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

List of Figures

List of Tables

List Of Acronyms and Abbreviations

FK	:	Forward Kinematics
IK	:	Inverse Kinematics
PD	:	Proportional Derivative

Notation

$^{\text{Frame}}_{\text{From}}\, r\,_{\text{To}}$		:	Direction Vectors
$^{\text{From}}\, T\,_{\text{To}}$		:	Direction Cosine (Transformation) Matrices
	$c_{\theta_{nm}}$	:	$\cos(\theta_n + \theta_m)$
	$s_{\theta_{nm}}$	:	$\sin(\theta_n + \theta_m)$

Preliminary and Detailed Design

1 Mechanical System

Description of mechanical design (include dynamics equations, all structural analysis, etc.)

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:

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

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.

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

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

$$G(\gamma) = \left(\frac{\partial U({}^I r(\gamma))}{\partial \gamma} \right)^T \ , \quad U_B = \begin{bmatrix} 0 & 0 & g \end{bmatrix} \left({}^I_B r_B m_B + {}^I\, T_B\, {}^B_B\Gamma \right)$$

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

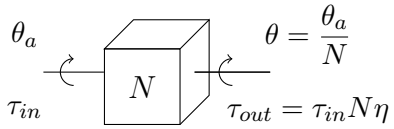
Given robot dynamics described by $H(\gamma)\ddot{\gamma} + n(\gamma, \dot{\gamma}) = \tau$, the torque, τ, provided by the servo motors is necessary to solve the closed loop dynamics of the system. Assuming the servo is driven by a D.C. motor with proportional derivative control,

$$\tau_a = K i_a = J_a \ddot{\theta}_a + b_a \dot{\theta}_a + \tau_L \tag{2}$$

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:

$$V_a = i_a R_a + K \dot{\theta}_a \tag{3}$$

Where V_a is the voltage applied to the actuator and R_a is the armature resistance. Given a gearbox with ${}^{\text{in}}/_{\text{out}}$ ratio N and efficiency η,



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

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

Substituing into equation (3) and solving for i_a:

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

$$V_a = \frac{R_a J_a N}{K} \ddot{\theta} + \frac{R_a b_a N}{K} \dot{\theta} + \frac{R_a}{KN\eta} \tau + KN\dot{\theta} \tag{4}$$

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:

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

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:

$$2\zeta\omega_n = \frac{b_a}{J_a} - \frac{K K_d}{R_a J_a N} + \frac{K^2}{R_a J_a} \tag{6}$$

$$\omega_n^2 = \frac{-K K_p}{R_a J_a N} \tag{7}$$

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

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

$$\begin{bmatrix} 1 & -(\alpha_1 J_1 + \beta_1 J_1) \\ 1 & -(\alpha_2 J_2 + \beta_2 J_2) \\ \vdots & \vdots \end{bmatrix} \begin{bmatrix} \dfrac{R_a b_a N^2 \eta - K K_d N \eta + K^2 N^2 \eta - K K_p N \eta}{R_a J_a N^2 \eta} \\ \\ \dfrac{1}{J_a N^2 \eta} \end{bmatrix} = \begin{bmatrix} \alpha_1 + \beta_1 \\ \alpha_2 + \beta_2 \\ \vdots \end{bmatrix} \tag{8}$$

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

$$\frac{R_a b_a N - K K_d + K^2 N \eta - K K_p}{R_a J_a N} \tag{9}$$

$$\frac{1}{J_a N^2 \eta} \tag{10}$$

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

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

The MATLAB code implementing this process can be found in the Appendix (see section 2.1, p. 18, *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,

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

Where

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

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 1.*

Figure 1 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).

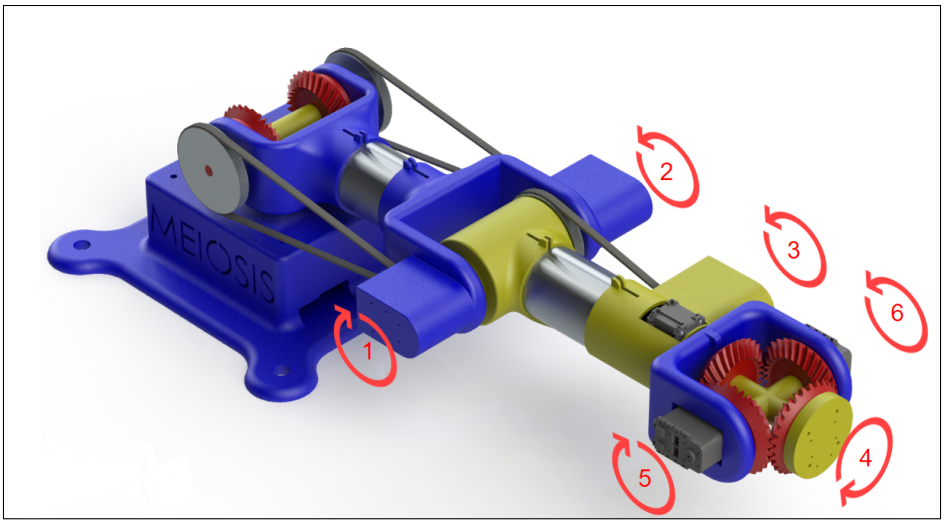


Figure 1: Motor Locations and Orientations

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

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.

$$\theta = A^{-1}\gamma \tag{13}$$

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.

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

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

$$H(\gamma)\ddot{\gamma} + d(\gamma, \dot{\gamma}) + G(\gamma) = -A^{-T}\left(C_1 A^{-1}\ddot{\gamma} + C_2 A^{-1}\dot{\gamma} + C_3\theta_d - C_3 A^{-1}\gamma\right)$$

$$\ddot{\gamma} = H(\gamma)^{-1}\left(-A^{-T}\left(C_1A^{-1}\ddot{\gamma} + C_2A^{-1}\dot{\gamma} + C_3\theta_d - C_3A^{-1}\gamma\right) - d(\gamma,\dot{\gamma}) - G(\gamma)\right) \tag{14}$$

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.

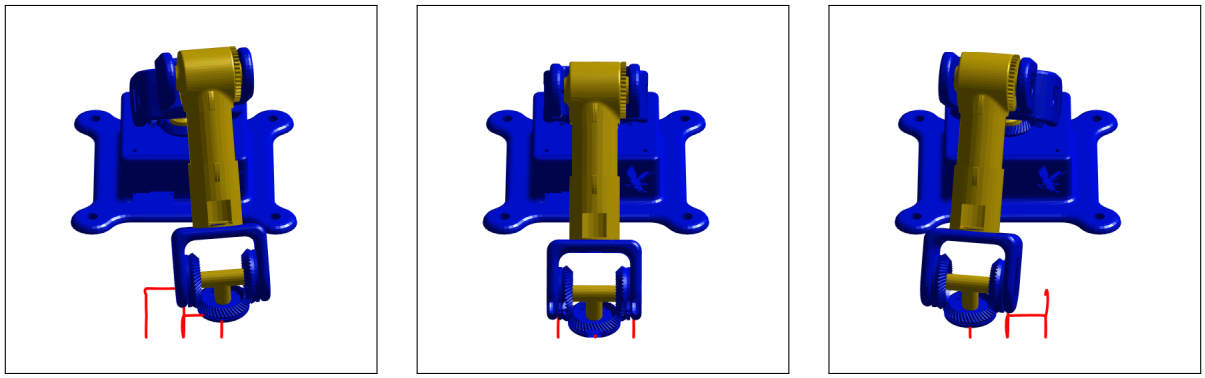
$$A^{-T}C_3u = G(\gamma) + d(\gamma,\dot{\gamma}) + A^{-T}C_3\theta_d$$

$$u = \left(A^{-T}C_3\right)^{-1}\left(G(\gamma) + d(\gamma,\dot{\gamma})\right) + \theta_d \tag{15}$$

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

$$\ddot{\gamma} = H(\gamma)^{-1}\left(-A^{-T}\left(C_1A^{-1}\ddot{\gamma} + C_2A^{-1}\dot{\gamma} + C_3u - C_3A^{-1}\gamma\right) - d(\gamma,\dot{\gamma}) - G(\gamma)\right) \tag{16}$$

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



(a) Frame Snapshot near Simulation Initiation

(b) Frame Snapshot near Simulation Middle

(c) Frame Snapshot near Simulation Termination

Figure 2: Closed-Loop Control Simulation Animation Snapshots

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 4* 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.

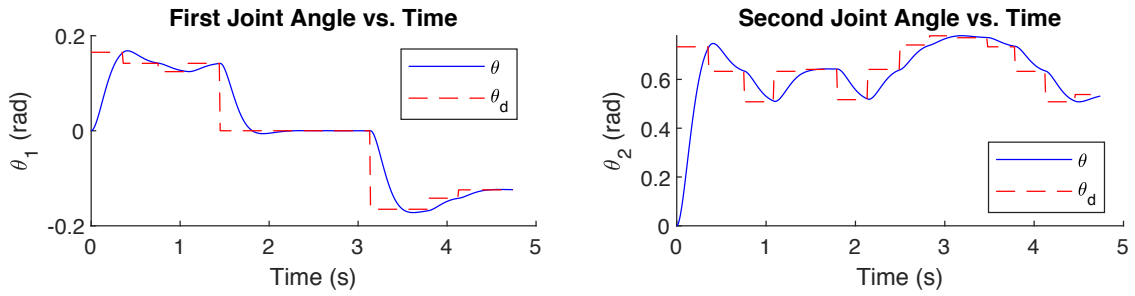


Figure 3: Joint Angles vs Time in Closed-Loop Simulation

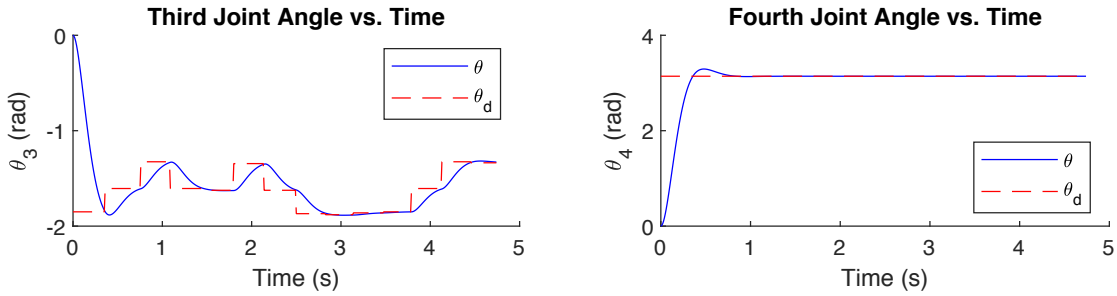


Figure 3 (cont.): Joint Angles vs Time in Closed-Loop Simulation

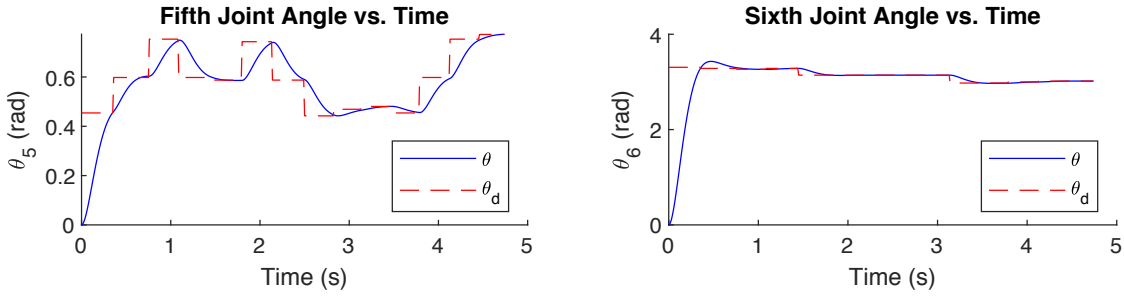


Figure 3 (cont.): Joint Angles vs Time in Closed-Loop Simulation

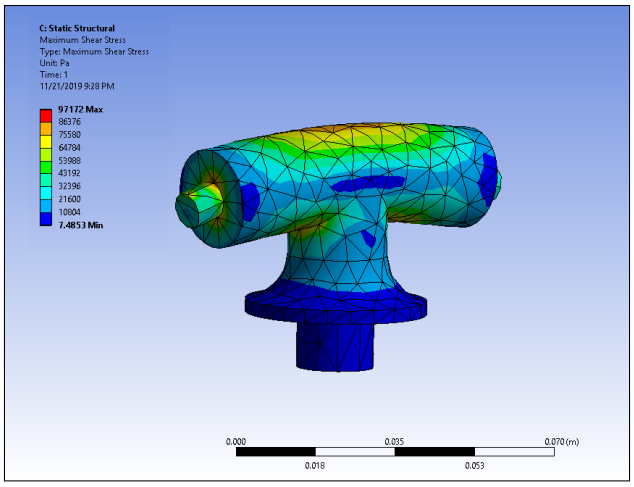


Figure 4: T-Bar ANSYS FEA

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 5*.

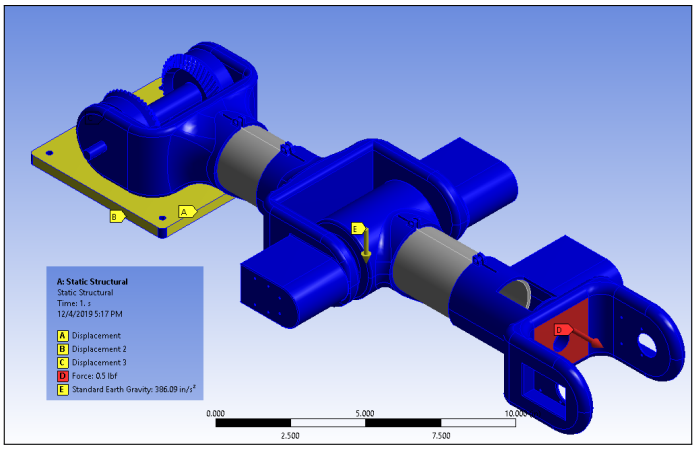
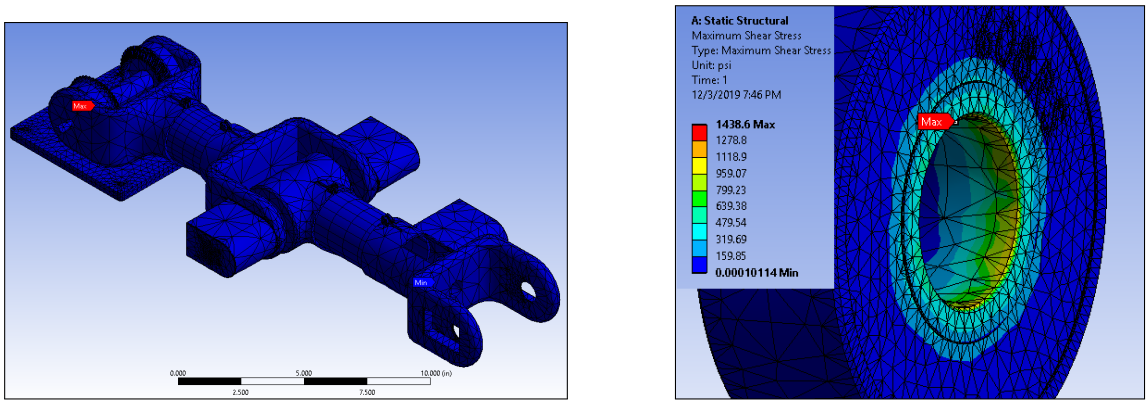


Figure 5: ANSYS Simulated Forces Image Capture

As shown in *Figure 5*, 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 6a*; a close-up image of the bearing analysis can be seen in *Figure 6b*.



(a) ANSYS Full View of Maximum Shear

(b) ANSYS Bearing Shear Stress Close-up

Figure 6: 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 7*, showing it is unlikely to fail due to material yeilding.

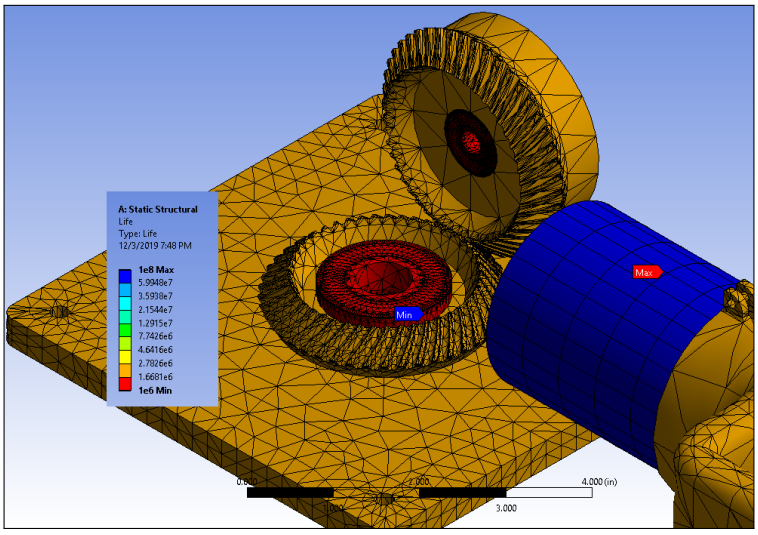


Figure 7: ANSYS Fatigue Test

As seen in *Figure 7*, the lower bearing of the differential drive on the shoulder of the manipulator would be the most likely component to fail under repeating loadings.

1.1 Parts list and budget

1.2 Parts List

Table 1 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 1* 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 1* 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 2* for physical hardware. To allow screw mounting in plastic with metal threads, *Table 2* 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 1*, *Table 2* 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 2* 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 2* assumes the end-user does not already possess an AX-12A servo to be used with the end-effector. Further, *Table 2* 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 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 poses 3D printers.

Table 1: MEIOSIS Bill of Materials with Costs

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

Table 2: End-User Bill of Materials with Costs

Part	Retailer	Quantity	Unit Cost (USD)	Total Cost (USD)
Aforementioned Costs	*Table 1*	—	—	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

2 Software (path/ trajectory planning, localization, object detection, and control algorithms)

2.1 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 8*.

Figure 8 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 8a*.

Figure 8a 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

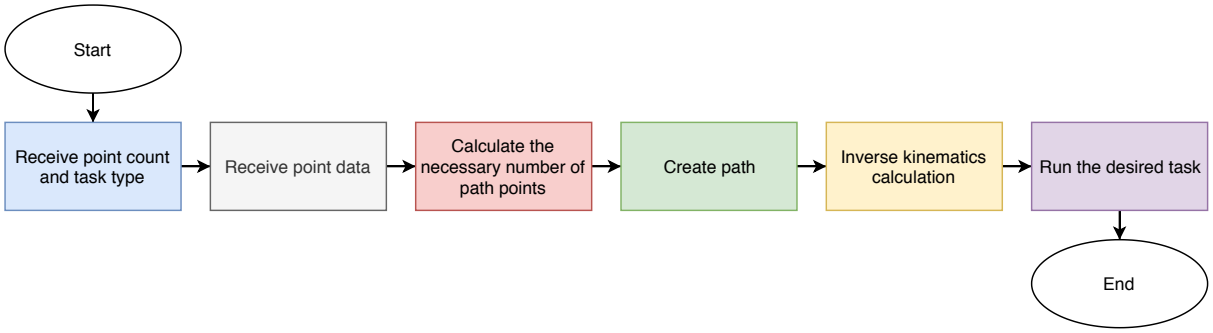


Figure 8: Software Flowchart

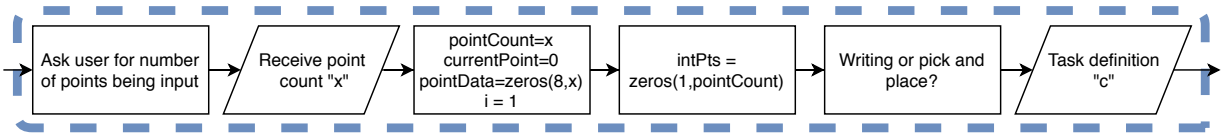


Figure 8a: Software Flowchart Subsection 1

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 8b*.

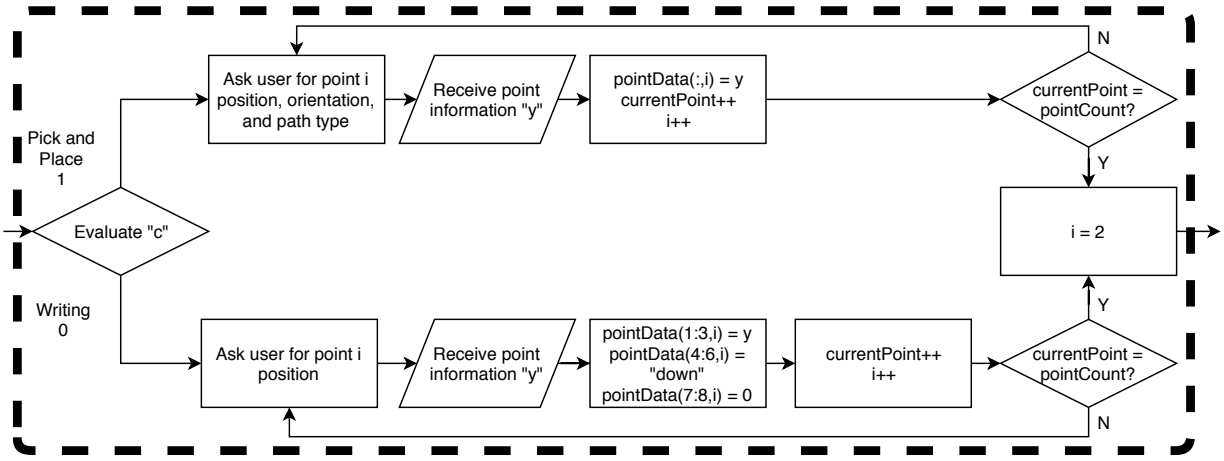


Figure 8b: Software Flowchart Subsection 2

Figure 8b 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 8c*.

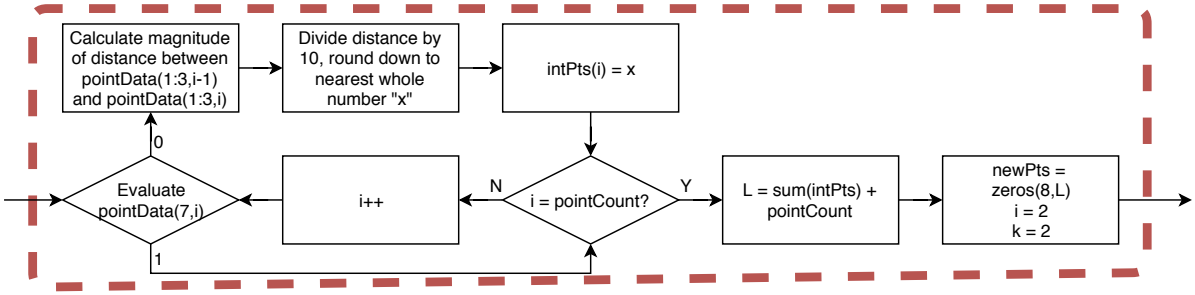


Figure 8c: Software Flowchart Subsection 3

As seen in *Figure 8c*, 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 until 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 8d*.

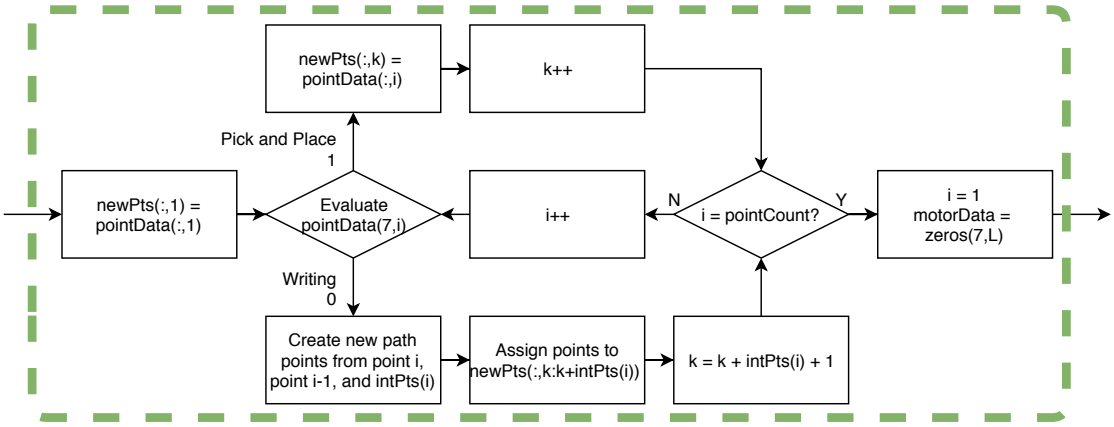


Figure 8d: Software Flowchart Subsection 4

The code shown in *Figure 8d* 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 8e*.

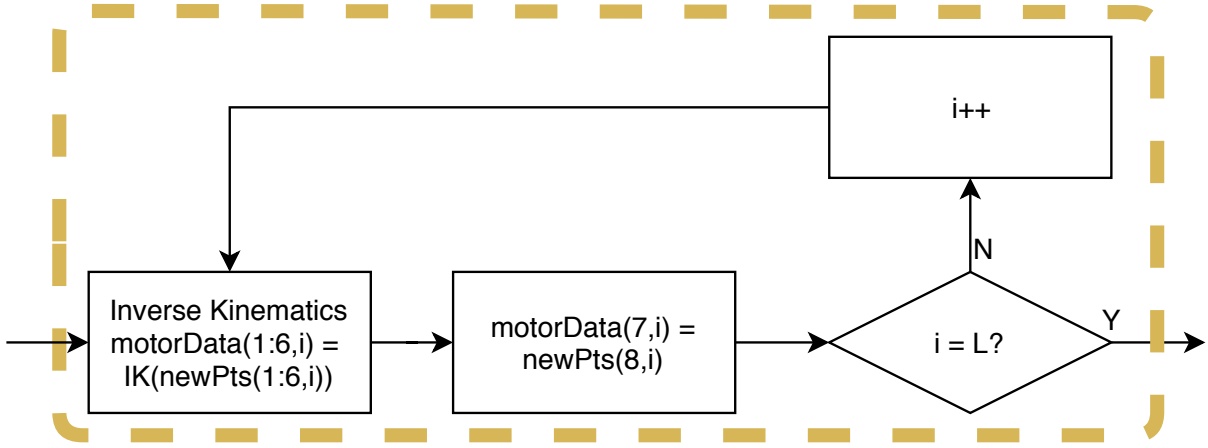


Figure 8e: Software Flowchart Subsection 5

Figure 8e 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 8f*.

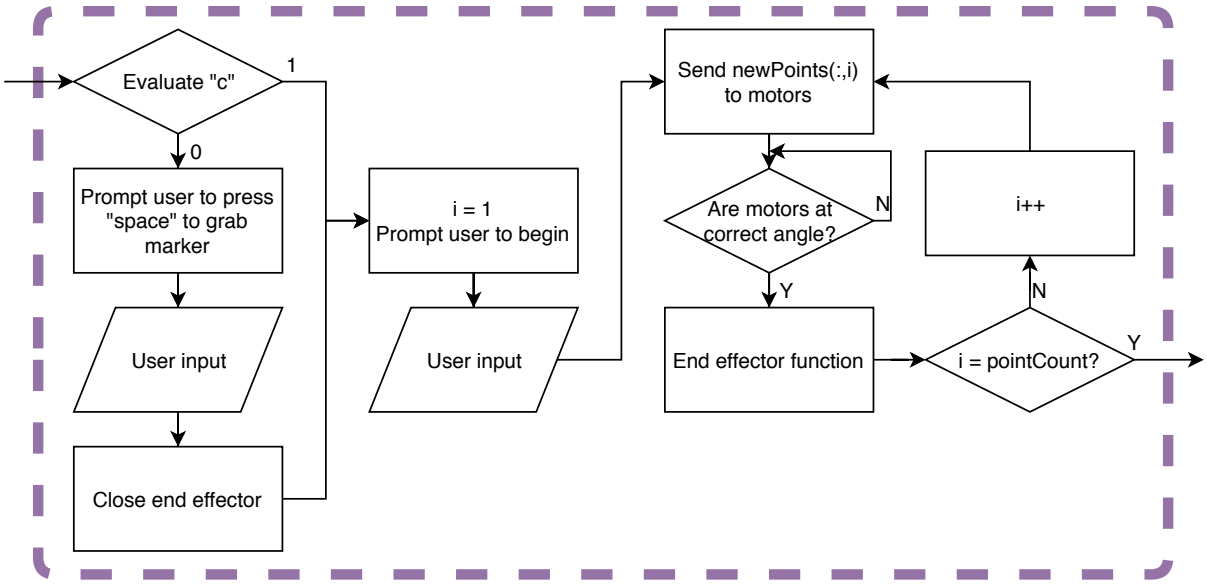


Figure 8f: Software Flowchart Subsection 6

The code in *Figure 8f* 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.

References

[1] Trossen robotics. URL *https://www.trossenrobotics.com/*.

[2] Embry-riddle aeronautical university-prescott. URL *https://www.collegetuitioncompare.com/edu/104586/embry-riddle-aeronautical-university-prescott/*.

[3] Range of robot cost – robot system cost series. URL *https://motioncontrolsrobotics.com/range-robot-cost/*.

[4] Sawyer: Rethink robotics unveils new robot. URL *https://spectrum.ieee.org/automaton/robotics/industrial-robots/sawyer-rethink-robotics-new-robot*.

[5] Pla density. URL *https://all3dp.com/2/pla-density-what-s-the-density-of-pla-filament-plastic/*.

[6] Michael Zeltkevic 1. Runge-kutta methods, 04 1998. URL *http://web.mit.edu/10.001/Web/Course_Notes/Differential_Equations_Notes/node5.html*.

[7] S. Hutchinson M. Spong and M. Vidyasager. *Robot Modeling and Control*. 2006.

[8] 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

```
% dynamixel motor model experiment
close all;clear;clc

% test loads mass moments of inertia
m = [.146 .088 0.108];          % mass (kg)
b = [.61277 .37227 0.28257];    % length (m)
h = [.01915 0.01915 0.0299];    % height (m)
J = (1/12)*m.*(h.^2 + b.^2);

% path to csv files relative to script
datapath = 'data/AX12A/';
files = dir(strcat(datapath,'*.csv'));
numFiles = length(files);
% initialize variables
[damp, wn, Tp] = deal(zeros(numFiles,1));

for ii = 1:numFiles
    % load experimental data, skip 5 header lines
        M = csvread(strcat(datapath, files(ii).name),5,0);
    % clean data by removing outliers
        nani = (find(diff(M(:,1)) > 100));
        M(nani,:) = [];
    % show response
         figure();
         plot(M(:,1),M(:,2))
         title('Experimental Data')
    % find % OS
        peak = max(M(:,2));                         % peak value
        peaki = find(M(:,2)==peak, 1, 'first');     % peak value index
        ss = M(end,2);                              % steady state
        os = ((peak - ss) / ss) * 100;              % % OS
    % damping ratio
        damp(ii) = -log(os/100) / sqrt(pi^2 + log(os/100)^2);
    % find where the motor begins responding
        start = M(find(diff(M(:,2)) > 1, 1, 'first'), 1);
    % time to peak
        Tp(ii) = (M(peaki,1) - start) / 1000;
    % natural frequency
        wn(ii) = pi / (sqrt(1 - damp(ii)^2)*Tp(ii));
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;
```