gammadot,T{5},rn{ii},zeros(3,1),wI,ii);
154 else
155     Jmat = Jb{ii-1};
156     wI = Jmat(1:3,:)*gammadot;
157     [Jb{ii},Jbdot{ii}] = Next_GeoJac_Meiosis(Jb{ii-1},Jbdot{ii-1},
158         ↪ gamma,gammadot,T{ii-1},rn{ii},zeros(3,1),wI,ii);
159 end
160
161 Jmat = Jb{ii};
162 wI = Jmat(1:3,:)*gammadot;
163
164 Hmat = Jb{ii}.*[J{ii},skew(Gam{ii})*T{ii}.';T{ii}*skew(Gam{ii}).',m{
165     ↪ ii}*eye(3)]*Jb{ii};
166 dvec = Jb{ii}.*[J{ii},skew(Gam{ii})*T{ii}.';T{ii}*skew(Gam{ii}).',m{
167     ↪ ii}*eye(3)]*Jbdot{ii}*gammadot + Jb{ii}.*[cross(wI,J{ii})*wI];T{
168     ↪ ii]*cross(wI,cross(wI,Gam{ii}))];
169 g = Jb{ii}.*[cross(rcm{ii},T{ii}).*[0;0;-m{ii}*9.81]);[0;0;-m{ii
170     ↪ }*9.81]];
171
172 if ii == 1
173     H = Hmat;
174     d = dvec;
175     G = g;
176 else
177     H = H + Hmat;

```

```

169     d = d + dvec;
170     G = G + g;
171   end
172 end
173
174 end

```

Listing 9: Joint Angle Plotting Code

```

1 function Meiosis_Joint_Angle_plot(b,gammad,t,Title)
2
3 % Plot joint angle 1
4 subplot(4,2,1)
5 hold on
6 plot(t, b(1,:),'b')
7 plot(t, gammad(1,:),'r--')
8 title('First Joint Angle vs. Time');
9 xlabel('Time (s)');
10 ylabel('\theta_1 (rad)');
11 legend('\theta','\theta_d')
12
13 % Plot joint angle 2
14 subplot(4,2,2)
15 hold on
16 plot(t, b(2,:),'b')
17 plot(t, gammad(2,:),'r--')
18 title('Second Joint Angle vs. Time');
19 xlabel('Time (s)');
20 ylabel('\theta_2 (rad)');
21 legend('\theta','\theta_d','Location','Southeast')
22
23 % Plot joint 3 movement
24 subplot(4,2,3)
25 hold on
26 plot(t, b(3,:),'b')
27 plot(t, gammad(3,:),'r--')
28 title('Third Joint Angle vs. Time');
29 xlabel('Time (s)');
30 ylabel('\theta_3 (rad)');
31 legend('\theta','\theta_d')
32
33 % Plot joint 4 movement

```

```

34 subplot(4,2,4)
35 hold on
36 plot(t, b(4,:),'b')
37 plot(t, gammad(4,:),'r--')
38 title('Fourth Joint Angle vs. Time');
39 xlabel('Time (s)');
40 ylabel('\theta_4 (rad)');
41 legend('\theta','\theta_d','Location','Southeast')
42
43 % Plot joint 5 movement
44 subplot(4,2,5)
45 hold on
46 plot(t, b(5,:),'b')
47 plot(t, gammad(5,:),'r--')
48 title('Fifth Joint Angle vs. Time');
49 xlabel('Time (s)');
50 ylabel('\theta_5 (rad)');
51 legend('\theta','\theta_d','Location','Southeast')
52
53 % Plot joint 6 movement
54 subplot(4,2,6)
55 hold on
56 plot(t, b(6,:),'b')
57 plot(t, gammad(6,:),'r--')
58 title('Sixth Joint Angle vs. Time');
59 xlabel('Time (s)');
60 ylabel('\theta_6 (rad)');
61 legend('\theta','\theta_d','Location','Southeast')
62
63
64 % annotation('textbox', [0 0.9 1 0.1], ...
65 % 'String', Title, ...
66 % 'EdgeColor', 'none', ...
67 % 'HorizontalAlignment', 'center')
68
69 end

```

Listing 10: MEIOSIS Name Plotting

```

1 function [xDes,yDes,zDes,thetad] = Meiosis_Name2()
2
3 %Define Desired Workspace Coordinates

```

```

4 [xM,yM,zM] = LetterM();
5 [xE,yE,zE] = LetterE();
6 [xI1,yI1,zI1] = LetterI1();
7 [x0,y0,z0] = Letter0();
8 [xS1,yS1,zS1] = LetterS1();
9 [xI2,yI2,zI2] = LetterI2();
10 [xS2,yS2,zS2] = LetterS2();
11
12 xDes = [xM,xE,xI1,x0,xS1,xI2,xS2];
13 yDes = [yM,yE,yI1,y0,yS1,yI2,yS2];
14 zDes = [zM,zE,zI1,z0,zS1,zI2,zS2];
15
16 R = rotx(-pi/2);
17
18
19 %Calculate Joint Angles
20 for ii = 1:length(xDes)
21     [thetad(:,ii),errorCheck] = MeiosisIK([xDes(ii);yDes(ii);zDes(ii)],R);
22     if errorCheck == 1
23         break
24     end
25 end
26 end
27
28 function [x,y,z] = LetterM()
29
30 %Define Desired Workspace Coordinates
31 x = [ 270, 270, 240, ];
32 y = [ 290, 290, 250, ];
33 z = [ 10, 0, ];
34
35 end
36
37 function [x,y,z] = LetterE()
38
39 %Define Desired Workspace Coordinates
40 x = [210, 210, 210, 150, 150, 200, 200, 170, 170, 200, 200, 150, 150,
41     ↪ 210, 210];
42 y = [260, 260, 340, 340, 330, 330, 300, 300, 290, 290, 270, 270, 260,
43     ↪ 260, 260];
44 z = [ 10, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 10];
45
46 function [x,y,z] = LetterI1()

```

```

47
48 %Define Desired Workspace Coordinates
49 x = [130, 130, 130, 110, 110, 130, 130, 80, 80, 100, 100, 80, 80, 130,
      ↪ 130];
50 y = [260, 260, 270, 270, 330, 330, 340, 340, 330, 330, 270, 270, 260,
      ↪ 260, 260];
51 z = [ 10, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 10];
52
53 end
54
55 function [x,y,z] = LetterO()
56
57 %Define Desired Workspace Coordinates
58 x = [ 60, 60, 60, 0, 0, 60, 60, 50, 50, 50, 10, 10, 50, 50];
59 y = [260, 260, 340, 340, 260, 260, 260, 270, 270, 330, 330, 270, 270,
      ↪ 270];
60 z = [ 10, 0, 0, 0, 0, 0, 10, 10, 0, 0, 0, 0, 0, 0, 10];
61
62 end
63
64 function [x,y,z] = LetterS1()
65
66 %Define Desired Workspace Coordinates
67 x = [-20, -20, -20, -50, -50, -20, -20, -60, -60, -30, -30, -60,
      ↪ -60, -20, -20];
68 y = [260, 260, 300, 300, 330, 330, 340, 340, 290, 290, 270, 270, 260,
      ↪ 260, 260];
69 z = [ 10, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 10];
70
71 end
72
73 function [x,y,z] = LetterI2()
74
75 %Define Desired Workspace Coordinates
76 [x,y,z] = LetterI1();
77 x = x - 210;
78
79 end
80
81 function [x,y,z] = LetterS2()
82
83 %Define Desired Workspace Coordinates
84 [x,y,z] = LetterS1();
85 x = x - 130;
86

```

87 end

Listing 11: MEIOSIS Robot State Calculator

```
1 function bdot = Meiosis_robot1(b,gammad)
2
3 %Preallocate variables
4 bdot = zeros(12,1);
5
6 %Variable reassigments
7 gamma = b(1:6);
8 gammadot = b(7:12);
9
10 [H,d,G] = Meiosis_HdG(gamma,gammadot);
11 C1 = diag((1 / 82.5459264810444)*[1 1 1 1 1 1]);
12 C2 = diag((13.4787019763844 / 82.5459264810444)*[1 1 1 1 1 1]);
13 C3 = diag((88.3252169803572 / 82.5459264810444)*[1 1 1 1 1 1]);
14
15 N = 10; %Gear Ratio
16 A = [ 1/(2*N), 1/(2*N), 0, 0, 0, 0;
17      1/(2*N), -1/(2*N), 0, 0, 0, 0;
18      0, 0,-1/N, 0, 0, 0;
19      0, 0, 0, 1, 0, 0;
20      0, 0, 0, 0,-1/2, 1/2;
21      0, 0, 0, 0, 1/2, 1/2];
22
23 bdot(1:6) = b(7:12);
24 u = (A.\C3)\(G + d) + A\gammad;
25 bdot(7:12) = (H + A.\C1/A)\(-d - G - A\C2/A*gammadot - A.\C3/A*gamma
26      ↪ + A.\C3*u);
27 end
```

Listing 12: MEIOSIS Velocity Calculator

```
1 function [dotr,w] = MeiosisVelocity(T,r,gamma,dotgamma)
2
3 dotr = simplify(jacobian(r, gamma)*dotgamma);
```

```

5 Jw = [T(:,3).*jacobian(T(:,2),gamma);...
6     T(:,1).*jacobian(T(:,3),gamma);...
7     T(:,2).*jacobian(T(:,1),gamma)];
8
9 M = simplify(Jw,'steps',200,'criterion','preferReal');
10
11 w = simplify(M*dotgamma);
12
13 end

```

Listing 13: Geometric Jacobian Calculator

```

1 function [JbN,JbNdot] = Next_GeoJac_Meiosis(Jb,Jbdot,gamma,gammadot,TI,rn,
2     ↪ rndot,wI,link)
3
4 Ihat = zeros(3,length(gamma));
5 Itilda = zeros(3,length(gamma));
6
7 %link = 8 corresponds to the screen
8 %
9 switch link
10 case 1
11     Ihat(3,1) = 1;
12     Tn = rotz(gamma(1));
13 case 2
14     Ihat(1,2) = 1;
15     Tn = rotx(gamma(2));
16 case 3
17     Ihat(1,3) = 1;
18     Tn = rotx(gamma(3));
19 case 4
20     Ihat(2,4) = 1;
21     Tn = roty(gamma(4));
22 case 5
23     Ihat(1,5) = 1;
24     Tn = rotx(gamma(5));
25 case 6
26     Ihat(2,6) = 1;
27     Tn = roty(gamma(6));
28 case 7 %Gear 1
29     Ihat(1,1) = 1;
30     Tn = rotx(gamma(1));

```

```

30     case 8 %Gear 2
31         Ihat(1,1) = 1;
32         Tn = rotx(-gamma(1));
33     case 9 %Gear 3
34         Ihat(1,1) = 1;
35         Tn = rotx(-gamma(6));
36     case 10 %Gear 4
37         Ihat(1,1) = 1;
38         Tn = rotx(gamma(6));
39
40 end
41
42 %GeoJac Matrices
43 mat1 = [Tn.',zeros(3);-TI*skew(rn),eye(3)];
44 mat2 = [Ihat;TI*Itilda];
45
46 JbN = mat1*Jb + mat2;
47
48 %GeoJacdot matrices
49 mat1 = [ -skew(Ihat*gammadot)*(Tn.'), zeros(3);
50           -TI*skew(wI)*skew(rn) - TI*skew(rndot), zeros(3)];
51 mat2 = [ Tn.', zeros(3);
52           -TI*skew(rn), eye(3)];
53 mat3 = [ zeros(3,6);
54           TI*skew(wI)*Itilda];
55
56 JbNdot = mat1*Jb + mat2*Jbdot + mat3;
57
58 end

```

Listing 14: Open-Loop Animation Code

```

1 % Open Loop Animation
2 % 11/11/2019
3 % Adapted from code written by: Zack Johnson and Edward Pierce
4
5 clear all
6 close all
7 clc
8
9 dt = 0.001; %Integration timestep
10 fr = 24; %Animation Framerate

```

```

11 delta_t = round(1/(dt*fr)); %Animation timestep
12 t = 0:dt:30; %simulation time array
13 b = zeros(12,length(t)); %pre-allocate motor angles for all time
14 Va = zeros(6,length(t)); %pre-allocate input torque for all time
15
16 %Define initial conditions
17 b(:,1) = [0;0;0;0;0;0;0;0;0;0;0;0]; %Initial position and velocity
18
19 for ii = 1:(length(t)-1)
20 k1 = Meiosis_robot1(b(:,ii),Va(:,ii));
21 k2 = Meiosis_robot1(b(:,ii) + k1.*dt/2,Va(:,ii));
22 k3 = Meiosis_robot1(b(:,ii) + k2.*dt/2,Va(:,ii));
23 k4 = Meiosis_robot1(b(:,ii) + k3.*dt,Va(:,ii));
24 b(:,ii+1) = b(:,ii) + dt*(k1./6 + k2./3 + k3./3 + k4./6);
25 ii/length(t)
26 end
27
28 % for ii = delta_t:delta_t:(length(b))
29 jj = 1;
30 for ii = 1:delta_t:length(b)
31 Meiosis_draw2(b(1:7,ii),'clear',t(ii))
32 F(jj) = getframe(gcf);
33 jj = jj + 1;
34 end
35
36 % Uncomment the following section to store the animation as a video
37 writerObj = VideoWriter('OpenLoopAnimation.avi');
38 writerObj.FrameRate = fr;
39 open(writerObj);
40
41 % write the frames to the video
42 for i=1:length(F)
43 writeVideo(writerObj, F(i));
44 end
45 close(writerObj);
46
47 figure('Renderer', 'painters', 'Position', [10 100 700 700], 'Name', 'Joint
    → Angles vs. Time','NumberTitle','off')
48 Meiosis_Joint_Angle_plot(b(1:6,:),t(1:length(b)), 'Joint Angles vs. Time')
49 print('JointAnglePlot.pdf', '-dpdf')

```

Listing 15: Velocity Kinematics Code

```

1 clear all
2 close all
3 clc
4
5 %Define Symbolic Variables
6 syms t1 t2 t3 t4 t5 t6 dt1 dt2 dt3 dt4 dt5 dt6 L1 L2 L3 L4 L5 L6
7 q = [t1 t2 t3 t4 t5 t6].';
8 dotq = [dt1 dt2 dt3 dt4 dt5 dt6].';
9
10 %Relative Positions
11 rBfromI = [ 0; 0; 0];
12 r1fromB = [ 0; 0; L1]; % Frame 1 is 122.75mm away from the Base frame in the
    ↪ Z direction
13 r2from1 = [ 0; 0; 0]; % Frame 2 is centered on Frame 1
14 r3from2 = [ 0; L2; 0]; % Frame 3 is 250mm away from Frame 2 in the Y
    ↪ direction
15 r4from3 = [ 0; L3; 0]; % Frame 4 is 200mm away from Frame 3 in the Y
    ↪ direction
16 r5from4 = [ 0; L4; 0]; % Frame 5 is 70mm away from Frame 4 in the Y
    ↪ direction
17 r6from5 = [ 0; L6; 0]; % Frame 6 is 47.5mm away from Frame 5 in the Y
    ↪ direction
18
19 %Orientations wrt I:
20 T(:,:,1) = rotz(t1);
21 T(:,:,2) = T(:,:,1)*rotx(t2);
22 T(:,:,3) = T(:,:,2)*rotx(t3);
23 T(:,:,4) = T(:,:,3)*roty(t4);
24 T(:,:,5) = T(:,:,4)*rotx(t5);
25 T(:,:,6) = T(:,:,5)*roty(t6);
26
27 %Positions wrt I:
28 rB = rBfromI;
29 r(:,1) = rB + r1fromB;
30 r(:,2) = r(:,1) + T(:,:,1)*r2from1;
31 r(:,3) = r(:,2) + T(:,:,2)*r3from2;
32 r(:,4) = r(:,3) + T(:,:,3)*r4from3;
33 r(:,5) = r(:,4) + T(:,:,4)*r5from4;
34 r(:,6) = r(:,5) + T(:,:,5)*r6from5;
35
36
37 for ii = 1:6
38     [dotr(:,ii),w(:,ii)] = MeiosisVelocity(T(:,:,ii),r(:,ii),q,dotq);

```

```
39     disp(ii)
40     disp(dotr(:,ii))
41     disp(w(:,ii))
42
43 end
```

Listing 16: meiosis_kinematics.py

```
import numpy as np
import math as m
from meiosis_utils import matrixOp as mf

r1 = np.matrix([[0.00000],[0.00000],[0.12275]])
r2 = np.matrix([[0.00000],[0.00000],[0.00000]])
r3 = np.matrix([[0.00000],[0.26000],[0.00000]])
r4 = np.matrix([[0.00000],[0.20000],[0.00000]])
r5 = np.matrix([[0.00000],[0.07000],[0.00000]])
r6 = np.matrix([[0.00000],[0.09600],[0.00000]])

def FK(gamma):
    T1 = mf.rotz(gamma[0])
    T2 = T1*mf.rotx(gamma[1])
    T3 = T2*mf.rotx(gamma[2])
    T4 = T3*mf.roty(gamma[3])
    T5 = T4*mf.rotx(gamma[4])
    T6 = T5*mf.roty(gamma[5])

    Ir1 = r1
    Ir2 = Ir1 + T1*r2
    Ir3 = Ir2 + T2*r3
    Ir4 = Ir3 + T3*r4
    Ir5 = Ir4 + T4*r5
    Ir6 = Ir5 + T5*r6

    return Ir6

def IK(pos,R):
    pi = m.pi
    L1 = r1[2]
    L2 = r3[1]
    L3 = r4[1] + r5[1]
```

```

npos = pos - R*r6
xc = npos[0]
yc = npos[1]
zc = npos[2]

#Inverse Position
t1 = m.atan2(xc,yc) - pi/2
if t1 < -pi:
    t1 = t1 + 2*pi
if t1 > pi:
    t1 = t1 - 2*pi
D = (xc**2 + yc**2 + (zc - L1)**2 - L2**2 - L3**2)/(2*L2*L3)
t3 = m.atan2(-m.sqrt(1 - D**2),D)
t2 = m.atan2(zc - L1, m.sqrt(xc**2 + yc**2) - atan2(L3*m.sin(t3), L2 + L3*cos(t3)
    ↪ ))
#Inverse Orientation
T3 = mf.rotz(t1)@mf.rotx(t2)@my.rotx(t3)
T = T.I*R
t6 = m.atan2(T[1,0],-T[1,2])
t4 = m.atan2(T[0,1],T[2,1])
while t4 > pi/2:
    t4 -= pi
while t4 < -pi/2:
    t4 += pi

if t6 < 0:
    t6 += 2*pi

if (m.sin(t4) > -10e-6) and (sin(t4) < 10e-6):
    t5 = m.atan2(T[2,1]/m.cos(t4),T[1,1])
else:
    t5 = m.atan2(T[0,1]/m.sin(t4),T[1,1])

return np.matrix([[t1],[t2],[t3],[t4],[t5],[t6]])

```

Listing 17: meiosis_servo.py

```

#!/usr/bin/env python
# -*- coding: utf-8 -*-

"""basic readable servo commands"""

```

```

from dynamixel_sdk import *

# Control table addresses
ADDR_MX_TORQUE_ENABLE = 24
ADDR_MX_GOAL_POSITION = 30
ADDR_MX_PRESENT_POSITION = 36

# Protocol version
PROTOCOL_VERSION = 1.0

# Default setting
numdxl = 1;
BAUDRATE = 115200
DEVICENAME = '/dev/ttyUSB0'

gearidx = [(39.0)*(128.0/45.0), (39.0)*(128.0/45.0)]
offset = [0,13700]

# Initialize PortHandler instance
portHandler = PortHandler(DEVICENAME)
# Initialize PacketHandler instance
packetHandler = PacketHandler(PROTOCOL_VERSION)

class Servo:
    def __init__(self):
        self.data = []
        #self.initialize()

    def initialize(self):
        # Open port
        if portHandler.openPort():
            print("Succeeded to open the port")
        else:
            print("Failed to open the port")
            quit()
        # Set port baudrate
        if portHandler.setBaudRate(BAUDRATE):
            print("Succeeded to change the baudrate")
        else:
            print("Failed to change the baudrate")
            quit()

        # Enable Dynamixel Torque
        for i in range(0, numdxl):

```

```

dxl_comm_result, dxl_error = packetHandler.write1ByteTxRx(portHandler,
    ↪ i, ADDR_MX_TORQUE_ENABLE, 1)
if dxl_comm_result != COMM_SUCCESS:
    print("%s" % packetHandler.getTxRxResult(dxl_comm_result))
elif dxl_error != 0:
    print("%s" % packetHandler.getRxPacketError(dxl_error))
else:
    print("Dynamixel[ID]" + str(i) + "Torque has been enabled.")

def close(self):
    # Disable Dynamixel Torque
    for i in range(0,numdxl):
        dxl_comm_result, dxl_error = packetHandler.write1ByteTxRx(portHandler,
            ↪ i, ADDR_MX_TORQUE_ENABLE, 0)
        if dxl_comm_result != COMM_SUCCESS:
            print("%s" % packetHandler.getTxRxResult(dxl_comm_result))
        elif dxl_error != 0:
            print("%s" % packetHandler.getRxPacketError(dxl_error))
        else:
            print("Dynamixel[ID]" + str(i) + "Torque has been disabled.")
    # Close port
    portHandler.closePort()

def getPos(self, ID):
    return packetHandler.read2ByteTxRx(portHandler, ID,
        ↪ ADDR_MX_PRESENT_POSITION)[0]

def setPos(self, ID, pos):
    dxl_comm_result, dxl_error = packetHandler.write4ByteTxRx(portHandler, ID,
        ↪ ADDR_MX_GOAL_POSITION, pos)
    #print(dxl_comm_result, dxl_error)

def setVel(self, ID, vel):
    dxl_comm_result, dxl_error = packetHandler.write2ByteTxRx(portHandler, ID,
        ↪ 32, vel)

def moving(self, ID):
    return packetHandler.read1ByteTxRx(portHandler, ID, 46)[0]

def anyMoving(self, IDLIST):
    statusidx = [0,0]
    for i in range(0,len(IDLIST)):
        statusidx[i] = self.moving(IDLIST[i])
    return any(statusidx)

```

```

def setRes(self, ID, res):
    dxl_comm_result, dxl_error = packetHandler.write1ByteTxRx(portHandler, ID,
        ↪ 22, res)

def readRes(self, ID):
    return packetHandler.read1ByteTxRx(portHandler, ID, 22)[0]

def disableTorque(self, ID):
    dxl_comm_result, dxl_error = packetHandler.write1ByteTxRx(portHandler, ID,
        ↪ ADDR_MX_TORQUE_ENABLE, 0)

def enableTorque(self, ID):
    dxl_comm_result, dxl_error = packetHandler.write1ByteTxRx(portHandler, ID,
        ↪ ADDR_MX_TORQUE_ENABLE, 1)

def setMultiturn(self, ID, on):
    if(on):
        dxl_comm_result, dxl_error = packetHandler.write2ByteTxRx(portHandler,
            ↪ ID, 6, 4095)
        dxl_comm_result, dxl_error = packetHandler.write2ByteTxRx(portHandler,
            ↪ ID, 8, 4095)
    else:
        dxl_comm_result, dxl_error = packetHandler.write2ByteTxRx(portHandler,
            ↪ ID, 6, 0)
        dxl_comm_result, dxl_error = packetHandler.write2ByteTxRx(portHandler,
            ↪ ID, 8, 4095)

def setJA(self, IDLIST, angle):
    if(type(IDLIST) == int):
        IDLIST = [IDLIST]
    if(type(angle) == int):
        angle = [angle]
    for i in range(0, len(IDLIST)):
        self.setPos(IDLIST[i], int(round(angle[i] * gearidx[i] + offset[i])))
        print(int(round(angle[i] * gearidx[i] + offset[i])))

def setOffset(self, ID, amount):
    offset[ID] = amount
    dxl_comm_result, dxl_error = packetHandler.write2ByteTxRx(portHandler, ID,
        ↪ 20, amount)
    return offset[ID]

```

Listing 18: meiosis_utils.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import numpy as np
# import math as m
from math import *

class matrixOp:
    def rotx(theta):
        return np.matrix([[1,0,0],[0,cos(theta),-sin(theta)],[0,sin(theta),cos(theta)]])

    def roty(theta):
        return np.matrix([[cos(theta),0,sin(theta)],[0,1,0],[-sin(theta),0,cos(theta)]])

    def rotz(theta):
        return np.matrix([[cos(theta),-sin(theta),0],[sin(theta),cos(theta),0],[0,0,1]])

    def skew(r):
        return np.matrix([[0,-r[2],r[1]],[r[2],0,-r[0]],[r[1],r[0],0]])
```

Listing 19: twolink.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

"""messy two link manipulator testing"""

import meiosis_encoder as ME
import RPi.GPIO as GPIO
import meiosis_servo as MS
import time
from math import *

ser = MS.Servo()
ser.initialize()

#ser.setRes(0,4)
#ser.setRes(1,4)
#ser.setMultiturn(0, True)
#ser.setMultiturn(1, True)
```

```

IDs = [0,1]
xoff = 0
yoff = 0

def twoLinkIK(x,y):
    l1 = 265.0
    l2 = 165.0
    D = (x**2 + y**2 - l1**2 - l2**2)/(2*l1*l2)
    theta2 = atan2(sqrt(1.0 - D**2),D)
    theta1 = atan2(y,x) - atan2(l2*sin(theta2),l1+l2*cos(theta2))
    print(degrees(theta1),degrees(theta2))
    return [degrees(theta1),degrees(theta2)]

def zeroArm():
    print("Init:")
    print(ser newPos(0),ser newPos(1))
    print("servo_0_offset:")
    print(ser setOffset(0,ser newPos(0)))
    print("servo_1_offset:")
    print(ser setOffset(1,ser newPos(1)))
    print("New:")
    print(ser newPos(0),ser newPos(1))

def moveArm(gamma):
    #global IDs
    ser.setJA(IDs, gamma)
    while (ser.anyMoving(IDs)):
        pass
    return True

def pick(x,y):
    #global xoff, yoff
    moveArm(twoLinkIK(x-xoff,y-yoff))

def place(x,y):
    moveArm(twoLinkIK(x,y))

try:
    print("Entering Loop, press Ctrl-C to escape!")
    pos = ser newPos(0)*ser.readRes(0)
    print(ser newPos(0))
    try:
        dir = raw_input('Enter Direction (u/d), or (f) for Finished:')
        zeroing = True
        while zeroing:

```

```

while dir == 'u':
    userInput = raw_input()
    if userInput == 'f':
        zeroing = False
        break
    elif userInput == 'd':
        dir = 'd'
    else:
        pos += 50
        ser.setPos(0,pos)
        print(ser newPos(0))
while dir == 'd':
    userInput = raw_input()
    if userInput == 'f':
        zeroing = False
        break
    elif userInput == 'u':
        dir = 'u'
    else:
        pos -= 50
        ser.setPos(0,pos)
        print(ser newPos(0))
print(ser newPos(0))

except KeyboardInterrupt:
    print("interrupted")

"""

print(ser newPos(0))
ser.setPos(0,10000)
while ser.moving(0):
    pass
print(ser newPos(0))
#pick(250,50)
#place(-250,50)
"""

except KeyboardInterrupt:
    print("\nInterrupted!")

print("Disabling Servo Interface...")
ser.close()

```
