


October 2020

## Control of a Human Arm Robotic Unit Using Augmented Reality and Optimized Kinematics

Carlo Canezo  
*University of South Florida*

Follow this and additional works at: <https://scholarcommons.usf.edu/etd>

 Part of the [Computer Sciences Commons](#), [Mechanical Engineering Commons](#), and the [Robotics Commons](#)

---

### Scholar Commons Citation

Canezo, Carlo, "Control of a Human Arm Robotic Unit Using Augmented Reality and Optimized Kinematics" (2020). *Graduate Theses and Dissertations*.  
<https://scholarcommons.usf.edu/etd/8519>

This Thesis is brought to you for free and open access by the Graduate School at Scholar Commons. It has been accepted for inclusion in Graduate Theses and Dissertations by an authorized administrator of Scholar Commons. For more information, please contact [scholarcommons@usf.edu](mailto:scholarcommons@usf.edu).

Control of a Human Arm Robotic Unit Using Augmented Reality and Optimized Kinematics

by

Carlo Canezo

A thesis submitted in partial fulfillment  
of the requirements for the degree of  
Master of Science in Mechanical Engineering  
Department of Mechanical Engineering  
College of Engineering  
University of South Florida

Co-Major Professor: Redwan Alqasemi, Ph.D.  
Co-Major Professor: Rajiv Dubey, Ph.D.  
Sudeep Sarkar, Ph.D.

Date of Approval:  
October 29, 2020

Keywords: Prosthesis, Resolved Rate, Jacobian, Eye Fixation, Singularity Robust

Copyright © 2020, Carlo Canezo

## Dedication

I dedicate this to my wife Mizuratha Canezo, who steadfastly supported me in all my endeavors and dreams, no matter how lofty or nonsensical it may be especially as it deals with robotics. I also dedicate this to my pet Haru, who patiently listened to my frustrations and ramblings, steadfastly kept me company in the late nights, and given me her quiet support throughout my research. I name a part of the system after her.

## Acknowledgments

I would like to acknowledge my team members who helped make this project take form: Hunter Bassett, Paola Rossi, Dika Ezevillo, Tommy Truong, Eric Thivierge, and Mark Thivierge. Each person dedicated their time, skills, and knowledge to help move this project forward and I cannot thank them enough for their contribution. I would also like to acknowledge Professor Alqasemi and the CARRT Lab, for providing guidance and insight into making this robotic system.

## Table of Contents

List of Tables .....	iv
List of Figures .....	v
Abstract.....	viii
Chapter 1: Introductions.....	1
1.1 History .....	1
1.2 Motivation.....	1
Chapter 2: Background Literature Review.....	4
2.1 Existing Powered Prostheses .....	4
2.2 Control Strategies .....	5
2.3 Human Machine Interfaces [HMI] .....	7
Chapter 3: Research Goal and Objectives.....	8
3.1 Research Goal .....	8
3.2 Research Objectives.....	9
Chapter 4: Hanson Arm Solid Model and Kinematics.....	11
4.1 Hanson Arm Solid Model .....	11
4.1.1 Frame Assignments.....	12
4.1.2 DH Parameters.....	13
4.2 Kinematics and Redundancy Resolution.....	14
4.2.1 Forward Kinematics and Transformation Matrices .....	14
4.2.2 Optimized Jacobian.....	15
4.2.3 Weight Matrix - Gradient Projection Method .....	16
4.2.4 Singularity Robust Inverse (SRI).....	18
4.2.5 Trajectory Generation and Matching .....	19
4.2.6 Position Errors.....	20
4.2.7 Rotation Errors.....	21
4.2.8 Resolved Rate Method and Updated Forward Kinematics .....	24
Chapter 5: Programming.....	26
5.1 Programming Overview .....	26
5.2 Control System Flowchart.....	26
5.3 Magic Leap – Lumin SDK .....	27

5.3.1 User Interface Development.....	28
5.3.2 Sensor Implementation for User Interface (UI).....	32
5.3.2.1 Eye Tracking Implementation .....	33
5.3.2.2 Head Pose Implementation .....	33
5.3.2.3 Data Filtering.....	34
5.3.2.4 Telemetry Logging.....	35
5.4 Controller C++ Code .....	36
5.4.1 Kinematics.cpp.....	37
5.4.2 Motor.cpp and Dynamixel SDK .....	38
5.4.3 Controller.cpp .....	39
Chapter 6: Hardware Development and System Integration .....	41
6.1 Magic Leap .....	41
6.1.1 Eye Tracking .....	43
6.1.2 Head Tracking .....	43
6.1.3 Depth Tracking.....	44
6.2 Robotic Arm Hardware Development .....	45
6.2.1 Housings.....	45
6.2.2 Hardware Build Progression .....	48
6.2.3 Lessons Learned.....	59
6.3 Cost .....	60
6.4 Integration of Hardware and Software.....	61
6.4.1 Low Power Testing (LPT).....	61
6.4.2 High Powered Testing (HPT) .....	62
6.4.3 Proof of Concept Realization .....	63
Chapter 7: Testing and Results .....	64
7.1 Methodology.....	64
7.2 Simulation .....	64
7.2.1 Comparison Between VRML and MATLAB Simulation .....	64
7.2.2 Simulation Results of the Control System .....	65
7.2.3 Stable Trajectory Comparison.....	66
7.2.4 Unstable Trajectory Comparison .....	70
7.3 Experimental Testing .....	73
7.3.1 Experimental Graphed Results .....	74
7.3.2 Discussion.....	75
Chapter 8: Conclusions and Future Work.....	78
8.1 Conclusions .....	78
8.2 Future Work – AI Development.....	78
8.2.1 Object Recognition.....	79
8.2.2 Pose Estimation.....	80

8.2.3 Intention Recognition .....	81
References .....	82
Appendix A: Copyright Permissions .....	86

## List of Tables

Table 2.1: Various Manufacturers of Consumer Powered Prosthesis.....	4
Table 4.1: DH Paramaters Hanson Right Arm – Length in CM, Angles in Degrees.....	13
Table 6.1: Linkage Type per Joint.....	51
Table 6.2: High Level Cost of Project .....	60



## List of Figures

Figure 1.1: DEKA Arm.....	1
Figure 3.1: High Level System Flowchart .....	9
Figure 4.1: Hanson Arm 3D Model.....	11
Figure 4.2: Frame Assignments for the Right Arm.....	12
Figure 5.1: HARU Control System Flowchart .....	27
Figure 5.2: Lumin Runtime Editor .....	28
Figure 5.3: User Interface View from Headset .....	29
Figure 5.4: Prism Visualization.....	30
Figure 5.5: Pseudocode for State Structuring for Graphical User Interface.....	32
Figure 5.6: Pseudocode of Finite Impulse Response Filter.....	35
Figure 5.7: Text Output from Magic Leap.....	35
Figure 5.8: Code Flowchart .....	36
Figure 5.9: Controller.cpp Control Flow .....	40
Figure 6.1: Magic Leap Headset.....	41
Figure 6.2: Magic Leap Light Pack.....	42
Figure 6.3: Infrared Sensor Location.....	43
Figure 6.4: Electronic Board Location .....	44
Figure 6.5: IR Projector Location.....	44
Figure 6.6: Hanson Arm See Through Top View .....	45
Figure 6.7: FDM 3D Print Material Comparison: (1) Nylon, (2) PETG, (3) PLA.....	47

Figure 6.8: Bracket Construction Comparison between PETG (Left, Black Color) versus ABS (Right, White Color) .....	47
Figure 6.9: Finger Construction Comparison between PETG (Left, Black Color) versus SLA Resin (Right, Grey Color) .....	48
Figure 6.10: ABS Housing with Support Structures from 3D printing Still Fused to the Housing .....	49
Figure 6.11: Base Structure for Robotic Arm .....	50
Figure 6.12: Fit Up of Housings for Arm .....	51
Figure 6.13: Region 1 Major Components (1: Housing, 2: Pulley Block, 3: Motor) .....	52
Figure 6.14: Direct Drive Interface for Motor 1 .....	53
Figure 6.15: Region 2 Major Components (1: Bracket, 2: Pulley Block, 3: Motor, 4: Driven Gear) .....	53
Figure 6.16: Region 3 Major Components (1: Drive Motor, 2: Drive Motor) .....	54
Figure 6.17: Region 4 Major Components (1: Pulley Block, 2: Bracket, 3: Motor, 4: Driven Gear) .....	55
Figure 6.18: Region 5 Major Components (1: Pulley Block for Joint 7, 2: Motor for Joint 6, 3A/3B: Motor for Finger Actuation, 4: Pulley Block for Joint7, 5: Pulley Block for Joint 6) .....	56
Figure 6.19: Region 6 Hand Model .....	57
Figure 6.20: Assembled Arm .....	58
Figure 6.21: Typical Wiring Setup for Motors .....	58
Figure 6.22: Dynamixel Wizard Motor Test .....	61
Figure 6.23: High Power Test In Progress .....	62
Figure 6.24: Proof of Concept Visualization .....	63
Figure 7.1: Comparison of VRML (Left) and Wireframe (Right) Example 1 .....	65
Figure 7.2: Comparison of VRML (Left) and Wireframe (Right) Example 2 .....	65
Figure 7.3: Joint Angles, Joint Velocities, and Manipulability for Stable Trajectory for Arm Configuration A .....	69

Figure 7.4: Joint Angles, Joint Velocities, and Manipulability for Stable Trajectory for Arm Configuration B .....	69
Figure 7.5: Joint Angles, Joint Velocities, and Manipulability for Stable Trajectory for Arm Configuration C .....	70
Figure 7.6: Stable Manipulability Graphs Scaled for Comparison Configuration A (Left), Configuration B (Middle), and Configuration C (Right) .....	70
Figure 7.7: Joint Angles, Joint Velocities, and Manipulability for Stable Trajectory for Arm Configuration A.....	71
Figure 7.8: Joint Angles, Joint Velocities, and Manipulability for Stable Trajectory for Arm Configuration B .....	72
Figure 7.9: Joint Angles, Joint Velocities, and Manipulability for Stable Trajectory for Arm Configuration C .....	72
Figure 7.10: Unstable Manipulability Graphs Scaled for Comparison Configuration A (Left), Configuration B (Middle), and Configuration C (Right) .....	73
Figure 7.11: Simulation Results for Joint Position and Manipulability .....	74
Figure 7.12: Motor Feedback Results for Joint Position & Manipulability for Run 1 .....	74
Figure 7.13: Motor Feedback Results for Joint Position & Manipulability for Run 2 .....	75
Figure 7.14: Motor Feedback Results for Joint Position & Manipulability for Run 3 .....	75
Figure 8.1: High Level Visual System Code FlowChart .....	79
Figure 8.2: State/Action Network .....	81

## Abstract

There are more than 350000 amputees in the US who suffer loss of functionality in their daily living activities, and roughly 100000 of them are upper arm amputees. Many of these amputees use prostheses to compensate part of their lost arm function, including power prostheses. Research on 6-7 degree of freedom powered prostheses is still relatively new, and most commercially available powered prostheses are typically limited to 1 to 3 degrees of freedom. Due to the myriad of possible options for various powered prostheses from different manufacturers, each configuration is governed by a distinct control scheme typically specific to the manufacturer. The user will then have to be accustomed to its custom control scheme to be able to use such prostheses for ADL tasks. Control of available powered prosthesis options utilize different strategies such as individual joint control, partial endpoint control, or switching between different modes that is mentally demanding for the user leading to possible abandonment of such devices in favor of more passive systems. To overcome such issues, a novel resolved rate algorithm using Cartesian control was developed for universal use by having the user specify where the end effector must go through visual targeting. This is achieved by utilizing an augmented reality device "Magic Leap" to provide the spatial targeting information to the controller, which will then autonomously move the end effector to the targeted location. This controller system is simulated on a virtual humanoid arm model and tested on a Human Arm Robotic Unit, which is the hardware version of the arm model.

## Chapter 1: Introductions

### 1.1 History

The DEKA arm is a 7 DOF robotic prosthesis arm system that mimicked a natural arm developed by DARPA for veteran amputee rehabilitation [Resnik et al, 2018] as shown in (Figure 1.1). Historically, the system's robotic arm motion was controlled by employing several positioning inputs for each respective joint, and preprogrammed commands for various hand grips. The major drawback of its control scheme was that it was mentally and physically fatiguing just to move the arm to a desired location especially through prolonged use [Phillips et al, 2013].



Figure 1.1: DEKA Arm

Reprinted from "Endpoint Control for a Powered Shoulder Prosthesis," by S.L. Phillips, L. Resnik, C. Fantini, and G. Latlief, *Journal of Prosthetics and Orthotics*, Vol. 25, No. 4, pp. 193-200, 2013. Reprinted with permission.

### 1.2 Motivation

In analyzing the DEKA System, we found out that it employs 12 different positioning inputs along with one switching input just to place the hand where the user wants it in space with the

wrist oriented properly which would use an inertial measurement unit (IMU) mounted on a free foot as the “joystick” controller [Resnik et al, 2013]. Afterwards, another input is required to switch the system to the Grip Control Scheme and the user must select a pre-programmed grip by cycling through the choices using another switch. This method, while effective, does have some drawbacks: (i) the user must fine tune the final hand and wrist location via the 12 position inputs, and (ii) the user must frequently use a switch input back and forth to control various segments of the arm, which can cause frustration and fatigue on the user. We believe that these input requirements are a major contributor to the complexity of use and led us to speculate that the current control effort would eventually compound into mental and physical fatigue for the user [Philips et al, 2013].

The proposed project’s goal is to minimize the complexity of controlling the arm system by optimizing the control scheme of the unit and create a human-like arm motion. To achieve this goal, we integrated sensory information to feed directly to the arm controller in order to create arm trajectories that can help the user readily complete typical ADL tasks as well as reduce the mental and physical load of using the prosthesis. The controller must also be agnostic that it can be applied to other types of powered prosthesis without significant modification.

One aspect of this project is the use of visual servoing principles [Perez et al, 2016] to control the arm end effector (hand) position relative to the user. Using a wearable headset that can track distance and position based on eye gaze, the user can target an object within their vicinity just by looking at it and activating a trigger. The subsequent sensory information will then provide the required spatial cartesian coordinates to the robotic arm controller to create a

trajectory from its current position to the target object. The controller will then move the end effector (in this case, hands, and wrist) towards the desired location.

## Chapter 2: Background Literature Review

### 2.1 Existing Powered Prostheses

Initial research on different types of available powered prosthesis, as shown in (Table 2.1), indicated that while powered prostheses are commonly used as an assistive technology, due to the myriad of possible options for various powered prostheses from different manufacturers each configuration is governed by a distinct control scheme typically specific to that manufacturer. The user will then have to be accustomed to its custom control scheme, typically some form of myoelectric control, to be able to use the prosthesis for ADL tasks. Important also to note is that these devices are typically limited to 1-3 degrees of freedom (DOF). Prostheses that utilize 6-7 DOF are still under research and development, and only few had made it to commercialization.

Table 2.1: Various Manufacturers of Consumer Powered Prosthesis

Description	Manufacturer	Key Items
DynamicArm[Ottobock, 2019]	Ottobock	<ol style="list-style-type: none"><li>1. 50 N Lifting Force</li><li>2. Flexion Angle 15 to 145 degrees</li><li>3. Myoelectric Type</li><li>4. Software Controlled</li><li>5. 6 kg Max Lift Weight</li></ol>
MyoRotronic[Ottobock, 2019]	Ottobock	<ol style="list-style-type: none"><li>1. Allows for Electromotive Pronation and Supination, and open and close hand</li><li>2. Combine with MyoWrist (Passive) for Flexion and extension</li></ol>
Utah Arm U3/U3+[Fillauer, 2019]	Fillauer	<ol style="list-style-type: none"><li>1. 22 kg Max load limit</li><li>2. 135 deg excursion range (20 - 155)</li><li>3. 1.2 sec excursion time</li><li>4. Humeral Rotation with Quick Disconnect Wrist - 360 deg</li><li>5. Software Controlled</li></ol>



Table 2.1: Continued

Description	Manufacturer	Key Items
Boston Arm[Liberating Technologies Inc, 2012]	Liberating Technologies (LTI)	<ol style="list-style-type: none"> <li>1. Lift 10ftlbs</li> <li>2. 8 inputs, 4 outputs</li> <li>3. Flexion Angle 0 - 135</li> <li>4. Terminal Device Board for multi options</li> </ol>
In Hand Wrist Rotator[Fillauer, 2019]	Fillauer	<ol style="list-style-type: none"> <li>1. Motor Drive</li> <li>2. High torque 15inlbs</li> <li>3. 32 rpm</li> <li>4. Microprocessor control</li> </ol>
BeBionic[Ottobock, 2020]	Ottobock	<ol style="list-style-type: none"> <li>1. 14 grip patterns</li> <li>2. 2 thumb positions</li> <li>3. Battery installed in arm</li> <li>4. Software Control</li> </ol>
Michelangelo[Ottobock, 2020]	Ottobock	<ol style="list-style-type: none"> <li>1. Control by AxonSoft software</li> <li>2. 7 patterns by default</li> </ol>
Taska[Fillauer, 2018]	Fillauer	<ol style="list-style-type: none"> <li>1. Water Resistant</li> <li>2. 23 grips</li> <li>3. Software Support</li> <li>4. Passive Wrist</li> </ol>
iLimb[Ossur, 2020]	Ossur/Touch Bionic	<ol style="list-style-type: none"> <li>1. Up to 24 grips available, can program extra 12</li> <li>2. App controlled</li> </ol>

## 2.2 Control Strategies

Upper limb prosthesis development has seen some significant development recently with its integration of robotics. Good comparative examples born from DARPA are the DEKA and John Hopkins' Applied Physics Lab's (APL) Modular Prosthetic Limb (MPL) with the intention of integrating a neural interface for controlling the arm, along with a slew of other capabilities [Johannes et al, 2011]. However, such development comes with challenges. [Resnik et al, 2018] noted that as the demand of such devices increases, the effectiveness of using such a device will now be a factor due to its high cost. Many users will reject such devices if it proves to be too

complicated to use [Resnik et al, 2017]. Evaluations done by [Resnik et al, 2017] found that while a robotic prosthesis like DEKA does allow for some added capabilities, it mostly supplements rather than replaces other standard prosthesis. This indicates that such robotic systems still have a long way to go as a front runner solution for amputees.

Controlling a powered prosthesis presents several challenges and complexities depending on the degree of the amputation. As an example, a powered prosthesis replacement for a shoulder amputee means the person must impart more effort to control all the degrees of freedom to accomplish a typical activity of daily living (ADL) [Resnick et al, 2018]. With most standard powered prosthesis available, the widely adopted control scheme is Direct Joint Control. This can entail several different configurations such as: 1) sequential control, 2) simultaneous control, 3) Linked movements, and/or 4) combination of 1 & 2. Ultimately, these control schemes have drawback that normally translate as difficulty in use for the user [Phillips et al, 2013]. To check if the control system can be optimized, evaluation was done using a partial endpoint control scheme [Phillips et al, 2013]. Partial endpoint control entails simultaneous actuation of some of the power joints to bring the endpoint to a desired partial (limited) spatial location through inverse kinematics. This method reduces the amount of user input for motion compared to the joint control, and it is possible to simulate a more natural arm movement.

Computer vision systems can also be effectively used to directly or supplement control of robotic manipulators with the main goal of reducing the amount of cognitive load it takes to control the system. [Perez et al, 2016] presented and compared a comprehensive list of various techniques and sensors already being used with robotic manipulators such as stereo vision, time of flight, and structured light strategies just to name a few. [Kofman et al, 2005] presented his

work on using a vision-based system to provide feedback to the robotic manipulator for accuracy. [Fujii et al, 2013] utilized a gaze system for cartesian control of a robotic arm for fine surgical movement. [Leeper et al, 2010] evaluated the feasibility of using stereo vision for robotic grasping in a cluttered workspace, while [Ramisa et al, 2012] showcased a method of robotic grasping specifically for clothes using depth and appearance feature detection. Visual sensor assisted systems also help give greater control of the unit to the user increasing its performance efficiency by reducing the amount of user required input as well as robotic arm execution times as shown by [Yu et al, 2003]. More specifically to powered prosthesis, [Ghazaei et al, 2017] was also able to showcase combining computer vision with deep learning to increase grip functionality in hand prosthesis by classifying target objects for grasping, and having the AI determine the required grasp type.

### **2.3 Human Machine Interfaces [HMI]**

Since the secondary portion of our research involves integrating a visual platform to track targets or point locations, we looked at various research already being implemented. Regarding the initial target acquisition by stereo vision, several researches into this field have investigated its feasibility. Using stereo vision systems, [Kang et al, 2008] were able to capture distance and velocity measurements from a target with good accuracy. [Wang et al, 2009] used binocular stereo vision for target detection by filtering out noise and background data and was successful in detecting a moving object. [Postelnicu et al, 2011] even utilized electrooculography (EOG) and electroencephalography (EEG) to control a robotic arm.

## Chapter 3: Research Goal and Objectives

### 3.1 Research Goal

The goal of this research is to present a novel approach to use an existing prosthetic arm technology to assist amputees by developing a new universal control system for upper arm powered prostheses to reduce the amount of effort needed to operate the arm, utilizing a robust controller program with visual servoing principles, and to implement this with an actual robotic arm unit. In the future, the plan also is to add artificial intelligence (AI) to the vision system and motion controller in relation to object recognition and intention recognition in order to increase the efficiency of the system so users will acclimate easier to using powered prostheses, regardless of the type of manufacturer or design and control configurations.

The system we are proposing involves using visual servoing principles to control the most common powered prosthetic devices. This will utilize a wearable head mounted visual sensor, in the form of a state-of-the-art augmented reality (AR) goggles, to determine the intended location of the end effector/hand and the intended target objects the user wishes to interact with. Eye gaze and sensory information from the AR device can determine the required coordinates of objects and provide the information to the prosthetic arm controller once an object is selected through a specific trigger and lock on to a target point. The controller can then be programmed to determine the requisite kinematics equations and plan a path of the end effector towards the desired location. Once the path is set, the user uses a single input to control the speed along the path to the intended target.

Of the existing technologies that can be implemented to better control the prostheses, visual servoing with sensory feedback information promises to be a viable solution. Our proposed new control system for powered prostheses, as shown in (Figure 3.1), should be easier to implement and use than existing methods without any significant change to the existing hardware.

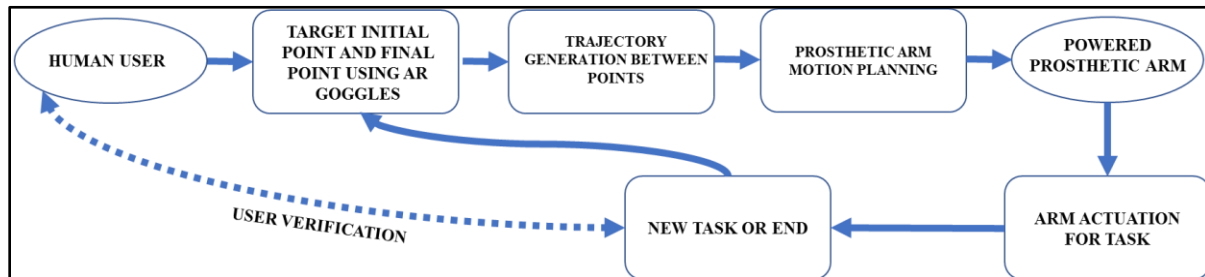


Figure 3.1: High Level System Flowchart

### 3.2 Research Objectives

Due to the amount of effort to develop such a system, this research was broken up into distinct specific objectives. For this thesis, the objectives for this portion of work are as follows:

1. Develop a sensory suite software for the advanced wearable sensor technologies to determine the telemetry information required for the robotic arm trajectories
2. Develop hardware for the robotic arm unit as well as the subsequent motion controller software
3. Integrate both the hardware and software required for the arm to function
4. Perform testing and assessment on relevant metrics such as accuracy, speed, and stability.

Successful completion of the above objectives will allow further development of the planned artificial intelligent capabilities and bring the system further closer to human subject testing.

## Chapter 4: Hanson Arm Solid Model and Kinematics

### 4.1 Hanson Arm Solid Model

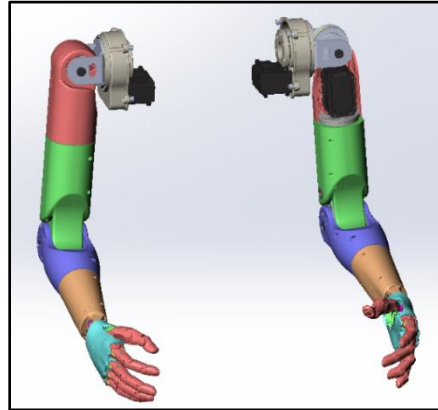


Figure 4.1: Hanson Arm 3D Model

Note: CC-BY-SA 4.0 by Copyright Holder

The base model of the arm system we are using, as shown in (Figure 4.1), was designed by Gerardo Morales as part of the development of the Sophia Robot Project by Hanson Robotics Ltd [Morales, 2018]. The arm has 7 degree of freedom (DoF) and is proportional to the typical dimensions of an adult human arm. The CAD Software utilized was Solidworks 2017, and the design fully incorporates the required hardware that is necessary to physically build the unit. For this project, only the right arm was used. Successful implementation of the right will then lead to the implementation of the left arm in future iterations.

During the evaluation of the 3D model, it was realized that significant modification was required to be able to utilize the parametric data for simulation. The first major issue was that the model was inherently too detailed, and significant effort was needed to remove extraneous parts and features to the absolute barebones required for simulation without sacrificing too

much of the inherent form and function of the design. The second major issue was that the origin framing of the individual Arm Link Models was tied to a different universal coordinate system that is not what will be used in the simulation. Each individual Arm Link Model's origin position and orientation needed to be updated to the corresponding origin position and orientation that will be used for the simulation. One effect of this was that the Solidworks Mates for assembly of the links were altered or broken for the simulation, so two distinct model assembly files would be needed for simulation and for actual physical build assembly. After the modifications were completed, the individual links were converted to a minimized resolution ".wrl" 3D format that can be used in a Virtual Reality Modeling Language (VRML) Simulation that will be discussed later in the thesis.

#### 4.1.1 Frame Assignments

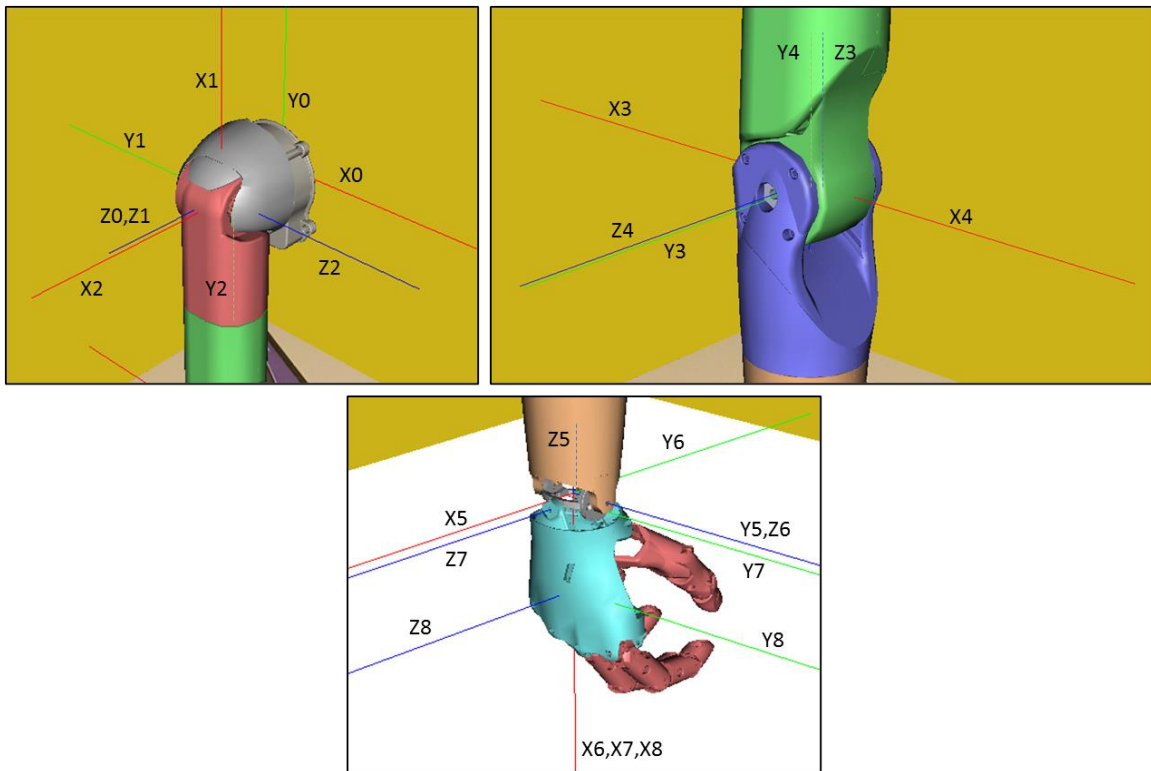


Figure 4.2: Frame Assignments for the Right Arm



The methodology of the frame assignments was to ensure that the kinematics of the arm model would apply a zero angle or “rest” state where the robotic arm is hanging freely. This had the effect of introducing offset 90-degree angles to the Denavit-Hartenburg (DH) table as shown in (Table 4.1). Furthermore, since the simulation program utilizes a parent-child relationship for connecting the arm links together, for each link connection, the subsequent orientation of the origin of the link can force the positive direction of the rotation to inadvertently flip if not carefully accounted for. This was done by tying the sign of any rotation to the axis of Frame 0 / Anchor Frame via right hand rule. (Figure 4.2) shows the final frame assignments localized at each link origin. This will be key in mapping the required kinematic parameters as shown in the next section.

#### 4.1.2 DH Parameters

To describe the arm kinematically, the arm link parameters are identified using the Denavit-Hartenburg (DH) Notation System [Craig, 2018]. The subsequent DH Parameter set up for the right arm is shown in (Table 4.1). This will be used in the subsequent evaluation of the Transformation Matrices in the next section.

Table 4.1: DH Paramaters Hanson Right Arm – Length in CM, Angles in Degrees

	$\alpha_{(i-1)}$	$a_{(i-1)}$	$d_i$	$\theta_i$
1	0	0	7.63	$\theta_1$
2	90	0	0.52	$\theta_2$
3	90	0	-25.49	$\theta_3$
4	-90	0.29	0	$\theta_4$
5	-90	0.35	23.00	$\theta_5$

Table 4.1 Continued

	$\alpha_{(i-1)}$	$a_{(i-1)}$	$d_i$	$\theta_i$
6	-90	0	0	$\theta_6$
7	90	0.5	0	$\theta_7$
8	0	5	0	0

In general, there are a total of 7 distinct mobile joints that actuate the arm up to the hand: Shoulder Flexion/Extension ( $\theta_1$ ), Shoulder Abduction/Adduction ( $\theta_2$ ), Humeral Rotation( $\theta_3$ ), Elbow Flexion/Extension( $\theta_4$ ), Wrist Pronation/Supination( $\theta_5$ ), Wrist Flexion/Extension( $\theta_6$ ), and Wrist Ulnar/Radial Deviation( $\theta_7$ ). The hand is modeled in, but it primarily functions as a static end effector to help visualize position and orientation for the purposes of this simulation.

## 4.2 Kinematics and Redundancy Resolution

### 4.2.1 Forward Kinematics and Transformation Matrices

Utilizing the DH parameters and inputting them in the general form of the Transformation Matrices for each links as shown in equation (1) [Craig, 2018], we can get mathematical representation for each arm link that determines its general rotation matrix {the upper left 3x3 portion of the matrix} and its position vector {the upper right 3x1 portion of the matrix}.

$${}^{i-1}T_i = \begin{bmatrix} \cos \theta_i & -\sin \theta_i & 0 & a_{i-1} \\ \sin \theta_i \cos \alpha_{i-1} & \cos \theta_i \cos \alpha_{i-1} & -\sin \alpha_{i-1} & -d_i \sin \alpha_{i-1} \\ \sin \theta_i \sin \alpha_{i-1} & \cos \theta_i \sin \alpha_{i-1} & \cos \alpha_{i-1} & d_i \cos \alpha_{i-1} \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (1)$$

Since we have a total of 8 links per the DH Parameters table, the overall transformation matrix needs to be calculated by concatenating all the link transformations as shown in equation (2).

$${}^0T_8 = {}^0T_1 * {}^1T_2 * \dots * {}^7T_8 \quad (2)$$

We will also concatenate each arm link progressively in order to determine the basic position matrices for each link in relation to the Zero/Anchor frame. This will be very important for the wireframe simulation that will be created using MATLAB.

#### 4.2.2 Optimized Jacobian

The Jacobian “J” matrix is the multidimensional representation of the derivatives that relate the joints’ angular velocities vector “ $\dot{q}$ ” to the end effector Cartesian and angular velocities vector “V” as shown in equation (4) and equation (5), where “q” is the joint angles of the robotic arm joints as shown in equation (3).

$$q = \{\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6, \theta_7\} \quad (3)$$

$$V = \{X, Y, Z, \omega_x, \omega_y, \omega_z\} \quad (4)$$

$$V = J \frac{dq}{dt} = J \dot{q} \quad (5)$$

Using the velocity propagation technique [Craig, 2018] with the consideration of using only revolving joints, we derive the end effector Cartesian velocities (linear and angular velocities) and relate them to the Zero/Anchor Datum Frame. Normally, inverting the Jacobian will allow the calculation of the angular velocities vector for each joint, however, the Hanson arm system has an inherent redundancy in its design since the 7-DoF nature of the arm can translate to an infinite possible solutions set when determining joint angles for any particular end effector trajectory, causing the Jacobian to be non-square and therefore not invertible. To get around this, we apply the Weighted Pseudo Inverse Method [Alqasemi et al, 2007] as shown in equation (6). In this method we introduce a diagonal “n x n” , or in our case (7 x7) for 7 joints, positively defined Weight Matrix “W”. Each value of the diagonal corresponds to a specific joint which is highly dependent on whether that joints is within the presence of joint limits or singularity.

Depending on the value, the program will preferentially move or stop moving that specific joint based off the defined ruleset that will be discussed in further detail in the next subsection. This also known as the Weighted Least Norm Optimization Solution.

$$\dot{q} = W^{-1}J^T(JW^{-1}J^T)^{-1}V \quad (6)$$

#### 4.2.3 Weight Matrix - Gradient Projection Method

To determine the weight matrix “W”, we utilize the joint limit function defined in equation (7) [Chan et al, 1995]. As the joints reach the limits, the function goes to infinity, while the function goes to 1 when the joint is in the middle between its max and min range. This automatically assigns a high weight to those joints reaching their limits so that they do not move any further. Currently, the limits of the joints were determined using the Solid works CAD model while in collision detection mode. Actual joint limits of the physical build might vary due to hardware, wire, and pulley belt considerations; hence these limits can be modified freely without any impact to core coding.

$$H(q) = \sum_{i=1}^n \frac{1}{4} * \frac{(q_{i,max} - q_{i,min})^2}{(q_{i,max} - q_i)(q_i - q_{i,min})} \quad (7)$$

The gradient projection of equation (7) takes the form in equation (8) which is used to directly optimize the weight matrix. One can note that the absolute value of equation (8) will be zero if the current joint angle is in between the joint limits, and that it will be at infinity if the current joint angle is at a joint limit. Therefore, the higher the absolute value for equation (8), the less preference to move that joint.

$$\frac{\partial H(q)}{\partial q_i} = \frac{(q_{i,max} - q_{i,min})^2 * (2 * q_{i,current} - q_{i,max} - q_{i,min})}{4 * (q_{i,max} - q_{i,current})^2 * (q_{i,current} - q_{i,min})^2} \quad (8)$$

There are a few caveats however to the gradient projection: Depending on the size of the time step that is used, it is possible that the program might still exceed joint limits under certain conditions. To counter this, we force the values for equation (8) using these conditional tests:

1. If the absolute value of the previous step is greater than the absolute value of the current step, and the current angle is within the joint's limits, then force the current absolute value to be zero. This indicates that the joint is actively moving away from a joint limit.
2. If the absolute value of the previous step is greater than the absolute value of the current step, and the current angle is above or equal the maximum limit or below or equal the minimum limit, then force the current absolute value to be infinity. This indicates that the joint is within an unsafe zone, therefore we need to freeze movement right away of that joint.
3. If the absolute value of the previous step is less than the absolute value of the current step, and the current angle is above or equal the maximum limit, or below or equal the minimum limit, then force the current absolute value to be zero. This indicates that the joint is within an unsafe zone, but it is moving back into a safe zone within the joint limits.

The gradient is summarily applied to the weighted matrix as shown in equation (9). The value "F" is the user inputted preference value for the joint which normally would be set to "1". This matrix is then applied to equation (6) to determine the resolved joint angle rates.

$$W = \begin{bmatrix} F_1 + \left| \frac{\partial H(q)}{\partial q_1} \right| & 0 & \dots & 0 \\ 0 & F_2 + \left| \frac{\partial H(q)}{\partial q_2} \right| & 0 & 0 \\ \vdots & \vdots & \ddots & \dots \\ 0 & 0 & 0 & F_7 + \left| \frac{\partial H(q)}{\partial q_7} \right| \end{bmatrix} \quad (9)$$

#### 4.2.4 Singularity Robust Inverse (SRI)

Singularities appear when an inverse is momentarily impossible due to a momentary loss of a degree of freedom due to joint alignments at that moment or if a workspace limitation is approached. In these cases, the determinant of the Jacobian reaches zero. Since the robotic arm is a redundant system, therefore non-square, to determine whether the arm is closing into a singularity domain, we need to define an objective function that represents the overall manipulability of system [Alqasemi, 2007]. This manipulability measure, defined by equation (10), showcases singularity when it equals zero, and showcases stability when the number is high. By maximizing this measure “M”, we can exploit redundancies to generate stable arm motion.

$$M = \sqrt{\det(J * W * J^T)} \quad (10)$$

Depending on the parameters of the robotic arm, the value of “M” can vary significantly for different trajectories. However, the manipulability measure is a good visualizer for the quirks of the system. As mentioned, higher values indicate stable performance, but there will be cases when even though the system is closing to a singularity such as a joint limit, the robot will still showcase stable operation. It is within this neighborhood of points where we apply a factor to the inverse calculation in equation (6) to ensure the system does not succumb to the singularity by sacrificing the positional and rotation trajectory accuracy calculated early on. This is done by figuring out two factors: The Stability Factor “w<sub>0</sub>” and an Accuracy Factor “k<sub>0</sub>”. The Stability

Factor is normally related to the Manipulability measure in equation (10) in that its value is typically equal to the highest M value for the system. The Accuracy Factor relates to the minimum value needed to avoid system instability. The factor “k” is modified according to equation (11) [Nakamura, 1991].

$$k = \begin{cases} k_0(1 - M/w_0)^2, & \text{for } M < w_0 \\ 0, & \text{for } M \geq w_0 \end{cases} \quad (11)$$

There are special cases, however, when the equation (11) is not enough to avoid system instability. In those cases, a dynamic Stability Factor is utilized as shown in equation (12) which once inputted into equation (11) significantly augments the system, sacrificing trajectory accuracy in order to facilitate motion until it reaches a more stable regime.

$$w_0 = \begin{cases} M^2/w_0, & \text{for } M/w_0 > 1 \\ w_0, & \text{for } M/w_0 \leq 1 \end{cases} \quad (12)$$

Equation (6) is then modified to account for this Singularity Robustness as shown in equation (13) where “I” is a (6 x 6) Identity Matrix. This is also known as the Weighted SR Inverse Optimization Solution.

$$\dot{q} = W^{-1}J^T(JW^{-1}J^T + k * I)^{-1}V \quad (13)$$

A modification of equation (13) is when “W” is an Identity matrix. This method will remove the joint limit avoidance and becomes known as the SR Inverse Optimization Solution. This is utilized as a comparison between the other Optimization Solutions since it applies the SR Inverse without the limitation of joint limits.

#### 4.2.5 Trajectory Generation and Matching

Determining the trajectory of the end effector will be dependent on a set of user inputted Transformation Matrices that will signify the initial position “P<sub>initial</sub>” and rotation “Rot<sub>initial</sub>”, and

the final position “ $P_{final}$ ” and rotation “ $Rot_{final}$ ” giving a linear trajectory between them. The user will then specify a speed limit “ $SL$ ” to determine the maximum cartesian end effector velocity for travelling between the two positions. From here, the cartesian distance is figured out as shown in equation (14).

$$Distance = \sqrt{(x_{final} - x_{initial})^2 + (y_{final} - y_{initial})^2 + (z_{final} - z_{initial})^2} \quad (14)$$

Once the cartesian distance is found, the time “ $t$ ” for the end effector to complete the trajectory is calculated as shown in equation (15).

$$t = \frac{Distance}{SL} \quad (15)$$

Next, the user inputs expected number of steps “ $n$ ” to carry out the motion and the resolution of the expected time change per step “ $dt$ ” is calculated as shown in equation (16).

$$dt = \frac{t}{n} \quad (16)$$

From this point, the next step is to determine the positional and rotational errors between the current position/orientation of the end effector versus the calculated trajectory for a specific time step. This will be discussed in the next two subsections: Position Errors, and Orientation Errors. The main idea is to minimize the errors between the actual hand position and rotation with the calculated trajectory.

#### 4.2.6 Position Errors

To calculate the positional errors “ $dP$ ”, the Cartesian trajectory typically is divided by the number of expected steps as shown in equation (17). This is done prior to the motion of the robotic arm. Typically, the higher the number of steps, the more accurate the expected motion for the arm, but at the cost of processing speed.



$$dP = \frac{P_{final} - P_{initial}}{n} \quad (17)$$

The next step is to determine the positional error between the calculated trajectory and the actual position of the end effector. First, the calculated trajectory must be defined as per equation (18) to relate to the specific time step “j”.

$$P_{traj}(j) = P_{initial} + j * dP \quad (18)$$

Positional error is then simply defined as the error between the current position of the hand and the expected trajectory as shown in equation (19) [Luh et Al, 1980].

$$e_{position}(j) = P_{traj}(j) - P_{hand}(j) = \begin{bmatrix} e_{px}(j) \\ e_{py}(j) \\ e_{pz}(j) \end{bmatrix} \quad (19)$$

#### 4.2.7 Rotation Errors

Rotational Trajectory will be calculated using the equivalent or Single Angle and Axis of Rotation method [Paul, 1982]. This will be a rotation matrix relating between the initial rotation matrix and the final rotation matrix inputted by the user. This matrix, which will be named “R” is defined in equation (20).

$$R = Rot_{initial}^T * Rot_{final} \quad (20)$$

where the values of “R” are shown as in equation (21).

$$R = \begin{bmatrix} n_x & o_x & a_x \\ n_y & o_y & a_y \\ n_z & o_z & a_z \end{bmatrix} \quad (21)$$

the Single Angle of Rotation “SA” is then defined as shown in equation (22).

$$SA = \text{Atan2} \left( \left( \sqrt{(o_z - a_y)^2 + (a_x - n_z)^2 + (n_y - o_x)^2} \right), (n_x + o_y + a_z - 1) \right) \quad (22)$$

The Single Axis of Rotation vector “K” is defined by its components as shown in equation (23), (24), & (25) typically good for SA values less than 90 degrees.

$$K_x = \frac{o_z - a_y}{2 * \sin(SA)} \quad (23)$$

$$K_y = \frac{a_x - n_z}{2 * \sin(SA)} \quad (24)$$

$$K_z = \frac{n_y - o_x}{2 * \sin(SA)} \quad (25)$$

note that if SA is very small, “K” must be renormalized to 1 such  $K_x = 1$ ,  $K_y = 0$ , &  $K_z = 0$ .

If SA values exceed 90 degrees, then the vector “K” is redefined as shown in equations (26), (27), & (28).

$$K_x = \text{sign}(o_z - a_y) \sqrt{\frac{n_x - \cos(SA)}{1 - \cos(SA)}} \quad (26)$$

$$K_y = \text{sign}(a_x - n_z) \sqrt{\frac{o_y - \cos(SA)}{1 - \cos(SA)}} \quad (27)$$

$$K_z = \text{sign}(n_y - o_x) \sqrt{\frac{a_z - \cos(SA)}{1 - \cos(SA)}} \quad (28)$$

Note that only the largest element of “K” is calculated by equation (26) – (28). Depending on which element is largest, a more accurate value of the remaining elements is determined by the following sets of equations:

- If  $K_x$  is largest:

$$K_y = \frac{n_y + o_x}{2 * K_x * (1 - \cos(SA))} \quad (29)$$

$$K_z = \frac{a_x + n_z}{2 * K_x * (1 - \cos(SA))} \quad (30)$$

- If  $K_y$  is largest:

$$K_x = \frac{n_y + o_x}{2 * K_y * (1 - \cos(SA))} \quad (31)$$

$$K_z = \frac{o_z + a_y}{2 * K_y * (1 - \cos(SA))} \quad (32)$$

- If  $K_z$  is largest:

$$K_x = \frac{a_x + n_z}{2 * K_z * (1 - \cos(SA))} \quad (33)$$

$$K_y = \frac{o_z + a_y}{2 * K_z * (1 - \cos(SA))} \quad (34)$$

Having an established Single Angle of Rotation will allow to segment the rotation change according to equation (35) by the time step. Note at time step 1, the value of the change is zero.

$$dSA = \frac{SA}{n - 1} \quad (35)$$

The rotational angle trajectory “dA” is then calculated to relate to the specific time step “j” is shown in equation (36).

$$dA(j) = (j - 1) * dSA \quad (36)$$

next is to convert dA to a rotational matrix format dR as shown in equation (37).

$$dR(j) = \begin{bmatrix} K_x K_x * (1 - \cos(dA(j))) + \cos(dA(j)) & K_y K_x * (1 - \cos(dA(j))) - K_z \sin(dA(j)) & K_z K_x * (1 - \cos(dA(j))) + K_y \sin(dA(j)) \\ K_x K_y * (1 - \cos(dA(j))) + K_z \sin(dA(j)) & K_y K_y * (1 - \cos(dA(j))) + \cos(dA(j)) & K_z K_y * (1 - \cos(dA(j))) - K_x \sin(dA(j)) \\ K_x K_z * (1 - \cos(dA(j))) - K_y \sin(dA(j)) & K_y K_z * (1 - \cos(dA(j))) + K_x \sin(dA(j)) & K_z K_z * (1 - \cos(dA(j))) + \cos(dA(j)) \end{bmatrix} \quad (37)$$

finally calculating the Rotational Matrix Trajectory as shown in equation (38).

$$\text{Rot}_{\text{traj}}(j) = \text{Rot}_{\text{Initial}} * dR(j) \quad (38)$$

Finding now the error between the Rotation Matrix of the End Effector and the Trajectory is defined in equation (39) [Luh et al, 1980].

$$e_{\text{rotation}}(j) = 0.5 * (N_{\text{hand}} \times N_{\text{traj}} + O_{\text{hand}} \times O_{\text{traj}} + A_{\text{hand}} \times A_{\text{traj}}) = \begin{bmatrix} e_{\text{rx}}(j) \\ e_{\text{ry}}(j) \\ e_{\text{rz}}(j) \end{bmatrix} \quad (39)$$

where “N” is column 1, “O” is column 2, and “A” is column 3 of the respective rotation matrix.

#### 4.2.8 Resolved Rate Method and Updated Forward Kinematics

The transformation matrices established by equation (2) need to be updated every time step to be able to represent the changes in position and rotation already enacted by the movement of the arm. To do this, using the errors established in equations (19) and (39), we can create a cartesian velocity vector as shown in equation (40) by dividing the errors with “dt”.

$$V = \begin{bmatrix} \frac{e_{\text{position}}}{dt} \\ \frac{e_{\text{rotation}}}{dt} \end{bmatrix} \quad (40)$$

To convert this cartesian velocity vector to individual joint angle velocities, we substitute this vector “V” into equation (13) in order to get “q̇”. In certain instances, the joint velocities can reach extremely high values especially if the robotic arm is reaching a singularity point. To prevent any potential damage to the physical robotic arm, we apply a velocity check for the current calculated values against a user predetermined angular speed limit “SpeedRev” as shown in equation (41).

$$\text{Factor} = \dot{q}/\text{SpeedRev} \quad (41)$$

Next, we find the maximum “Factor” value of the joint velocities. If any value is over 1, then it means that a joint has exceeded the speed limit and can potentially damage the physical robot system. If we force the exceeding joint or joints to a specific speed, the end effector will

start deviating away from its trajectory. In order to keep the system running smoothly, we divide the calculated velocities of the group by the maximum Factor effectively slowing down the farthest joint to the speed limit, and the other joints speed, factored down to match the percentage drop of the farthest joint as shown in equation (42).

$$\dot{q} = \begin{cases} \dot{q}/\max(\text{Factor}), & \text{for } \max(\text{Factor}) > 1 \\ \dot{q}, & \text{for } \max(\text{Factor}) \leq 1 \end{cases} \quad (42)$$

Applying the calculated velocity vector with equation (42) and multiplying out the time step, we can get the incremental change in angles for each joint as shown in equation (43).

$$dq = \dot{q} * dt \quad (43)$$

This new “dq” is added to the old “q” values from the previous step, which the new “q” then is used to update equation (2) to generate a new transform matrix, which includes the rotation and position components for the end effector.

## Chapter 5: Programming

### 5.1 Programming Overview

The main programming language chosen for this research is C++ since the SDK's for Magic Leap and the Motors both use that. While the simulations are performed in MATLAB, the functions utilized were converted to C++. This will be discussed in detail in the following sections.

An important concept of this research is to implement a wearable HMI System that can be coupled with the robotic arm hardware. Since the plan is to have the user's vision as part of the controller's input, the HMI must have a reliable eye tracking capability. It is also necessary that the HMI must have a visual feedback feature in the form of a graphical interface to keep the user aware and engaged of the working task for the robot arm. The Magic Leap was chosen for this interface since its capabilities, which will be discussed in a later chapter, matched our requirements. It also has the potential for more graphical functionality in the future due to the ongoing development work on its software.

### 5.2 Control System Flowchart

The Humanoid Arm Robotic Unit (HARU) Control System is broken up into four distinct programming blocks that work in tandem with each other to generate the motion as shown in (Figure 5.1). The code applied is in C++, but each block works off a different libraries and OS in its operation and will be discussed in detail in the following sections.

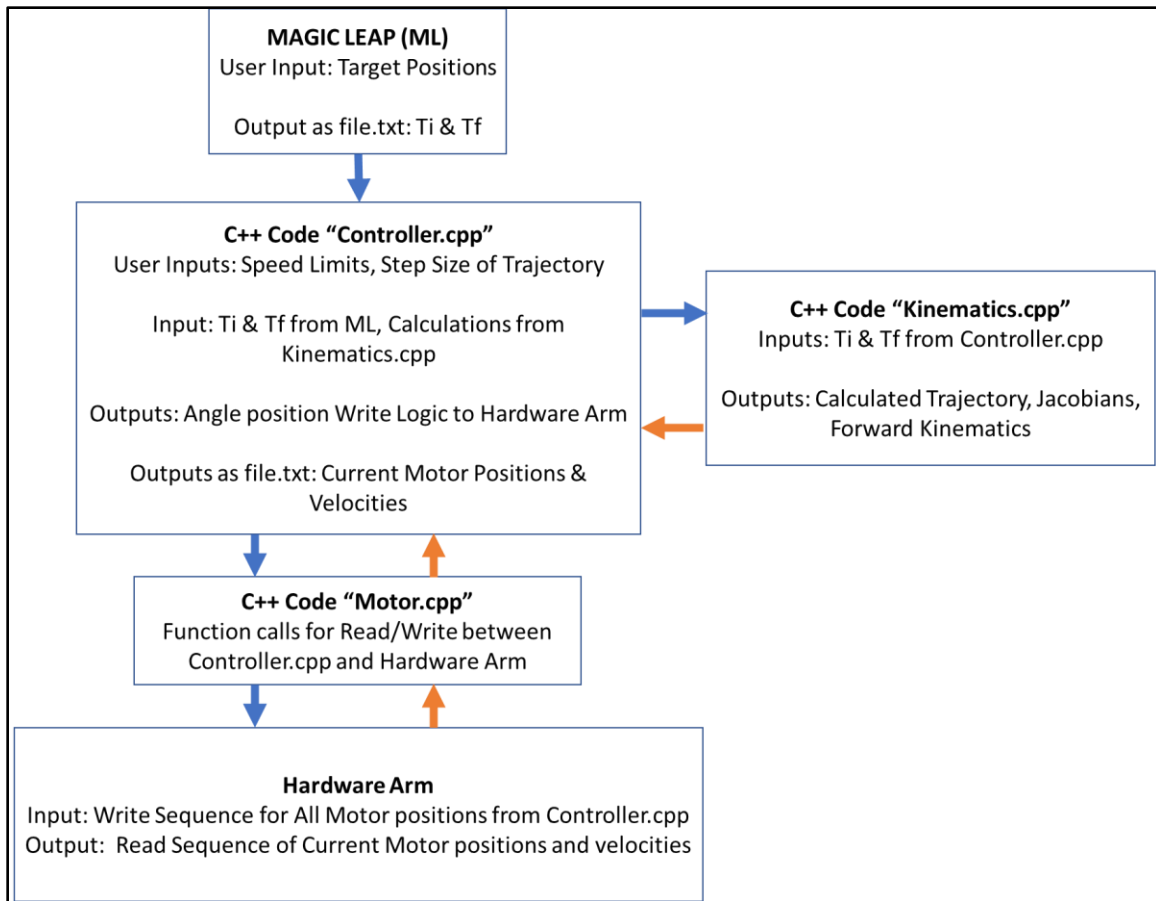


Figure 5.1: HARU Control System Flowchart

### 5.3 Magic Leap – Lumin SDK

Magic Leap utilizes its own Operating System called Lumin OS. The current version used for this project is V0.23. To help facilitate development of programs using Magic Leap, a development platform called “The Lab” is used to connect the device to the coding software program Visual Studios. Within the Lab, a subprogram called Lumin Runtime Editor is used to develop the augmented reality world components that will be used as the User Interface of our controller. Furthermore, the Lumin Libraries are geared for game development, and most of the function calls help facilitate the mathematics associated with handling 3D objects. Most of the

development work have been based off existing tutorials done by Magic Leap modified to suite our requirements.

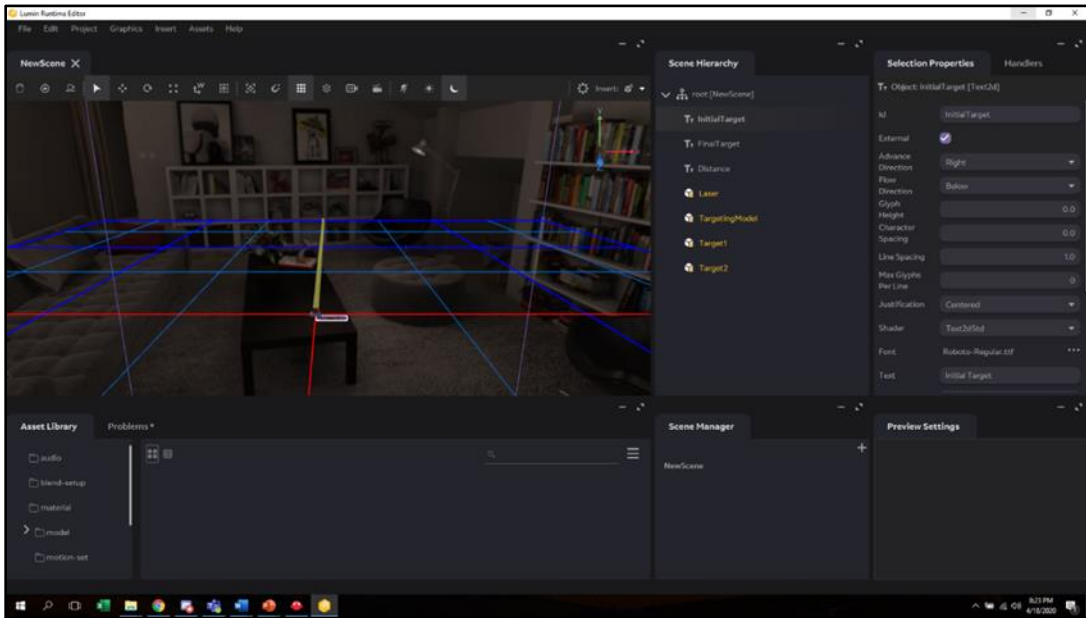


Figure 5.2: Lumin Runtime Editor

### 5.3.1 User Interface Development

Using the Lumin Runtime Editor, as shown in (Figure 5.2), we developed the augmented reality components that would be a key visual indicator of the information we wanted to present to the user as shown in (Figure 5.3). This was done utilizing Lumin’s predefined User Interface Nodes. Nodes are a 3D point information structure which serves as the origin point for displaying the 3D object in real time space. For our system, we utilized 2 distinct Nodes: A Text2D node for displaying text, and Model node for displaying primitive shapes. The Text2D nodes we utilized are meant to display the coordinate position for our initial and final positional viewing targets, and the Model nodes are meant to add a 3D marker on those subsequent points (represented as a pyramid), as well as another node showing a marker on the user’s current targeted gaze (represented as a cube). The position and orientation of these nodes are updated based off a



selected sensor value such as eye gaze or head position. This is typically done by a simple transform calculation as shown in equations (44) & (45).

$$\text{Node Position} = \text{Inverse}(\text{prismTransform}) * \text{SensorCalculatedPosition} \quad (44)$$

$$\text{Node Orientation} = \text{Inverse}(\text{prismTransform}) * \text{SensorCalculatedOrientation} \quad (45)$$



Figure 5.3: User Interface View from Headset

A typical Magic Leap Program works on the concept of world spaces called Prisms. Prisms define the coordinate system in which the program resides. This is normally placed at the discretion of the user at the start of using the program, as shown in (Figure 5.4). Wherever the user initially places the Prism space, defines the (0,0,0) coordinate of the program at that time of usage. This is different from the World Coordinate System that the Headset resides in. The headset, once turned on, defines a world coordinate system in which the origin point is the position of where the headset is located. This is important since telemetry information will need to transform between the Prism Coordinate System and the World Coordinate system regularly for proper operation of the program as most sensor values are based off the World Coordinate

System. The Node in question is then updated with the required position and orientation values that can be viewed from the Headset Display as shown in (Figure 5.3).

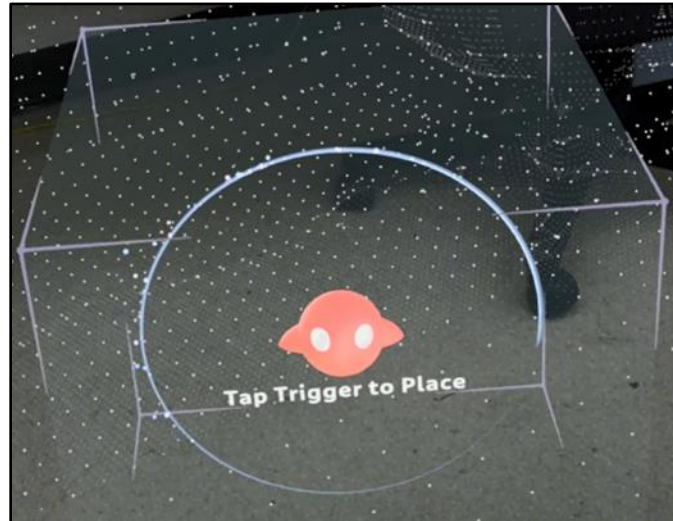


Figure 5.4: Prism Visualization

To minimize cluttering of information in the viewscreen, the program was set up into several “states”. Each state defines what information is available to be presented in the graphical interface, and any action the program needs to perform. The following states are defined as follows:

- State 0: Defines the initial start state of the eye tracking program. This is the scanning phase of the program for the Initial Position Target. A targeting reticule is visible and is tied to the eye tracking gaze location, and the program is returning the 3D position of the reticule as a visual feedback.
- State 1: Defines the Initial Position Locked state of the program. A switch trigger will lock the reticule in place, and the position is stored. The location of that Initial Target Position is then given an anchored marker.

- State 2: Defines the scanning phase of the program for the Final Position Target. A targeting reticule is visible and is tied to the eye tracking gaze location, and the program is returning the 3D position of the reticule as a visual feedback. Information from State 1 is kept visible for reference.
- State 3: Defines the Final Position Locked state of the program. A switch trigger will lock the reticule in place, and the position is stored. The location of that Final Target Position is then given an anchored marker. All Information from the previous state is kept visible for reference.
- State 4: Defines the Output file generation state of the program. All stored telemetry information from the previous states are written in a text file and stored on a local device.

Switching between the states is done using the Magic Leap's Controller buttons, but this can be modified to other switches in the future. At the end of State 4, the program will restart to State 1 and repeat the targeting loop. The number of states were chosen based of only utilizing a two-position waypoint system for generating the arm trajectory. Should more waypoints be necessary, additional states can be implemented. The following pseudocode shown in (Figure 5.5) highlights the information between the states.

**Algorithm** State Structuring for Graphical User Interface(GUI)

```
1: Start Loop
2: Set STATE = 0
3: Set All Nodes to NOT VISIBLE
4: Get SWITCH INPUT
5: if INPUT = TRUE then
6:   if STATE = 0 then
7:     Set STATE = 1
8:     Set NODE "InitialTarget" to VISIBLE
9:     Return FALSE
10:  end if
11:  if STATE = 1 then
12:    Set STATE = 2
13:    Set NODE "InitialTarget" to VISIBLE
14:    Return FALSE
15:  end if
16:  if STATE = 2 then
17:    Set STATE = 3
18:    Set NODE "FinalTarget" to VISIBLE
19:    Return FALSE
20:  end if
21:  if STATE = 3 then
22:    Set STATE = 4
23:    Write Node Information to NODE.TXT
24:    Return FALSE
25:  end if
26:  if STATE = 4 then
27:    Set STATE = 1
28:    Return FALSE
29:  end if
30:  Return FALSE
31: end if
32: End Loop
```

Figure 5.5: Pseudocode for State Structuring for Graphical User Interface

### 5.3.2 Sensor Implementation for User Interface (UI)

Part of the UI is dependent on two sensor values to provide feedback on what and where the user is looking. One feedback is the eye tracking sensor data, and the other is the head pose of the headset unit. These are discussed below:

### 5.3.2.1 Eye Tracking Implementation

Lumin's SDK already has a working library for eye tracking. This includes a calculated value of an eye gaze position fixation. This fixation point is then utilized to position the Model nodes to follow the eye gaze of the user. As the user switches states, the position is locked and can be stored for use by the Arm Controller. As the sensor value updates every loop, the Model Node position is subsequently updated in real time. The user can then effectively use a Targeting Model Node as a reticule to home into desired target objects.

There are some limitations to using the eye gaze fixation point and they are listed below:

- Accuracy of targeting for objects further than approximately 3 feet starts to significantly drop.
- Targets closer than approximately 6 inches will not be registered visually due to the field of view restriction of the display unit.
- There is a significant mental effort to use eye tracking as a locking method. This seems to be common knowledge and is even mentioned by Magic Leap to be aware of fatigue from using this for a prolonged period.

### 5.3.2.2 Head Pose Implementation

One key aspect of the UI is that it always must be persistent to be within the field of view (FOV) of the user. This means that the radial position and directional front vectors of both the Text nodes and Model nodes must face the user much like a floating billboard. If not, the nodes would stay static in 3D space. This is implemented by using the head pose information of the headset. Part of the Lumin SDK library is only a call out for the headset's Up Vector and Forward Direction Vector. However, the orientation call out for the nodes will need a quaternion format

input. We utilize a function to convert the two vectors to the quaternion format by using the following set of equations:

$$D = \text{norm}(\text{Forward Direction Vector}) \quad (46)$$

$$X = \text{norm}(\text{Up Vector} \times D) \quad (47)$$

$$Y = \text{norm}(D \times X) \quad (48)$$

$$Q_w = \frac{\sqrt{1 + X_x + Y_y + D_z}}{2} \quad (49)$$

$$Q_x = \frac{(D_y - Y_z)}{(4 * Q_w)} \quad (50)$$

$$Q_y = \frac{(X_z - D_x)}{(4 * Q_w)} \quad (51)$$

$$Q_z = \frac{(Y_x - X_y)}{(4 * Q_w)} \quad (52)$$

The subsequent  $Q_w$ ,  $Q_x$ ,  $Q_y$ , &  $Q_z$  are then used to update the orientation of the Nodes to ensure proper placement of texts and models in the User Interface. Minor modification to the sign of the values are updated depending on the set up of the nodes until proper orientation is achieved.

### 5.3.2.3 Data Filtering

The Targeting reticule node takes its position values based off the eye tracking sensor from the headset. Output from the eye tracking sensor refreshes every 30 Frames per second, which translates to a very jittery output if the raw value is directly taken. To minimize the effect of jitter, two strategies are implemented. The first is to create a confidence threshold for the eye tracking which will only register the eye gaze position if the confidence is above this number.

This confidence value is already available as a function call and the value of 0.90 is used. This

effect filters out raw value eye tracking data that might be considered outliers. The second is to create a Finite Impulse Response filter in the form of an averaging filter to smooth out the sensor values to effectively reduce the jittery movement of the targeting model. This is achieved using the following pseudocode shown in (Figure 5.6):

<p><b>Algorithm</b> Finite Impulse Response Filter</p> <ol style="list-style-type: none"> <li>1: Start Loop</li> <li>2: Set FilterBuffer Max Size</li> <li>3: <b>if</b> FilterBuffer == Size <b>then</b></li> <li>4:     For i = 0 to (Size - 1)</li> <li>5:         FixationValue = average of FilterBuffer values</li> <li>6:         Reset FilterBuffer Size to 0</li> <li>7:     <b>end if</b></li> <li>8: Add New Sensor Value to FilterBuffer</li> <li>9: End Loop</li> </ol>
---

Figure 5.6: Pseudocode of Finite Impulse Response Filter

#### 5.3.2.4 Telemetry Logging

Final outputs of the Node information are currently written to an output file, as shown in (Figure 5.7) once the user switches from State 3 to State 4 of the program. This output would be then read by the Arm Controller to complete the trajectory calculations.

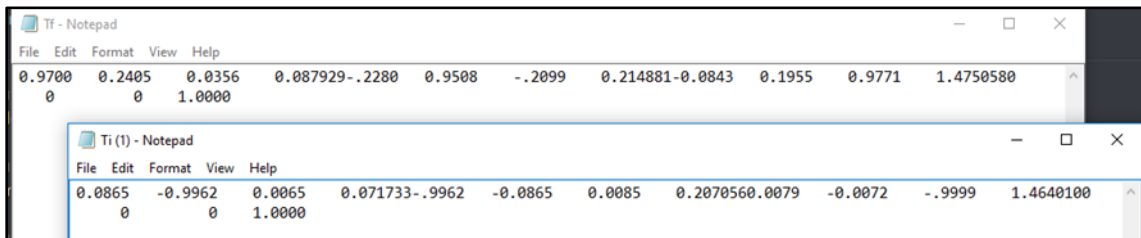


Figure 5.7: Text Output from Magic Leap

## 5.4 Controller C++ Code

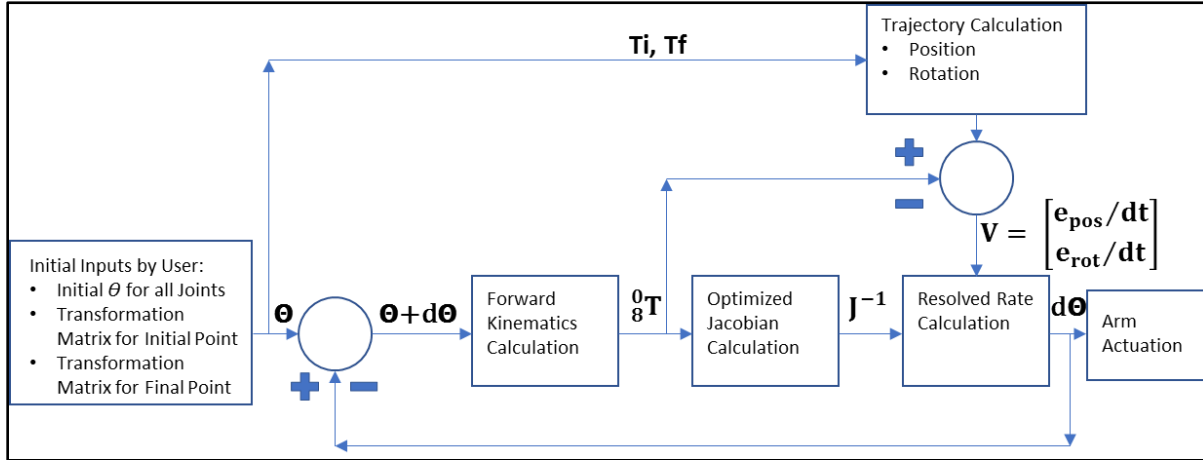


Figure 5.8: Code Flowchart

The main program initially used for the control system is MATLAB R2013a primarily due to the current availability of using the Simulink Simulation environment for the 3D simulation. This was then converted to C++ Code to facilitate the connection with the hardware motors for the robotic arm. The overall MATLAB code is broken up into several different high-level functions listed below:

- **DH Parameters Function:** Inputs the angle values, and outputs a global DH Matrix that will be utilized in several other functions.
- **Trajectory Function:** Inputs the initial and final User defined Transformation Matrices and outputs the trajectory matrix which includes the calculated Position and Rotation Matrices per time step.
- **Forward Kinematics Function:** Inputs from the DH Parameter function and calculates the subsequent transformation matrices for the whole arm. Outputs the Position Matrices for each link in reference to the zero frame.



- Joint Limit Function: User inputted physical joint limits that sets the maximum and minimum joint angles of the system that will be used in the Weight Matrix Gradient Projection calculation of the Jacobian Function.
- Optimized Jacobian Function: Inputs from the Forward Kinematics Function and Joint Limits Function and begins the optimized Jacobian calculation, including the weight matrix gradient projection and the SR Inverse. Outputs the Jacobian pseudoinverse to be use for the resolved rate calculation.
- Main Code Function: Main loop of the program incorporating the Optimized Jacobian, Forward Kinematics, Trajectory, and Animation Functions. Also includes the calculation for the resolved-rate solution. The main code follows the block diagram flow as shown in (Figure 5.8).

#### 5.4.1 Kinematics.cpp

The conversion of the MATLAB functions to C++ required combining the high-level functions, as discussed in the preceding section, into a block specifically for handling the calculations. This was done to simplify debugging and expedite the verification of the mathematics. Some of the functions created are direct conversions from MATLAB, however, new functions were created for to simplify the workflow of the calculations:

- Kinematics.initialStart: Input to this function is the  $T_i$  &  $T_f$  matrices derived from Magic Leap, as well as user inputted Speed Limits for calculating the time the program needs to complete the motion. This function serves as the initial check to ensure the matrix inputs are properly populated, and it is also responsible for

converting between the Bitwise reading from the motors (along with its required gear ratios) into a radian value.

- `Kinematics.updateTheta`: Considered the main code that handles the Jacobian calculation, the Forward Kinematics, and a reversion from radian values to Bitwise values output required for the motors. Inputs to this function are the previously calculated joint angles from a preceding loop, and the calculated trajectory for the arm motion. Output is the Bitwise value of the calculated angle.
- `Kinematics.getJacobian`: A subfunction within the `updateTheta` function. This function is responsible for the Optimized Jacobian, Joint Limit Avoidance. Input to the function are the previously calculated joint angles from a preceding loop, and the output is the Jacobian Inverse.
- `Kinematics.forwardKinematics`: A direct conversion of the Forward Kinematics function from MATLAB.
- `Kinematics.findDHTable`: A direct conversion of the DH Parameter Function from MATLAB.
- `Kinematics.getTrajectory`: A direct conversion of the Trajectory Function from MATLAB.

#### 5.4.2 Motor.cpp and Dynamixel SDK

Since the Joint Motors used are manufactured by Dynamixel, the primary control program must use the Dynamixel SDK C++ Libraries provided by ROBOTIS [Robotis, 2020]. From the `Kinematics.cpp`, the main process of actuating the motors is through angular position control

which means the input to the motors are specific angles. As such, from the Dynamixel Libraries, three main functions were created to facilitate this control scheme:

- `Dynamixel.moveTogether`: This is considered another main function of the controller. This function activates the Torque setting on all the motors at the same time, moves the motors to the inputted angle, and does stall detection for each motor all at the same time.
- `Dynamixel.setParams`: Another key function that alters the characteristics of the motors. This function sets a value to a specific parameter in a motor depending on which address on its electrically erasable programmable read-only memory (EEPROM) table is called. This can be used to change maximum velocity values or maximum acceleration values for better system stability as an example.
- `Dynamixel.getAllValues`: This is the read function that detect the current value of the parameter we are interested in. This is used for reading position and velocity values used for data analysis during testing.

As a note, the Dynamixel Motors in our unit all have an in-built position encoder that determines the current Angular position of the joint. The resolution of the Encoder is 4095 Pulses/Revolution. This entails that our angular resolution is approximately 11.375 degree per pulse. This is critical since input to the motors are in integers with lower resolution, while calculated angles are far more precise which can lead to positional accuracy issues

#### 5.4.3 Controller.cpp

This block is responsible for tying all the other programs to facilitate Arm motion. This main code is responsible for the trajectory generation, and the reading and writing of files. The

flow chart of the system is as shown in (Figure 5.9). The main checks to the hardware, as well as the reading and writing of information to motors are handled by this block.

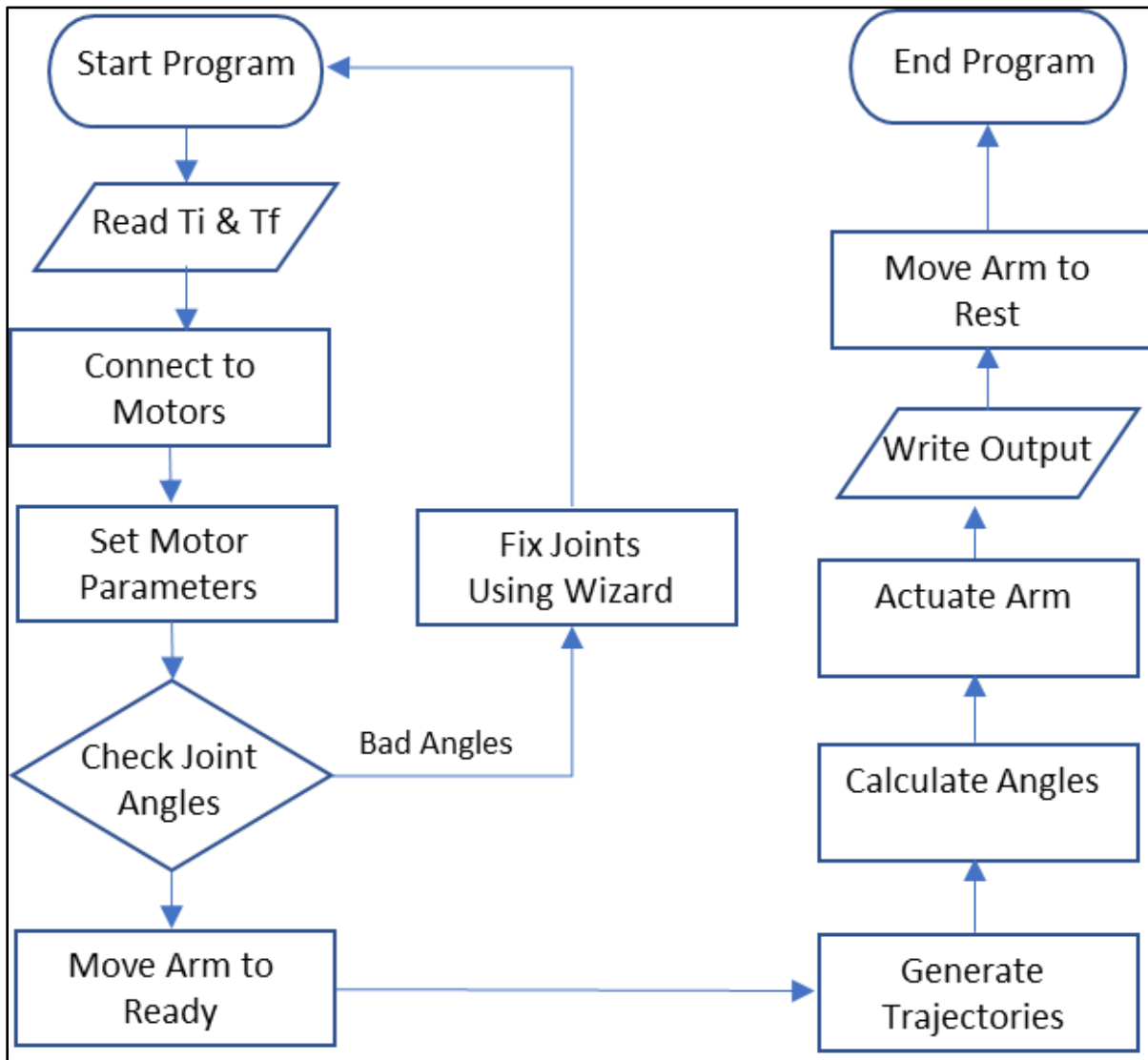


Figure 5.9: Controller.cpp Control Flow

## Chapter 6: Hardware Development and System Integration

### 6.1 Magic Leap



Figure 6.1: Magic Leap Headset

For the visual sensor, we utilized Magic Leap, shown in (Figure 6.1), which is off the shelf wearable head unit worn similarly to glasses [Magic Leap, 2019]. The headset weighs 316 grams and has a Field of view of 50 degrees. It is powered by a portable computing unit called the Light Pack as shown in (Figure 6.2). Weighing 415 grams, this unit has a 6 core CPU, Nvidia CUDA Graphics Card, and can run at 1.7 Ghz. All programs are uploaded into the Light Pack.



Figure 6.2: Magic Leap Light Pack

While the Magic Leap has a multitude of sensor equipment and capabilities, only the relevant systems to this project will be discussed, primarily the following:

- Eye Tracking
- Head Tracking
- Depth/Room Perception

Integrated to the head unit are eye tracking sensors that track the pupil orientation via infrared reflection. A gyroscope and accelerometer sensors are also installed on the unit for determining head orientation and movement; this information is necessary in order to keep the GUI always facing the user especially when the head is moving around.

### 6.1.1 Eye Tracking

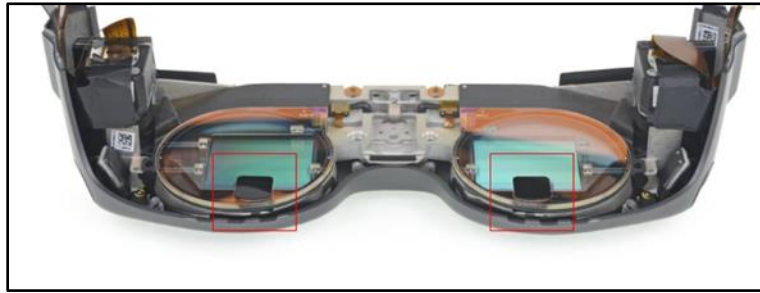


Figure 6.3: Infrared Sensor Location

Note: BY-NC-SA-3.0 by Copyright Owner.

As mentioned, the Magic Leap has two IR Omnivision CameraCubeChip camera sensors imbedded in each lens behind a dark filter as shown in (Figure 6.3) [iFixit, 2020]. Working off the principle of infrared reflection off the cornea and the pupil of the eye, a gaze direction can be calculated for each eye. If the user focuses on a point, through the concept of stereo scoping parallax, a fixation point in space can be estimated. The eye tracking sensor and the fixation point currently update the reading at 30 frames per second. Due to the placement of the sensors below the lens, it was discovered that this may potentially impact the accuracy of the eye tracking, especially if the headset utilizes multiple users or if the person uses spectacles. To minimize this issue, an eye tracking calibration event must be performed at start up for every new user.

### 6.1.2 Head Tracking

Located in the Headset is an integrated IMU for determining head pose. While information about its exact location and manufacturer is scant, most of the onboard systems converge into an electronics board area in the middle of the headset unit as shown in (Figure 6.4). During start up, Magic Leap takes the boot up position of the headset and uses that as the

world origin for that session. All world or relative coordinate systems will be based off that initial point. From the IMU sensor, we can derive an Up Vector and a Forward-facing Vector to determine the head tilt and view direction of the user. These values are also used to ensure the augmented reality 3D objects or text the user is seeing are persistently within the user's field of view.

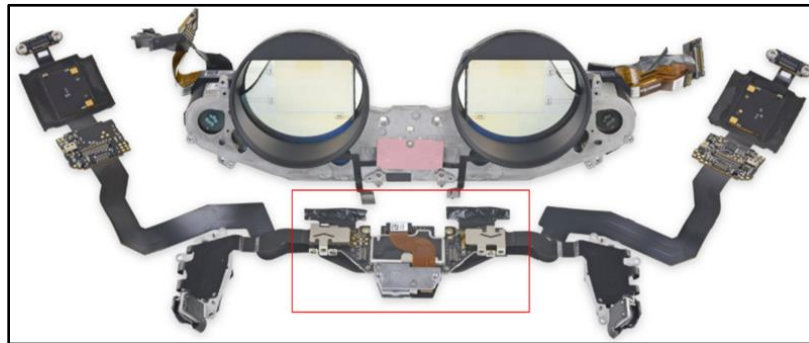


Figure 6.4: Electronic Board Location  
Note: BY-NC-SA-3.0 by Copyright Owner.

### 6.1.3 Depth Tracking



Figure 6.5: IR Projector Location  
Note: BY-NC-SA-3.0 by Copyright Owner.

Located at the Nose Bridge Support as shown in (Figure 6.5), this utilizes an Infrared Projector to mesh out a room, and a subsequent IR sensor to detect the reflected IR wave.



Extremely useful in mapping out obstacles or large objects, this sensor helps store a landscape map that determines virtual boundaries in which the AR information must adapt to.

## 6.2 Robotic Arm Hardware Development

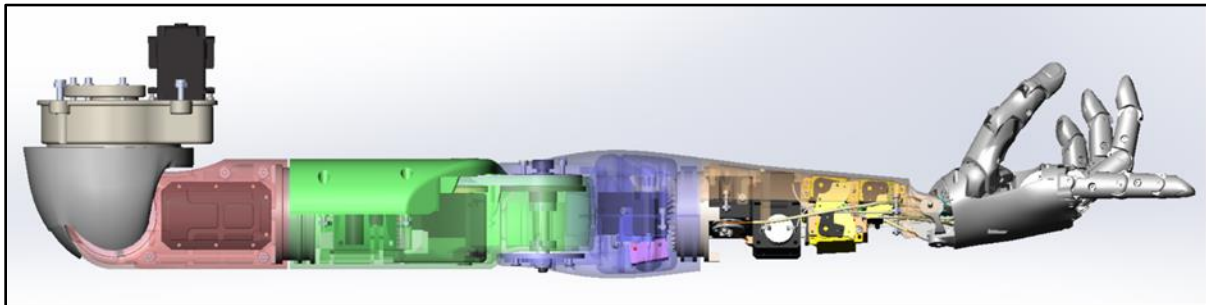


Figure 6.6: Hanson Arm See Through Top View

The base model of the arm system we are using, as shown in (Figure 6.6), was designed by Gerardo Morales as part of the development of the Sophia Robot Project by Hanson Robotics Ltd [Morales, 2018]. The arm has 12 degree of freedom (7 up to the wrist and 5 for hand/digit actuation) and is proportional to the typical dimensions of an adult human arm. The CAD Software utilized was Solidworks 2017, and the design fully incorporates the required hardware that is necessary to physically build the unit. For this project, only the right arm was used primarily to determine the feasibility of the design, and to minimize cost.

### 6.2.1 Housings

Housings for the arm was 3D printed using the 3D models as base reference. Most of the housings follow a “clam shell” type design to enclose the parts and motors. Three materials have been used in this build as discussed below:

- Fused Deposition Modeling (FDM) Polyethylene Terephthalate-Glycol (PETG) – Initially chosen for most of the housings since it provided a relatively cleaner part than the available materials of Nylon or Polylactic Acid (PLA) as shown in (Figure

6.7). Much of the issue from other materials stemmed from post print processing and clean up, and PETG offered the least post processing during that time. One main drawback, however, was the shrinkage rate of the material after printing. We estimate a roughly 2% - 3% shrinkage for the housings which caused extremely tight fits for certain parts. Other issues were also discovered, which will be discussed in detail in the next section.

- FDM Acrylonitrile butadiene styrene (ABS) – This material became the primary material of 3D printing choice due to its resistance to breaking and cracking, as well as its capability to produce high resolution prints. In areas of the arm that will potentially see higher stresses due to motion, key components have been replaced with this material as shown in (Figure 6.8). However, the main downside of this material is that it is very difficult to print with as it needs almost optimum environmental conditions during the print process to avoid any imperfections in the final part.
- Stereolithography (SLA) Resin – This method and material was chosen primarily for parts that needed an extremely high level of resolution and that does not see any significant stresses. Initial high parts, such as fingers or joint supports, that were printed in PETG or ABS resulted in un-usable parts. Though significantly more expensive (5X versus the normal PETG print cost), the new parts showed better quality as evidenced in (Figure 6.9). The main drawback, however, is that the parts need more curing time after printing to achieve its highest strength.

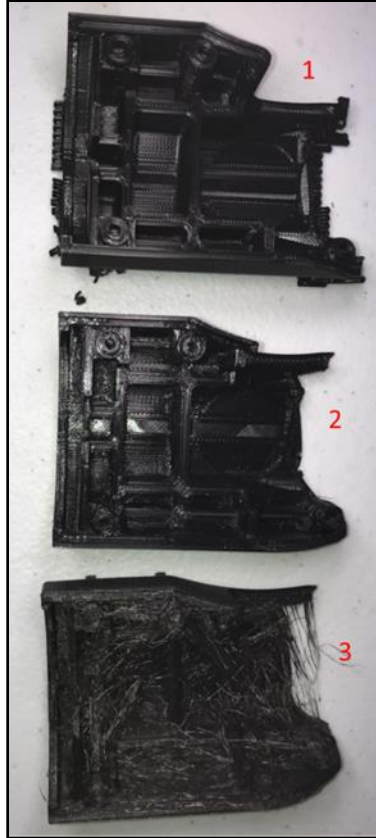


Figure 6.7: FDM 3D Print Material Comparison: (1) Nylon, (2) PETG, (3) PLA

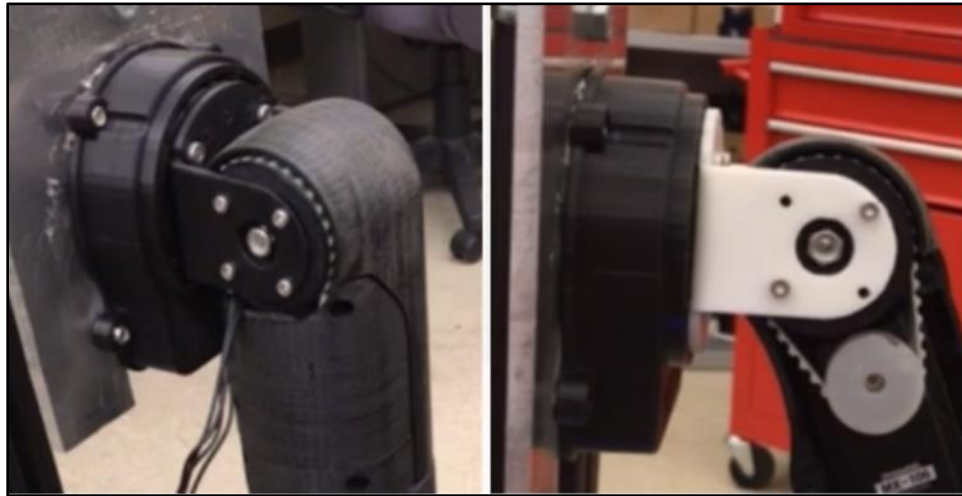


Figure 6.8: Bracket Construction Comparison between PETG (Left, Black Color) versus ABS (Right, White Color)



Figure 6.9: Finger Construction Comparison between PETG (Left, Black Color) versus SLA Resin (Right, Grey Color)

### 6.2.2 Hardware Build Progression

The build of the robotic arm is based off the Solidworks model developed by Morales. In further review of the design, it was determined that significant items were incomplete, and that the design was at best at a prototype level. It became understood that there will need to be on-the-fly adjustments to the design as the build progresses to address any gaps or inconsistencies. A detailed breakdown of the cost will be discussed in a subsequent section.

Priority was to order the equipment and housings for the arm. The Bill of Materials (BOM) was generated based off the model information, then a Make/Buy analysis was conducted to determine which parts will be off the shelf bought items or custom-made parts. It was determined that the hardware design needed 163 distinct parts, with some parts reporting up to 5 subassembly levels down from the main assembly.

Most of the custom-made parts were 3D printed housing of varying sizes and material depending on the part as discussed in another section. We single sourced the custom 3D parts to USF's Advanced Visualization lab with an approximately 1-2-week turnaround time for making the parts or the parts were printed on personal 3D printers. Upon receiving a part, significant

post processing (sanding, cutting, filing, smoothing, grinding) was required to remove any connected support structures, as shown in (Figure 6.10), or expand certain locations with extremely tight fits.



Figure 6.10: ABS Housing with Support Structures from 3D printing Still Fused to the Housing

In parallel to the receiving of parts and 3D print post processing, a test stand support structure known as the “Base Structure” was designed and built for supporting the arm. The structure utilized an 80/20 style frame members, with a mounting plate for holding the arm as shown in (Figure 6.11). This enables portability for the unit in the case of demo showing or transferring to another location.

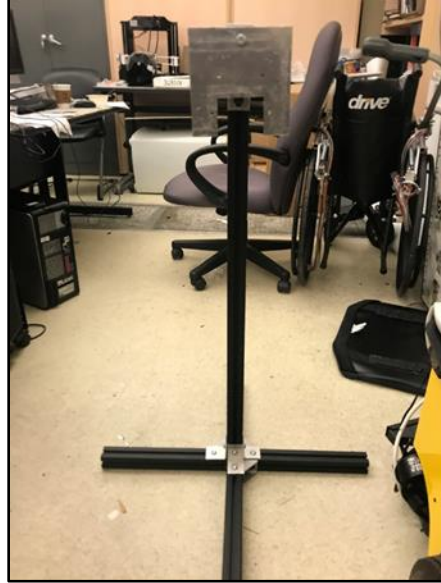


Figure 6.11: Base Structure for Robotic Arm

Prior to installing all the requisite parts and motors, an initial fit up of the overall arm was conducted to have an idea of the overall motion and workspace of the arm as shown in (Figure 6.12). This set up was also used to determine how to proceed with the motor installation by breaking up the subsequent parts of the arm into 5 “regions” as outlined in (Table 6.1) that roughly relate to the joint number to facilitate parallel installation of motors. Note that for each Region, an anchor point must be physically established with the preceding region to facilitate motion with the exception for Region 1 which is anchored to the test stand.

The progression sequence was to go from Shoulder Motors down to the wrist motors, and subsequently test the motors along the way. It is to be noted that each degree of freedom is actuated by a motor and linkage system as detailed in (Table 6.1).



Figure 6.12: Fit Up of Housings for Arm

Table 6.1: Linkage Type per Joint

Region #	Arm Joint #	Motor Description	Manufacturer	Linkage System	Ratio
1	1	MX-106R	Dynamixel	Timing Belt Pulley	1:1
2	2	MX-106R	Dynamixel	Timing Belt Pulley	2:1
3	3	XM430-W350-R	Dynamixel	Internal Gear	2.88:1
4	4	MX-64AR	Dynamixel	Timing Belt	2:1
4	5	XM430-W350-R	Dynamixel	Bevel Gear	2:1
5	6	XM430-W350-R	Dynamixel	Wire Pulley	1:1
5	7	XM430-W350-R	Dynamixel	Wire Pulley	1:1

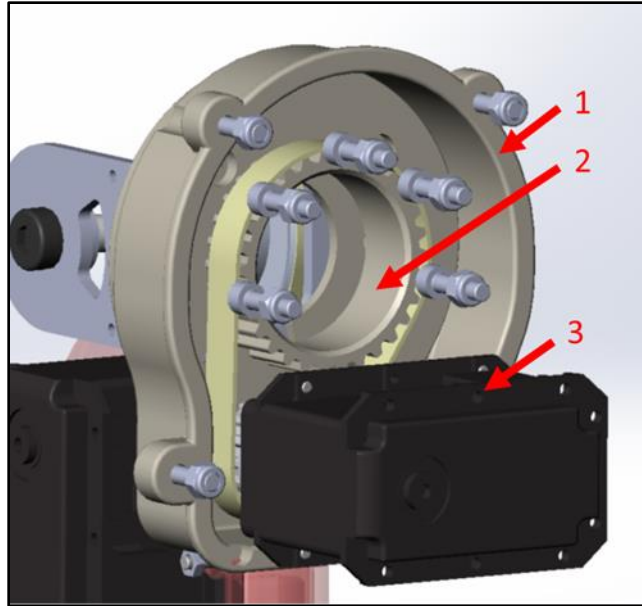


Figure 6.13: Region 1 Major Components (1: Housing, 2: Pulley Block, 3: Motor)

Region 1's purpose is to actuate the arm's Shoulder Flexion/Extension. The design involves a direct connection to the test structure as well as a set of matching housing that freely rotates on a central axis which marked the axis of rotation for that joint in the Kinematic evaluation. Region 1, as shown in (Figure 6.13), consists of 3 main parts, the mount housing, item 1, serves as the anchor connection between the test stand and rotating portions. Item 2 is the driven rotating pulley block, and Item 3 is the driving motor which is anchored. This area poses the highest levels of torques, especially if the arm motion showcases full arm extension, so techniques to minimize slippage in the driving mechanism must be implemented. This was solved by changing out the pulley to a direct drive interface as shown in (Figure 6.14).





Figure 6.14: Direct Drive Interface for Motor 1

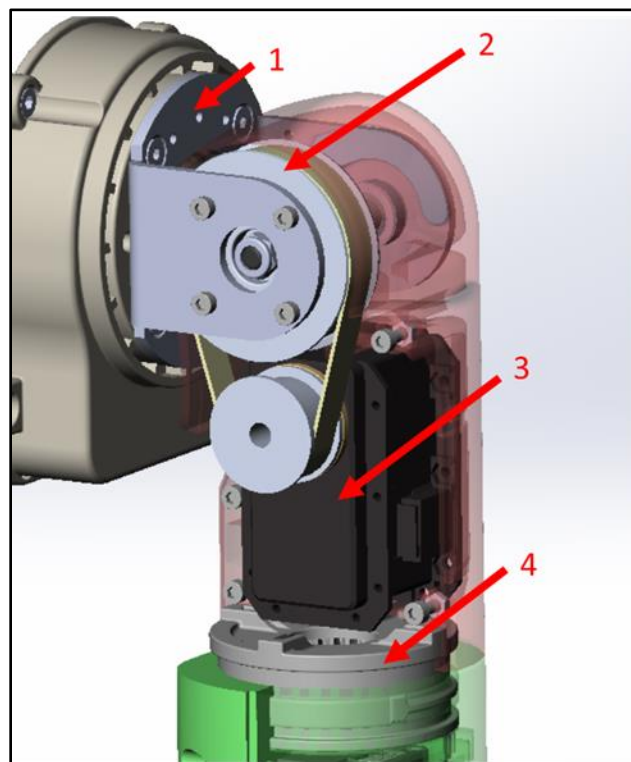


Figure 6.15: Region 2 Major Components (1: Bracket, 2: Pulley Block, 3: Motor, 4: Driven Gear)

Region 2 is designed to actuate the arm's Shoulder Abduction/Adduction motion. Consisting of 4 main parts, the actuation of the motion is done by anchoring the pinned shoulder axis, and having the driving motor float around pin as shown in (Figure 6.15). The main connection to Region 1 is a bracket (Item 1) connected to the rotating pulley block. This bracket

is a critical piece since it is a mechanical torque transmission point essentially carrying all the load of the arm and its accompanying motion stresses. Currently, the temporary material used for its construction is ABS just due to availability, however, we will switch it out to a metal construction as soon as it is available. Item 2 is the driven pulley block with the design intention of being anchored to Item 1 of which means the part does not spin freely. This rests on a pinned connection serving as the axis of rotation for the joint. Item 3 is the driving motor with pulley block, which is directly coupled with the overall housing of Region 2 in which actuating the motor will make the whole housing freely move about the pinned axis. Lastly, Item 4, is the driven gear for Region 3, directly coupled with the housing of Region 2 & Region 3 which serves as the anchor point in which Region 3 rotates from.

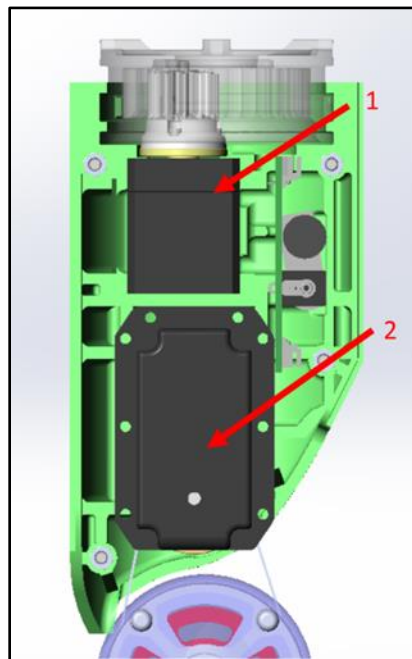


Figure 6.16: Region 3 Major Components (1: Drive Motor, 2: Drive Motor)

Region 3 is designed for the Humeral Rotation motion of the arm. The main axis of rotation for this joint is the centerline of the driven gear which connects both Region 2 and Region

3. This region consists of two main components as shown in (Figure 6.16). Item 1 is the driving motor connected to an inner gear system and is directly coupled with the housing. Note that the driven outer gear (Item 4 from Region 2) is not directly connected to the overall housing of Region 3, enabling the whole housing to “float” and rotate on that axis of rotation. Item 2 is the driving motor for the next joint motion, Elbow Flexion/Extension, and is anchored with Region 3’s housing. Region 3 also has space to allow storage of a motor controller board, but for this build, that space was not utilized since the motor controller board was placed elsewhere.

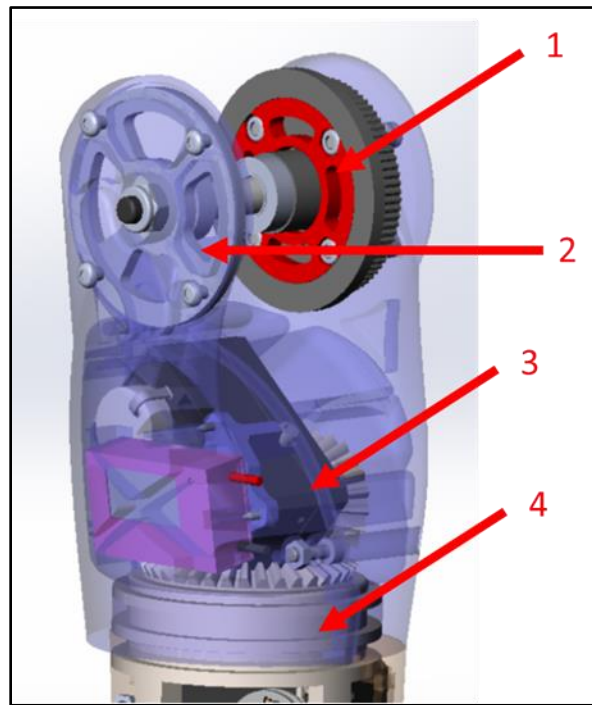


Figure 6.17: Region 4 Major Components (1: Pulley Block, 2: Bracket, 3: Motor, 4: Driven Gear)

Region 4 is designed for two joint motions: Elbow Flexion/Extension and Wrist Pronation/Supination. This region consists of 4 main components to achieve those motions as shown in (Figure 6.17). Item 1 is the driven pulley block for the elbow motion actuated by the driving motor from Region 3. This pulley block is connected to the region’s housing utilizing Item 2, a bracket/pin connection, and this pin’s centerline serves as the axis of rotation for the motion.

Region 3's housing also overlaps this pin connection, and this is what ties both regions together. Item 3 is the driving motor with a miter gear for the wrist twist motion. Coupled with Item 4 which is the driven gear of this assembly, this part floats to allow for free rotation within the housing. As a note, Item 4 is also anchored to the housing on Region 5.

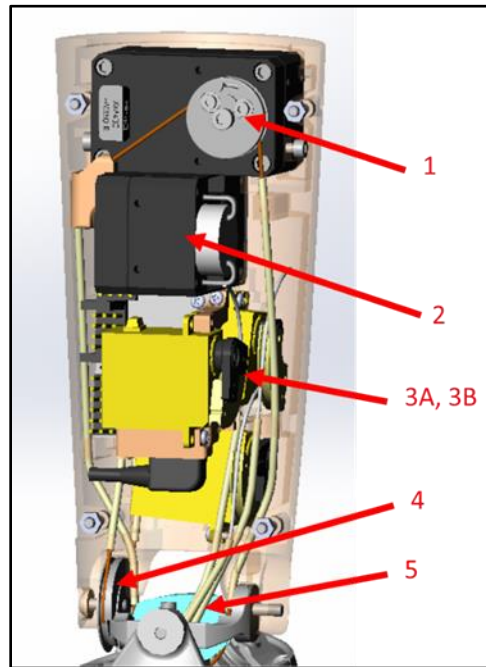


Figure 6.18: Region 5 Major Components (1: Pulley Block for Joint 7, 2: Motor for Joint 6, 3A/3B: Motor for Finger Actuation, 4: Pulley Block for Joint7, 5: Pulley Block for Joint 6)

Region 5 hosts the greatest number of motorized components as this region holds some of the motors to actuate the finger movements as well as the wrist motion for Flexion/Extension and Ulnar/Radial deviation. Most of the motors in this region will utilize a wire pulley system which will make the area congested with wires snaking its way to its intended locations. All the motors are anchored with the housing. There are 5 main components as shown in (Figure 6.18): Item 1 controls Joint 7, Item 2 controls Joint 6, Items 3A and 3B are motors for Finger motion in the hand region, Item 4 is the support structure with a pulley block whose centerline being the axis of rotation for moving Joint 7, and Item 5 is the pulley block for controlling the motion for

Joint 6. Note that since the current controller design is not meant to implement finger motion, Item 3 has been removed for the time being to optimize weight and space.

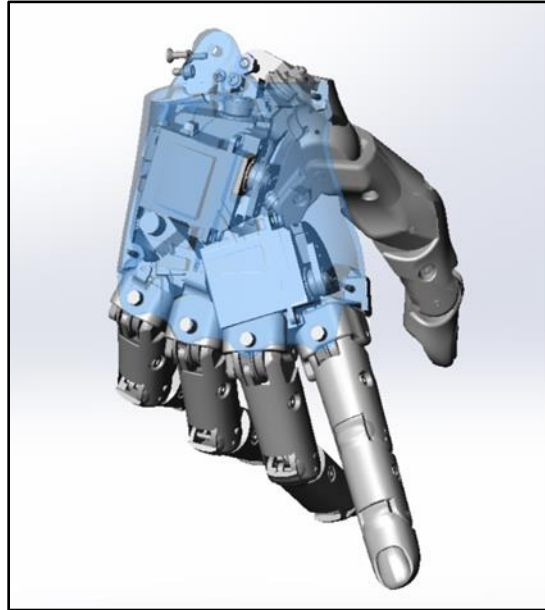


Figure 6.19: Region 6 Hand Model

Region 6, the hand area as shown in (Figure 6.19), is assembled primarily to act as a visual reference for the arm. As mentioned, the current controller's design only accounts for the 7 degrees of freedom for arm motion, and it does not apply any motion control for the fingers. To optimize weight and space, the subsequent finger motors are not installed yet for this level of the project.



Figure 6.20: Assembled Arm

Put together, as shown in (Figure 6.20), the arm is approximately 27 inches long, roughly 25 lbs in weight.

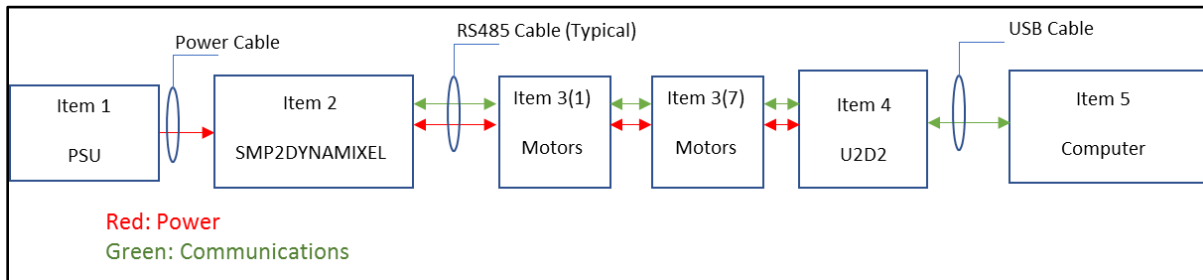


Figure 6.21: Typical Wiring Setup for Motors

Motor wiring setup for the Arm involves daisy chaining the motors in a serial configuration as shown in (Figure 6.21). This set up utilizes Dynamixel's U2D2 device, Item 4, which connects to the computer via USB. Power to the motors is provided using an external power supply, Item 1, which connects to the SMP2Dynamixel Board, Item 2, that acts as the power distributor for the motors. All the motors are serially daisy chained in order, connecting to the U2D2 device, which will provide data/communications to actuate the motors. The motors use a 4 pin RS485 cable

for power and communications. The default baud rate used for the communications with USB cable is 57600 bps.

### 6.2.3 Lessons Learned

During the hardware build, several items of improvement were recognized to improve the performance of the system and optimize the build of the assembly. These are discussed below:

1. 3D Printing Housing Shrinkage – The major factor in installation was accounting for the overall shrinkage of the 3D printed housing once received. As already mentioned, it was generally understood that thermoplastic printed materials shrank roughly 2-3% after printing, but upon initial inspection of the larger 3D printed pieces, this was not as evident. Only when more components began to populate the housing is when this issue came to the forefront. The option at that point was to reprint the housings to account for the shrinkage, or to use as is. The decision was to choose the latter option for the sake of continuation of the build. This had the effect of certain parts fitting extremely tight, which would periodically cause stress fractures if certain screws were fully tightened. A key design improvement is to determine prior to the build what would be the necessary volume scaling for every part to minimize any tight fit installs.
2. 3D Printing Parameterization “Infill” – Infill is the amount of material within the volume of an FDM 3D printed part. This is usually determined as a percentage of volume. The primary characteristic of infill is correlated to the overall strength of the part. This does come at a drawback of longer print times and more material

usage. As a rule of thumb, any part that will see any significant stresses will have 60-80% infill, while aesthetic parts can be 30%. This specifically relates to the components that see significant wear such as the beveled gears. Initial build had the infill at 30% which cause severe part failures. Currently infill at those areas are 100%.

### 6.3 Cost

The breakdown of the Cost is as shown in (Table 6.2). Approximately 40% of the cost is in the motors alone. A cost down analysis through alternative manufacturers was done on these motors potentially saving about \$800, however it was decided for the initial build, that the current motors would be vetted out. The 3D printed housing cost is an estimate based off volume and it include some of the geometries that are meant to be custom fabricated from metal which can potentially increase the cost of the overall system.

Table 6.2: High Level Cost of Project

High Level Cost Breakdown	
Description	Cost
Motors + Control Boards	\$3,000
Hardware (Nut/Bolts/Screws/Etc)	\$500
3D printed housings (Estimated)	\$1,200
Test Stand	\$150
Magic Leap	\$2,300
Total	\$7,150



## 6.4 Integration of Hardware and Software

### 6.4.1 Low Power Testing (LPT)

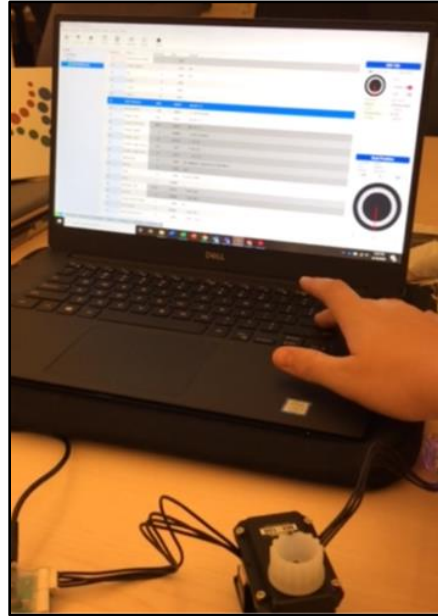


Figure 6.22: Dynamixel Wizard Motor Test

Low power testing (LPT) of the motors involved using only a 12volt/2 amp power supply to power the daisy chained motors and actuate each motor individually using the Dynamixel wizard as shown in (Figure 6.22). Through the LPT, key parameters that impact the code were identified and addressed as discussed below:

- Reverse Mode Setups: Positive axis rotation of the motors does not match the positive axis rotations of the frames established from (Figure 4.2). Joints 3,4,6, and 7 were identified to need to operate in reverse mode in order to match the same joint motions from the MATLAB simulations.
- Velocity Profiles: In position control mode, when given a goal position the motors will attempt to complete the actuation in the shortest time possible. This means the motors will rotate at its maximum velocity which causes damage to the whole

arm. Testing indicated that an angular velocity of 5 rpm a safe speed for operation and is implemented in the code.

- Joint Limits: During LPT, it was found that certain joints have a much more restrictive allowable angle zones than what was found in simulation. This was caused by tight tolerance, friction, or limitation on wiring. Exceeding these limits can potentially damage the unit that will need time for repair. The new Joint Limits for each joint was found implemented in the code.

#### 6.4.2 High Powered Testing (HPT)

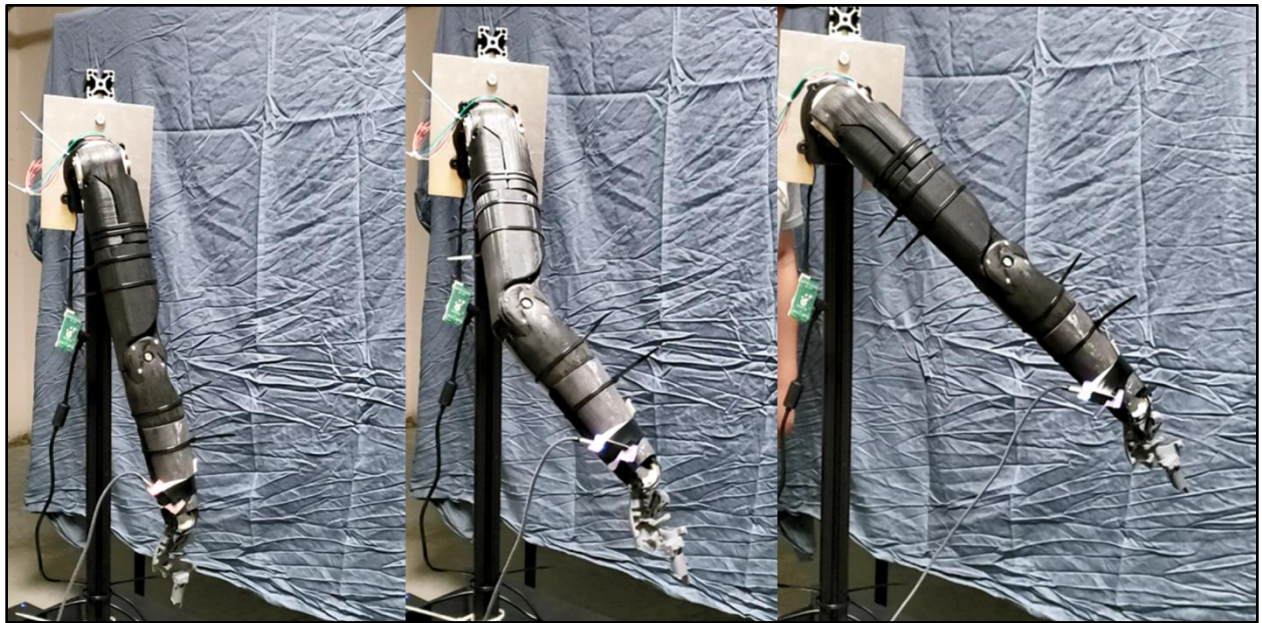


Figure 6.23: High Power Test In Progress

High Powered Testing involved using a 12volt/10 amp power supply, and primarily utilized the Controller.cpp C++ code for testing. This testing involved direct joint control of the motors, and the verification of the Dynamixel.moveTogether function. Several issues were found and addressed as discussed below:

- Acceleration Profiles: The velocity profile for the motors were a rectangular curve. This had the effect of high accelerations at the beginning and end of the motion loop which translated to extremely jerky movements. This was solved by adding an acceleration profile for each motor making the velocity profile trapezoidal.
- Read Speed Limitation: During every loop, the controller read the positions of each motors for feedback. This had the effect of delaying the motors from entering the next loop. This read loop was removed from the code, and the arm motion moves much quicker between loops.

#### 6.4.3 Proof of Concept Realization

To integrate the arm hardware, arm controller, and the Magic Leap, a series of steps must occur in order. The initial step was to set up the Magic Leap environment directly to the computer so the telemetry text file data can be accessed by the controller. Next, within Magic leap, the eye tracking program was opened, and the initial prism positioning was set to the physical anchor position of the arm. The user then proceeds to acquire the required trajectory positions using the Magic Leap. When the arm is ready to move, a switch is activated to actuate the motion of the arm. A visual of this is shown on (Figure 6.24).



Figure 6.24: Proof of Concept Visualization

## Chapter 7: Testing and Results

### 7.1 Methodology

To verify the efficacy of the Humanoid Arm Robotic Unit Control System, simulations and experimental tests were developed and conducted.

For the simulations, MATLAB was the primary computational program used. A 3D image of the Arm unit was developed in VRML for visualization along a wireframe model developed in MATLAB to track the positioning of the arm within a set coordinate system. Details on the simulation will be discussed in the following sections.

Once the simulations were completed, experimental tests were done on the Arm hardware to observe how well the controller works with an actual unit. Since the Arm hardware is susceptible to wear and tear, stable trajectories were chosen to minimize damage or maintenance during the testing phase. Results from the experimental testing are then compared with the simulation values.

### 7.2 Simulation

#### 7.2.1 Comparison Between VRML and MATLAB Simulation

Comparing the two simulation platforms, we see that both showcase the same positioning of the arm as dictated by the resolved-rate algorithm. (Figure 7.1) & (Figure 7.2) show trajectories that are known to be stable. One major difference, however, is that the VRML Simulation (left image) has a bit of perspective view versus the almost isometric view for the

Wireframe Simulation (right image). Note also that the right image has a bolded trajectory line (in red and green) representing the trajectory of motion, while the left image does not.

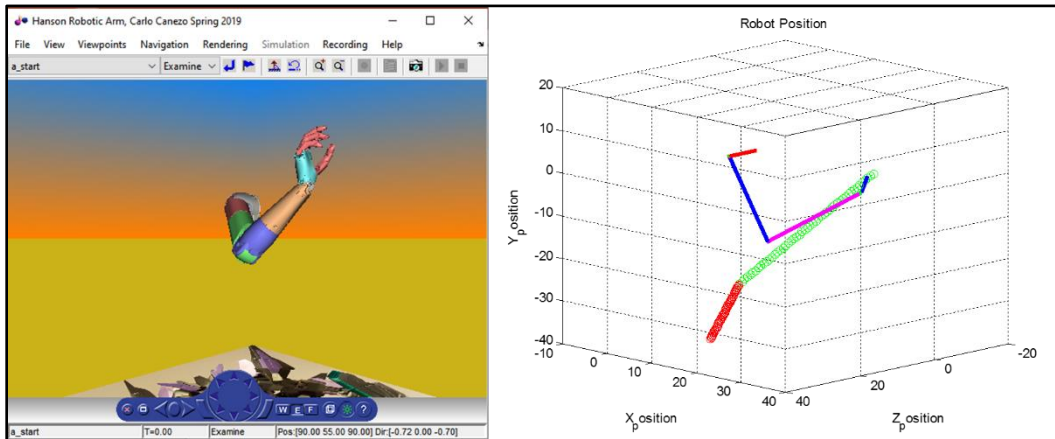


Figure 7.1: Comparison of VRML (Left) and Wireframe (Right) Example 1

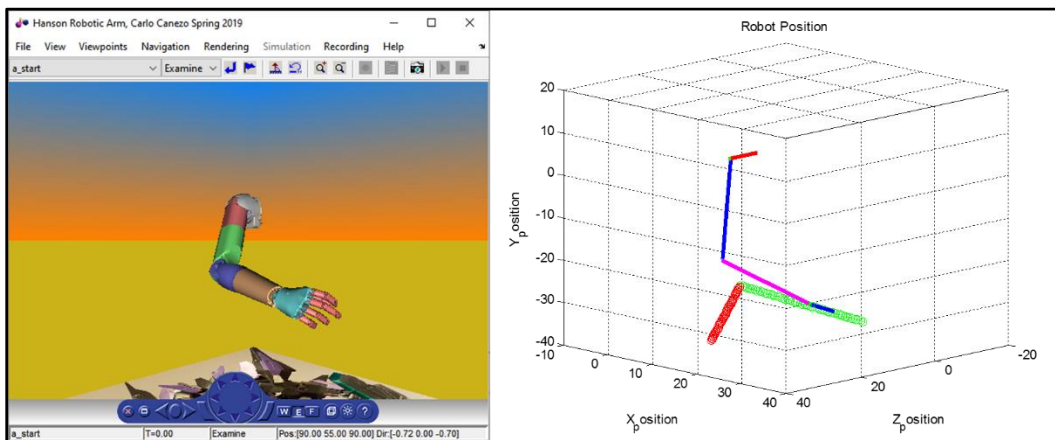


Figure 7.2: Comparison of VRML (Left) and Wireframe (Right) Example 2

### 7.2.2 Simulation Results of the Control System

To showcase the characteristics of the control system, we will compare the different Optimization Configurations between two trajectories: The first will showcase a “stable” trajectory where the end effector is well within the limits of its workspace, and that there are no significant singularity zones in its travel. The second will showcase an “unstable” trajectory effectively forcing the end effector through workspace limitations, well past joint limits, and

singularity zones. Each trajectory will start at a ready position where the arm is partially extended due to placing all the joints within the middle of their limits. This is done so the arm will not start at a potential singularity position. The following configurations are compared: A) With SR Inverse and With Joint Limit Avoidance (Weighted SR Inverse Optimization Solution), B) Without SR Inverse and With Joint Limit Avoidance (Weighted Least Norm Optimization Solution), and C) With SR Inverse and Without Joint Limit Avoidance (SR Inverse Optimization Solution). The following constants are used throughout the simulation test as shown in equation (53).

$$\begin{aligned}
 w_0 &= 100000 \\
 k_0 &= 100 \\
 \text{SpeedRev} &= 4 \text{ rpm} \\
 \text{SpeedLimit} &= 10 \text{ cm/s}
 \end{aligned} \tag{53}$$

### 7.2.3 Stable Trajectory Comparison

Configuration A as shown in (Figure 7.3) employs the SR Inverse and the Joint Limit Avoidance equations in calculating the motion of the arm (Weighted SR Inverse Optimization Solution). Joint limits representing the physical robotic arm are active in this mode. The characteristic of this configuration is that the robotic arm will do its best to complete the trajectory and maintain stability at the cost of end effector trajectory accuracy.

Configuration B as shown in (Figure 7.4) employs only Joint Limit Avoidance equations while striving to make the end effector as accurate to the trajectory as much as possible (Weighted Least Norm Optimization Solution).

Configuration C represents the best-case scenario of mobility for the arm as there are no limitations of the joint angles that can reduce its theoretical workspace while maintaining stability (SR Inverse Optimization Solution). As showcased in (Figure 7.5), the position angles, the

joint velocities, and the Manipulability curves are much more smoother than the other configurations. This will be the basis of comparison for the other configurations.

During the simulation for Configuration A and Configuration B, as the arm reaches its joint limits, the Joint Limit conditions from equation (8) are activated. This is evidenced by the sharp but very short gradients in the velocity graph indicating that a joint has reached a limit and is straddling that boundary, but its velocity is being reversed to bring back the joint within its allowable zones. In general, the overall velocity curve profile is still visible. An important note is that joint limits effectively reduce the overall workspace of the arm. This means that the calculated end effector trajectory can potentially cross a space where the end effector cannot physically go into. As the end effector traverses its trajectory, the end effector eventually ends up in such a zone. For Configuration A & Configuration B, this is usually characterized by rapid and large gradients in the velocity curves, and the curve profile is lost. It is in this zones that Configuration A & Configuration B start to differ significantly. First off, for Configuration A, the joint limits are observed throughout its motion as evidence of the flatlining of the position angle curves for some joints. Furthermore, as the calculated Manipulability values drop below the “ $w_0$ ” threshold, the second term in the singularity robust inverse calculations shown in equation (11) starts to activate. The “k” factor of equation (11) starts to augment equation (13) deviating the end effector from its calculated trajectory in order to keep the end effector out of the singularity zone, but not letting it stray too far off course. This augmentation also has the effect of keeping the Manipulability of the system relatively high, since we never go into singularity, especially in cases where the trajectory is clearly out of the workspace of the end effector. This is showcased in (Figure 7.6) where the manipulability for Configuration A is considerably much higher than

Configuration C. Configuration A will calculate and complete its motion, though at the expense of accuracy. This concept might be detrimental for standard robotic manipulators, but for our application of a wearable prosthesis, the loss of accuracy can be argued to be negligible since the user's body is not anchored and can move to compensate. In contrast, Configuration B has no way of solving for the singularity. The arm motion is already limited by the joint angle limits, hence the overall workspace for the arm is also reduced. In the instance the end effector reaches a singularity zone, the arm struggles to find a solution, and in the process exhibits jittery motion and, in some cases, forces a joint angle position that is not within its limits. The arm is not able to fully complete its trajectory either. In comparing the Manipulability graphs between Configuration B and Configuration C in (Figure 7.6), initially, the motion of the arm was in the stable regime. As it reached a singularity zone around 4 second time frame, the Manipulability values started to return a null value. Further investigation indicated from the Joint Limit Avoidance weight matrix "W" showed that two joints are weighted to infinity meaning they cannot move and when applied to equation (10) returns that null value. At this point, the arm was effectively stuck in place and couldn't continue its motion.



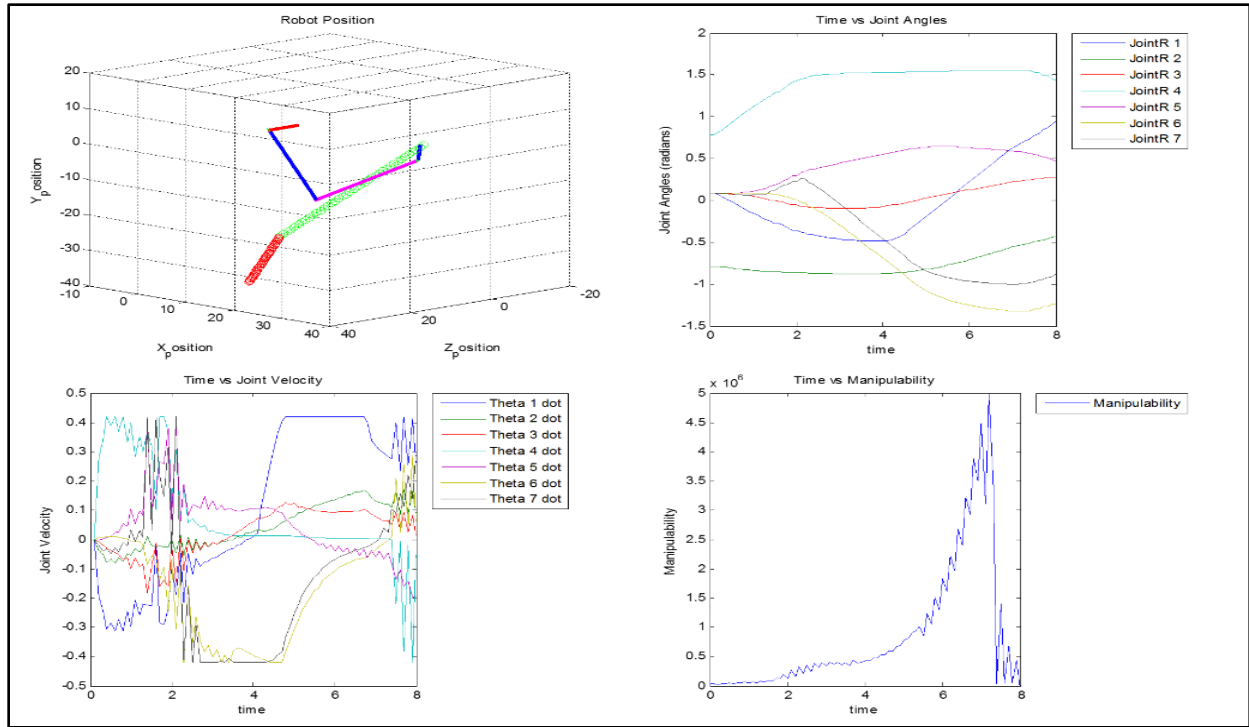


Figure 7.3: Joint Angles, Joint Velocities, and Manipulability for Stable Trajectory for Arm Configuration A

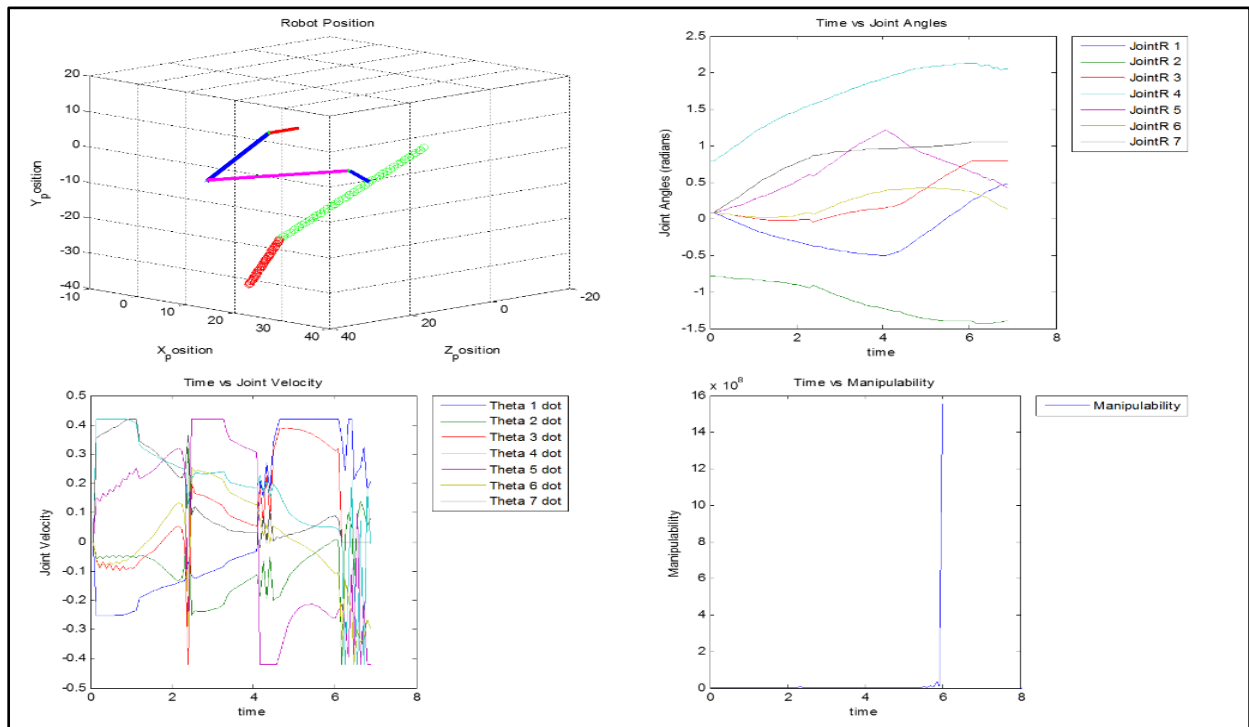


Figure 7.4: Joint Angles, Joint Velocities, and Manipulability for Stable Trajectory for Arm Configuration B

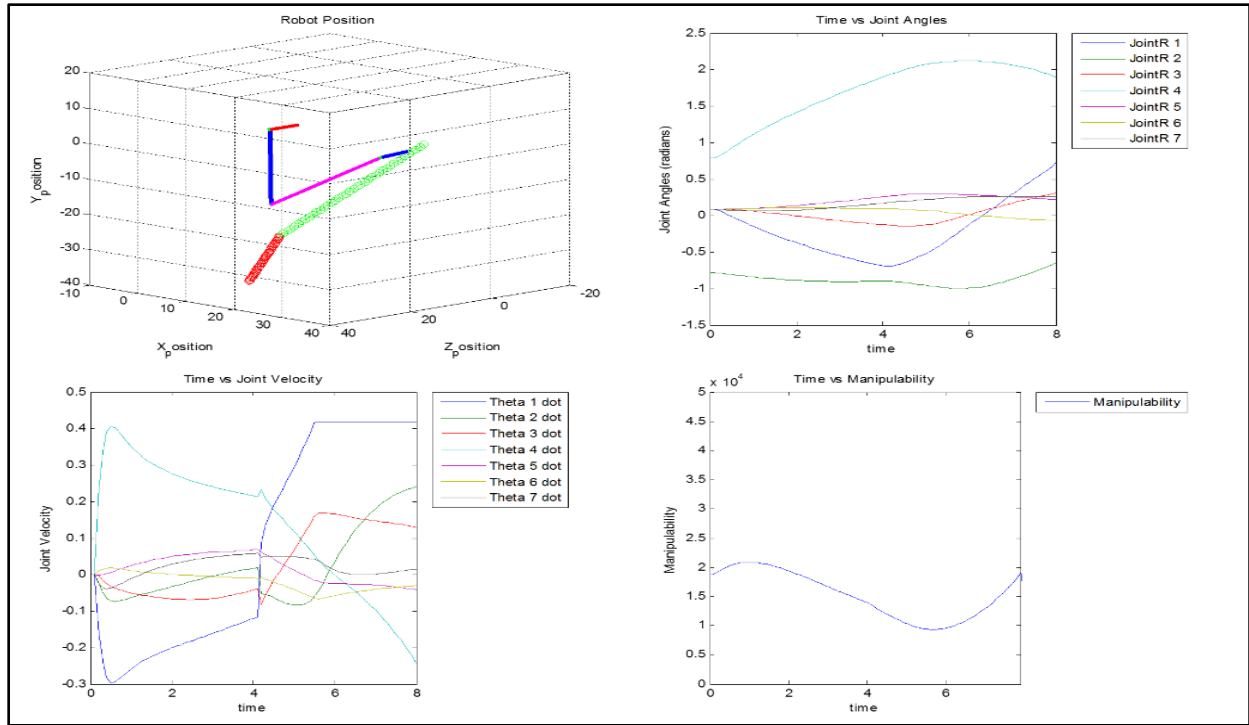


Figure 7.5: Joint Angles, Joint Velocities, and Manipulability for Stable Trajectory for Arm Configuration C

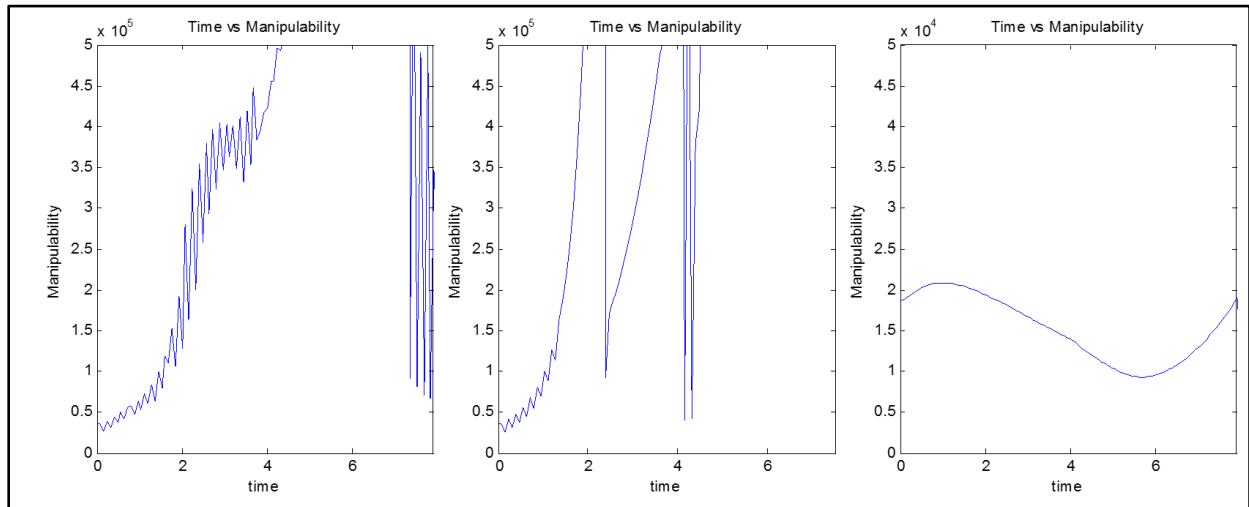


Figure 7.6: Stable Manipulability Graphs Scaled for Comparison Configuration A (Left), Configuration B (Middle), and Configuration C (Right)

### 7.2.4 Unstable Trajectory Comparison

The unstable trajectory regime is where Configuration A shown in (Figure 7.7) shows a distinct advantage over Configuration B shown in (Figure 7.8). Configuration A was able to

complete its trajectory, while Configuration B suffered a severe system instability that coincides with its Manipulability values unable to be calculated and the erratic gradients in the velocities frequently around the 4 second mark as shown in (Figure 7.8). Configuration A still also exhibits the same characteristics with its velocity gradients as its equivalent in Stable Trajectory, and again it is the only configuration to hold the joint limits intact. As the arm moves through its trajectory, the end effector encounters the singularity zones, but the SR inverse equations become active ensuring that the end effector is never in a singularity. This has the added effect of keeping Manipulability high as shown in (Figure 7.10).

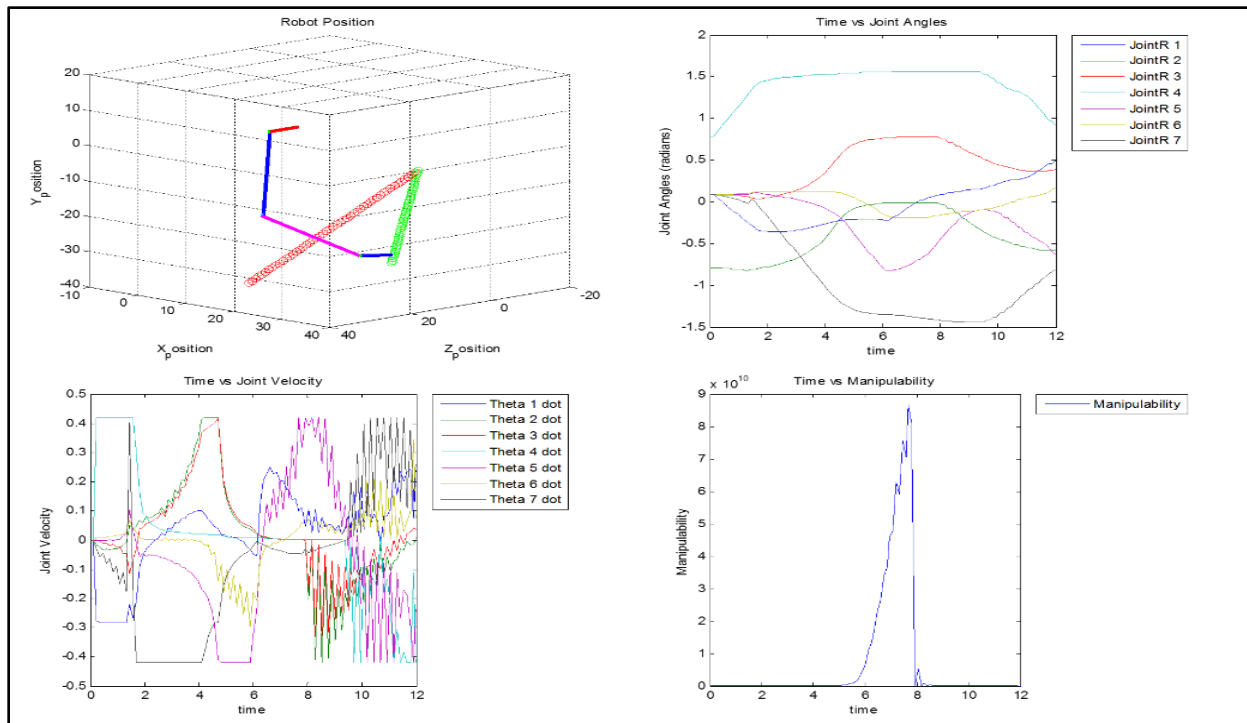


Figure 7.7: Joint Angles, Joint Velocities, and Manipulability for Stable Trajectory for Arm Configuration A

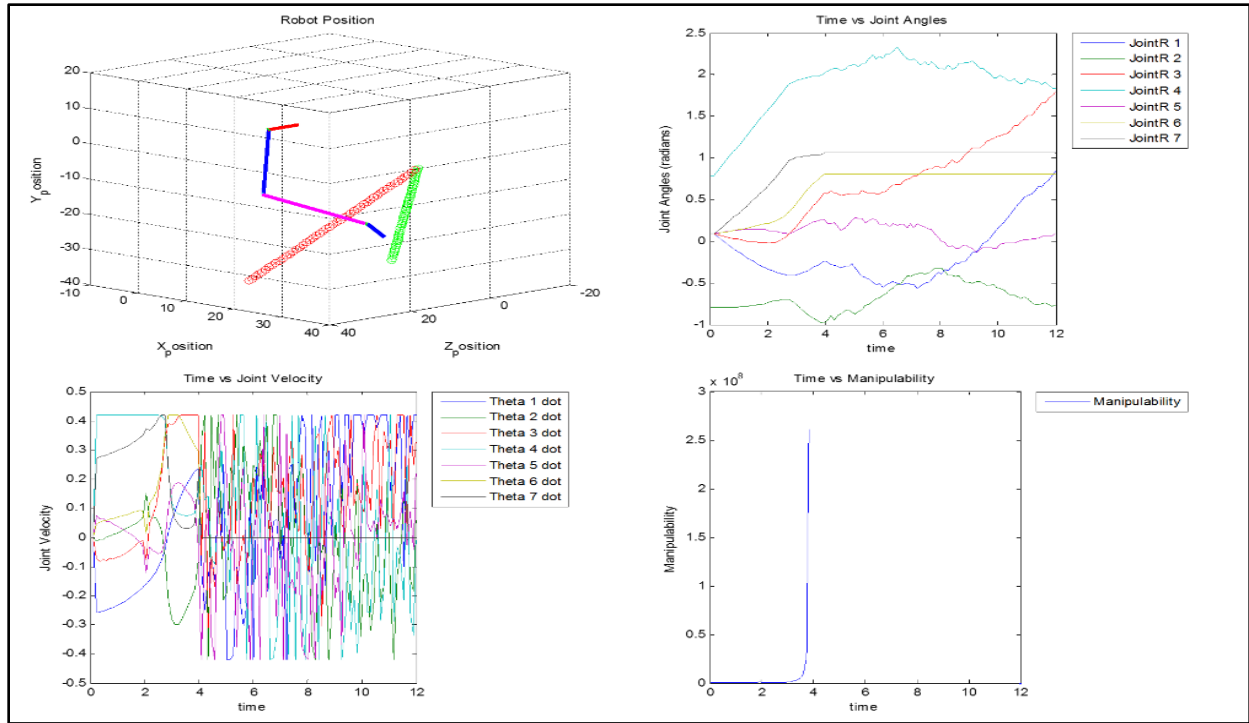


Figure 7.8: Joint Angles, Joint Velocities, and Manipulability for Stable Trajectory for Arm Configuration B

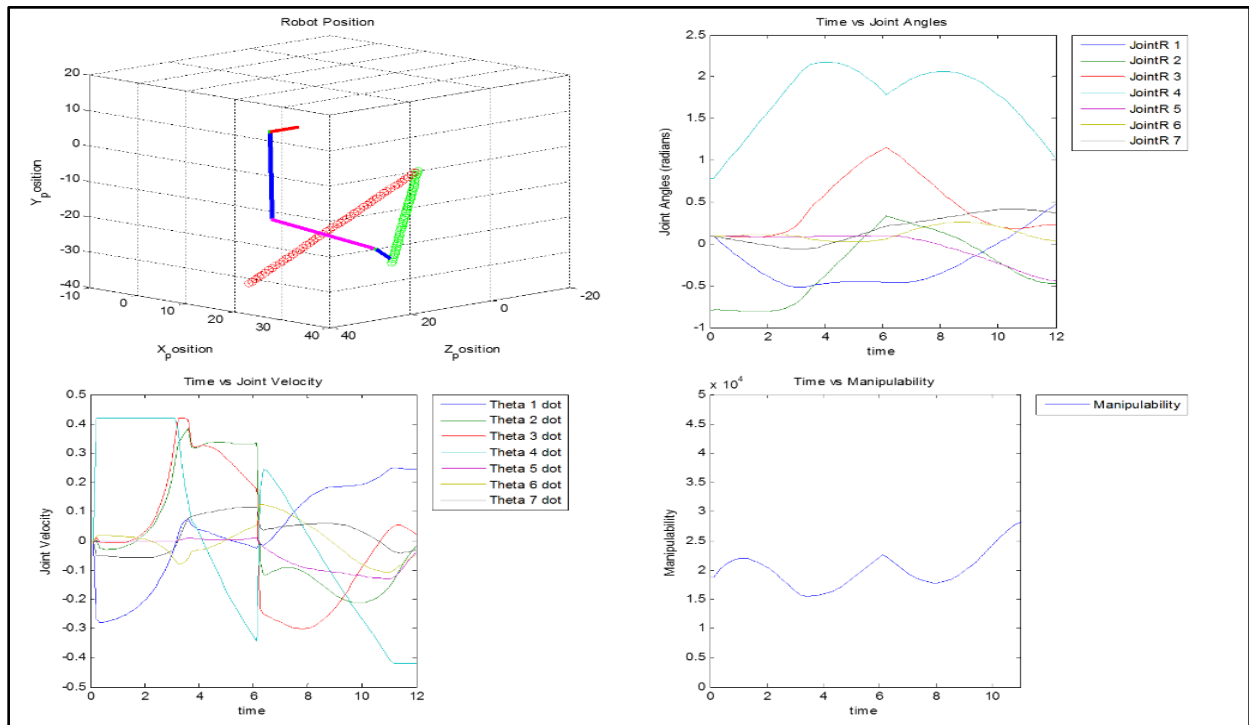


Figure 7.9: Joint Angles, Joint Velocities, and Manipulability for Stable Trajectory for Arm Configuration C

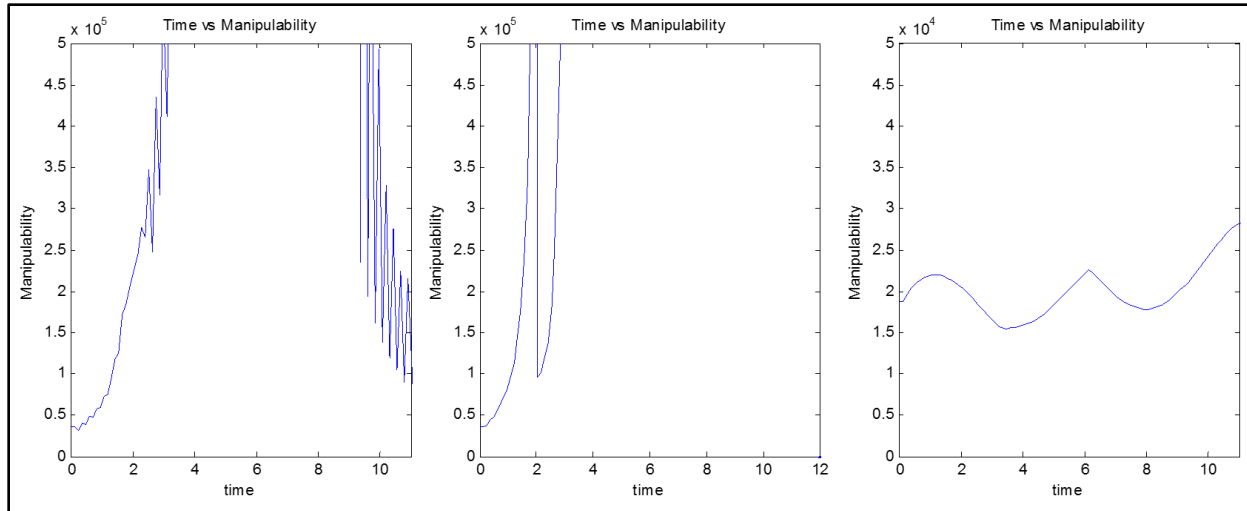


Figure 7.10: Unstable Manipulability Graphs Scaled for Comparison Configuration A (Left), Configuration B (Middle), and Configuration C (Right)

### 7.3 Experimental Testing

Using a predetermined stable trajectory, the robotic arm will operate 3 times from start to finish. The output angles from the motors will be recorded and graphed. The output graphs will be compared between each run and see the difference. This is to verify how repeatable the system motion is. The output graphs will then be compared to the simulation graphs. This is to verify how accurate the motion for the system is. Using the same predetermined trajectory, the calculated Manipulability from the simulations will be compared to the calculated Manipulability from the controller.

Three runs were conducted in succession, and the trajectory chosen for the runs was a known stable trajectory. Each run consists of a joint controlled motion from rest position to a ready position. The ready position is a predetermined set of joint angles that was chosen to minimize starting the calculations from a singularity or a joint-locked point. From the ready position, the arm will move to complete its calculated trajectory. Once it reaches its final

trajectory point, the arm resets back to the ready position and finally goes back to its rest position.

### 7.3.1 Experimental Graphed Results

Results from the testing with the HARU Control system are shown below. Three runs were performed, and each run is compared to the simulated values from MATLAB. Motor feedback is from the information read at a point during the trajectory sampled at 60Hz.

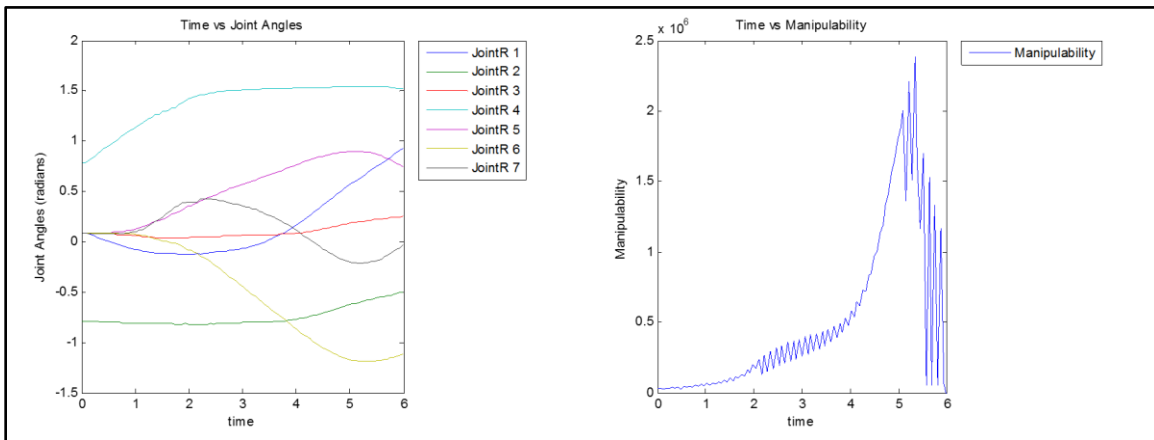


Figure 7.11: Simulation Results for Joint Position and Manipulability

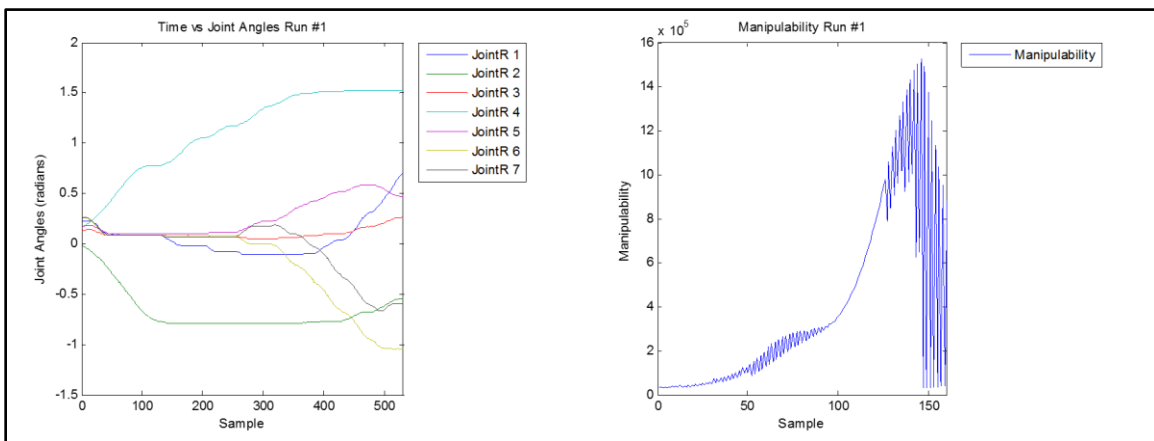


Figure 7.12: Motor Feedback Results for Joint Position & Manipulability for Run 1

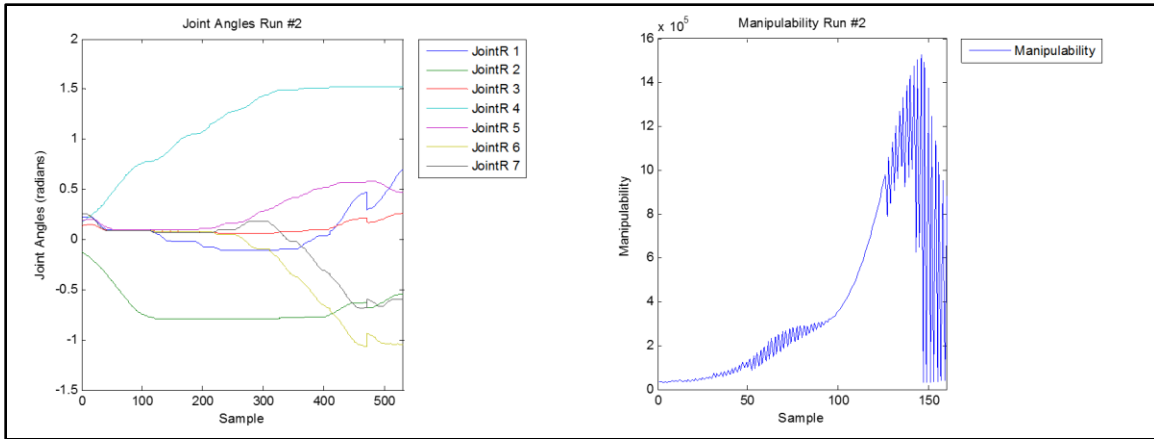


Figure 7.13: Motor Feedback Results for Joint Position & Manipulability for Run 2

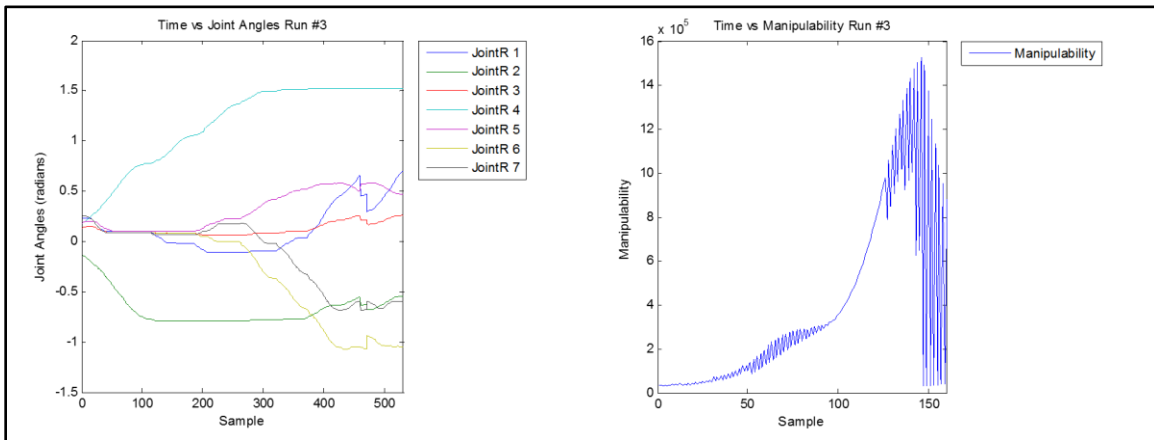


Figure 7.14: Motor Feedback Results for Joint Position & Manipulability for Run 3

### 7.3.2 Discussion

As a reference, the simulation results shown in (Figure 7.11) will act as the reference baseline for comparison between the experimental runs. The key points of comparison between the simulation and the experimental results are the following: the effect of the Joint Limits, the smoothness of the Joint Angle Profiles, and the effects of SR Inverse optimization.

For all the runs, motor Joint Angle feedback started at the rest position of the arm. Samples 0-100 of the Joint Angle feedback showcased the motion to the ready position. From Samples 100 onward showcased the trajectory motion.

First thing to highlight is that all the runs showed that the Joint Limit Avoidance performed as expected. While some joints did start to reach its limits, they never exceeded as evidenced by the flattening of the Joint Angle Curves. Furthermore, it is also noticed that once the motion was never in any sort of instability as evidenced by the high Manipulability values.

Comparing the Simulated Positions versus the runs, the first thing to note is the smoothness of the curves. Run 1 Position values as shown in (Figure 7.12) closely matches the smoothness of the simulation curves compared to the other runs, but the overall position profiles are very rough. This is primarily attributed to the velocity profile selected for the motors. The motors are limit locked at a velocity as in the simulation, but the simulation doesn't account for acceleration as best evidenced in the velocity profile from (Figure 7.3). Such rapid changes in velocity is very damaging to the arm, so a physical safety limit has been added to the motors to be locked at a certain acceleration that was found safe during High Power Testing phase.

It was also noticed that on the latter end of the trajectory, there were some high positional jumps occurring in certain joints. This was more prevalent in Run 2 & Run 3 as shown in (Figure 7.13) and (Figure 7.14) respectively. The arm does recover, however, accuracy from the original trajectory is affected. This can be attributed to several causes:

1. Spring motion due to cantilevered/dynamic effect from moving the arm to an extended position due to the weight of the arm. This tends to occur in trajectories where the arm will have to extend its reach.
2. Motors slowing down at the end of a certain angle position prior to getting a new angle to move to, then speeding up again. This helps contribute to the dynamic effects as discussed in the first bullet.



In comparing the Manipulability values between the simulation and controller, it was found that the simulation values reported larger than the controllers. Since the form of the curve for the simulation matches the controller's, the difference is attributed to rounding errors in the calculation. Comparing the Manipulability between runs showed consistency in the calculation with no appreciable difference between them.

## Chapter 8: Conclusions and Future Work

### 8.1 Conclusions

Presented in this thesis is an integrated control system for a powered prosthetic arm using an Augmented Reality Device. The control algorithm implemented is based off a Kinematic resolved-rate control structure utilizing an optimized Jacobian that minimizes singularities and maximizes manipulability. The hardware utilized to test the control theory is based off the Hanson Power Prosthetic Arm design for the Sophia project, which we built a replica of to test our control system. The subsequent AR device being used for the sensory telemetry that will interact the user and the environment is the Magic Leap Goggles.

Virtual simulations and experimental testing show that this control system is relatively robust to handle various linear trajectories, and that it is further optimized for better joint limit and singularity avoidance.

### 8.2 Future Work – AI Development

Part of the effort to further seamlessly integrate the robotic arm with the user's sense of perception and reduce cognitive load, the intention for the future of this project is to create a robust artificial intelligence to help complete the following objectives: (a) recognition of the environment/objects through detection and classification; (b) pose estimation of objects by simplifying volumetric shapes found via voxelization of superquadrics derived from 3D point cloud data; (c) recognition of the user's intentions to drive the actions of the robotic arm.

Combining all of these with the existing platform already developed will form the basis of the Visual System Code as shown in (Figure 8.1).

Another aspect of this project is to use a robust object detection algorithm, which will be integrated into the wearable headset, to determine what kind of object the user is looking at and determine the pose (position and orientation) of the object. The latter is important since this will also be fed into the robotic arm controller to help create the required trajectory and grasping configuration for the arm and hand.

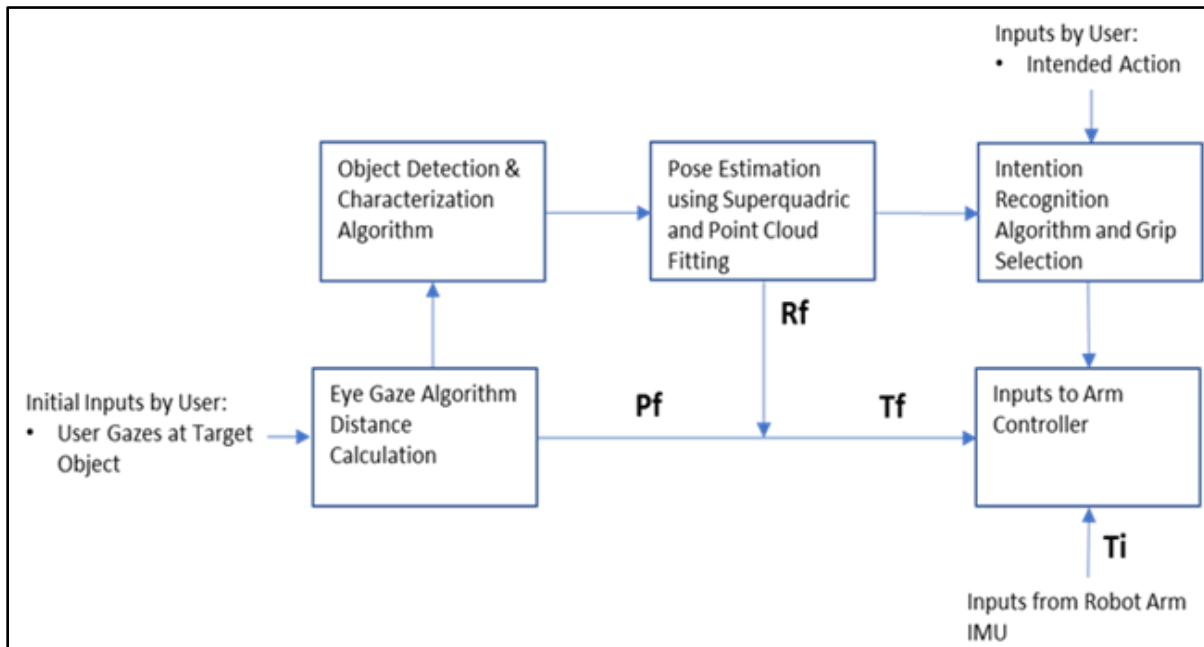


Figure 8.1: High Level Visual System Code FlowChart

### 8.2.1 Object Recognition

The expected object recognition algorithms will employ a degree of discrimination and generalization to provide information to the controller for determining grasping position. For example, a pen and a spoon could be generalized together for their shape, but the system needs

to know that one is a pen, and the other is a spoon leading to different grasping strategies depending on use of the object.

### 8.2.2 Pose Estimation

To determine hand orientation and potentially grip patterns, it is necessary to determine the general orientation or pose of the targeted object. Objects can vary in sizes and shapes; however, most household objects can be generalized into basic volumetric shapes called superquadrics. These basic shapes are easily renderable, and volumetric parameters such as moment of inertia, general volume size, and even object pose, can be readily extracted.

In conjunction with using 3D Point Cloud space data of the object gathered from the 3D Camera, which in this case will be from the Magic Leap goggles, the plan is to fit the point cloud into a voxel, which is a 3D representation of a Pixel, normally a cube, and slowly decrease the voxel size to a set error threshold that will best fit a superquadric. From here, the relevant pose information can be determined to be able to provide input on the best possible wrist orientation to use for the detected object. This information will be used for grasping the intended object using one of the grasp patterns supplied by the prosthesis control algorithm.

### 8.2.3 Intention Recognition

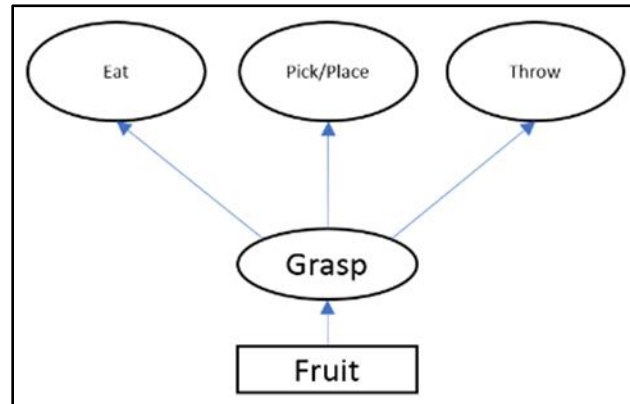


Figure 8.2: State/Action Network

The plan of intention recognition will utilize a network model represented as a graph, with the object nodes and action/goal nodes are set with interdependence. For simplicity, the action/goal states will be limited to a few action nodes just to prove its feasibility. (Figure 8.2) shows a simple example of a network using class object fruit. Note that some classes of objects might share the same action/goal node depending on the similarity. Creating this intention recognition model should help the user feel more in tune with the device. As more data is generated (cues from the object, user bias, voice recognized commands, etc.) to specifically aid with the intention recognition model, we will introduce machine learning to help analyze the minute differences in data that makes each user unique. If implementation is feasible, the information we get from this learning concept will help with minute fine tuning the operation of the powered prosthesis.

## References

- [1] R. Alqasemi, and R. Dubey, "Maximizing Manipulation Capabilities for People with Disabilities Using a 9-DoF Wheelchair-Mounted Robotic Arm System," 2007 IEEE 10th International Conference on Rehabilitation Robotics, Noordwijk, Netherlands, 2007, pp. 212-221.
- [2] T.F. Chan, and R.V. Dubey, "A Weighted Least-Norm Solution Based Scheme for Avoiding Joint Limits for Redundant Manipulators," IEEE Transactions on Robotics and Automation, Vol. 11, No. 2, pp. 286-292, April 1995.
- [3] J.J. Craig, Introduction to Robotics Mechanics and Control Third Edition. Upper Saddle River, NJ: Pearson Prentice Hall, 2005.
- [4] K. Fujii, A. Salerno, K. Sriskandarajah, K. Kwok, K. Shetty, and G. Yang, "Gaze contingent cartesian control of a robotic arm for laparoscopic surgery," 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems, Tokyo, 2013, pp. 3582-3589.
- [5] G. Ghazaei, A. Alameer, P. Degenaar, G. Morgan, and K. Nazarpour, "Deep learning-based artificial vision for grasp classification in myoelectric hands," Journal of Neural Engineering, Vol. 14, No. 3, May, 2017. [Online Serial]. Available: <http://iopscience.iop.org/article/10.1088/1741-2552/aa6802/meta>. [Accessed September 26, 2018].
- [6] M.S. Johannes, J.D. Bigelow, J.M. Burck, S.D. Harshbarger, M.V. Kozlowski, and T.V. Doren, "An Overview of the Developmental Process for the Modular Prosthetic Limb," John Hopkins Technical Digest, Vol. 30, No. 3, pp. 207-216, 2011.
- [7] M.J. Kang, C.H. Lee, J.H. Kim, and U.Y. Huh, "Distance and velocity measurement of moving object using stereo vision system," 2008 International Conference on Control, Automation and Systems, Seoul, 2008, pp. 2181-2184.
- [8] J. Kofman, X. Wu, T.J. Luu, and S. Verma, "Teleoperation of a robot manipulator using a vision-based human-robot interface," in IEEE Transactions on Industrial Electronics, vol. 52, no. 5, pp. 1206-1219, Oct. 2005.

- [9] A. Leeper, K. Hsiao, E. Chu, and J.K. Salisbury, "Using Near-Field Stereo Vision for Robotic Grasping in Cluttered Environments," In: Khatib O., Kumar V., Sukhatme G. (eds) Experimental Robotics. Springer Tracts in Advanced Robotics, vol 79, Berlin, Heidelberg: Springer, 2014.
- [10] L. Perez, I. Rodriguez, N. Rodriguez, R. Usamentiaga, and D.F. Garcia, "Robot Guidance Using Machine Vision Techniques in Industrial Environments: A Comparative Review," MDPI Sensors, Vol. 16, No. 335, 2016.
- [11] S.L. Phillips, L. Resnik, C. Fantini, and G. Latlief, "Endpoint Control for a Powered Shoulder Prosthesis," Journal of Prosthetics and Orthotics, Vol. 25, No. 4, pp. 193-200, 2013.
- [12] C.C. Postelnicu, D. Talaba, and M.I. Toma, "Controlling a Robotic Arm by Brainwaves and Eye Movement," IFIP International Federation for Information Processing AICT 349, 2011, pp. 157-164.
- [13] Ramisa, G. Alenyà, F. Moreno-Noguer, and C. Torras, "Using depth and appearance features for informed robot grasping of highly wrinkled clothes," 2012 IEEE International Conference on Robotics and Automation, Saint Paul, MN, 2012, pp. 1703-1708.
- [14] L.J. Resnik, C. Fantini, G. Latlief, S. Phillips, N. Sasson, and E. Sepulveda, "Use of the DEKA Arm for amputees with brachial plexus injury: A case series," PLoS ONE, Vol. 12, No. 6, June, 2017. [Online Serial]. Available: <https://doi.org/10.1371/journal.pone.0178642>. [Accessed September 18, 2018].
- [15] L.J. Resnik, F. Acluche, and S.L. Klinger, "User experience of controlling the DEKA Arm with EMG pattern recognition," PLoS ONE, Vol. 13, No. 9, September, 2018. [Online Serial]. Available: <https://doi.org/10.1371/journal.pone.0203987>. [Accessed September 28, 2018].
- [16] L.J. Resnik, M.L Borgia, and F. Acluche, "Perceptions of satisfaction, usability and desirability of the DEKA Arm before and after a trial of home use," PLoS ONE, Vol. 12, No. 6, June, 2017. [Online Serial]. Available: <https://doi.org/10.1371/journal.pone.0178640>. [Accessed September 25, 2018].
- [17] L.J. Resnik, M.L Borgia, F. Acluche, J.M. Cancio, G. Latlief, and N. Sasson, "How Do the Outcomes of the DEKA Arm Compare to Conventional Prostheses?," PLoS ONE, Vol. 13, No. 1, January, 2018. [Online Serial]. Available: <https://doi.org/10.1371/journal.pone.0191326>. [Accessed September 25, 2018].

- [18] L.J. Resnik, S.L. Klinger, and K. Etter, "The DEKA Arm: Its features, functionality, and evolution during the Veterans Affairs Study to optimize the DEKA Arm," *Prosthetics and Orthotics International*, Vol. 38, No. 6, pp. 492-504, 2013.
- [19] Z. Wang, T. Chen, and C. Yu, "Based on Binocular Stereo Vision of Moving Target Detection," *Proceedings of the 2009 International Symposium on Information Processing*, Huangshan, China, 2009, pp. 189-192.
- [20] W. Yu, B.E. Fritz, N. Pernalet, M. Jurczyk, and R.V. Dubey, "Sensors assisted telemanipulation for maximizing manipulation capabilities of persons with disabilities," *11th Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems, 2003. HAPTICS 2003. Proceedings.*, Los Angeles, CA, USA, 2003, pp. 295-301.
- [21] Ottobock, "DynamicArm," 2019. [Online]. Available: <https://shop.ottobock.us/Prosthetics/Upper-Limb-Prosthetics/Myoelectric-Elbows/DynamicArm-Elbow/DynamicArm/p/12K100N~550>. [Accessed May 28, 2020].
- [22] Ottobock, "MyoRotronic," 2019. [Online]. Available: <https://shop.ottobock.us/Prosthetics/Upper-Limb-Prosthetics/Myo-Hands-and-Components/Myo-Wrist-Units-and-Rotation/MyoRotronic/p/13E205>. [Accessed May 28, 2020].
- [23] Ottobock, "Electric Wrist Rotator," 2019. [Online]. Available: <https://shop.ottobock.us/Prosthetics/Upper-Limb-Prosthetics/Myo-Hands-and-Components/Myo-Wrist-Units-and-Rotation/Electric-Wrist-Rotator/p/10S17>. [Accessed May 28, 2020].
- [24] Ottobock, "Electric Wrist Rotator," 2020. [Online]. Available: <https://www.ottobockus.com/prosthetics/upper-limb-prosthetics/solution-overview/bebionic-hand/>. [Accessed May 28, 2020].
- [25] Ottobock, "Michaelangelo Prosthetic Hand," 2020. [Online]. Available: <https://www.ottobockus.com/prosthetics/upper-limb-prosthetics/solution-overview/michelangelo-prosthetic-hand/>. [Accessed May 28, 2020].
- [26] Fillauer, "Utah Arm 3+ (U3+)," 2016. [Online]. Available: <https://www.utaharm.com/Prosthetist-Directory/wpdm-package/utah-arm/>. [Accessed May 28, 2020].
- [27] Ossur, "I Limb Access," 2020. [Online]. Available: <https://www.ossur.com/en-gb/prosthetics/arms/i-limb-access>. [Accessed May 28, 2020].



- [28] Fillauer, "Multi-Flex Wrist Option for Standard ETD," [Online]. Available: <https://fillauer.com/products/multi-flex-wrist-option-for-standard-etd/>. [Accessed May 28, 2020].
- [29] Liberating Technologies Inc, "Boston Digital Arm Plus System," 2012. [Online]. Available: [https://www.college-park.com/media/wysiwyg/pdf/liti/Boston\\_Digital\\_Arm-Plus\\_System\\_clinical\\_manual\\_rev\\_2-12.pdf](https://www.college-park.com/media/wysiwyg/pdf/liti/Boston_Digital_Arm-Plus_System_clinical_manual_rev_2-12.pdf). [Accessed May 28, 2020].
- [30] Fillauer, "Taska," 2018. [Online]. Available: <https://www.utaharm.com/Prosthetist-Directory/wpdm-package/taska-hand/>. [Accessed May 28, 2020].
- [31] Y. Nakamura, Advanced Robotics Redundancy and Optimization. Reading, MA: Addison-Wesley Publishing Company, 1991.
- [32] J.Y. Luh, M.W. Walker, and R.P.C. Paul, "Resolved-acceleration control of mechanical manipulators," IEEE Transactions on Automatic Control, Vol. 25, No. 3, pp. 468-474, June 1980.
- [33] G. Morales, "Anthropomorphic Robot Hand and Arm CAD Model," 2018. [Online]. Available: <https://github.com/hansonrobotics/Anthropomorphic-Robot-Hand>. [Accessed February 1, 2019].
- [34] Magic Leap, "Magic Leap One Creator Edition," 2019. [Online]. Available: <https://www.magicleap.com/magic-leap-one>. [Accessed March 1, 2019].
- [35] R.P. Paul, Robot Manipulators: Mathematics, Programming, and Control. Cambridge, MA: MIT Press, 1981.
- [36] iFixit, "Magic Leap One Teardown," 2018. [Online]. Available: <https://www.ifixit.com/Teardown/Magic+Leap+One+Teardown/112245>. [Accessed March 10, 2020].
- [37] Robotis, "U2D2," 2020. [Online]. Available: <https://emanual.robotis.com/docs/en/parts/interface/u2d2/>. [Accessed March 10, 2020].

## Appendix A: Copyright Permissions

Permission below is for to use Figure 1.1

10/12/2020

RightsLink Printable License

WOLTERS KLUWER HEALTH, INC. LICENSE  
TERMS AND CONDITIONS

Oct 12, 2020

---

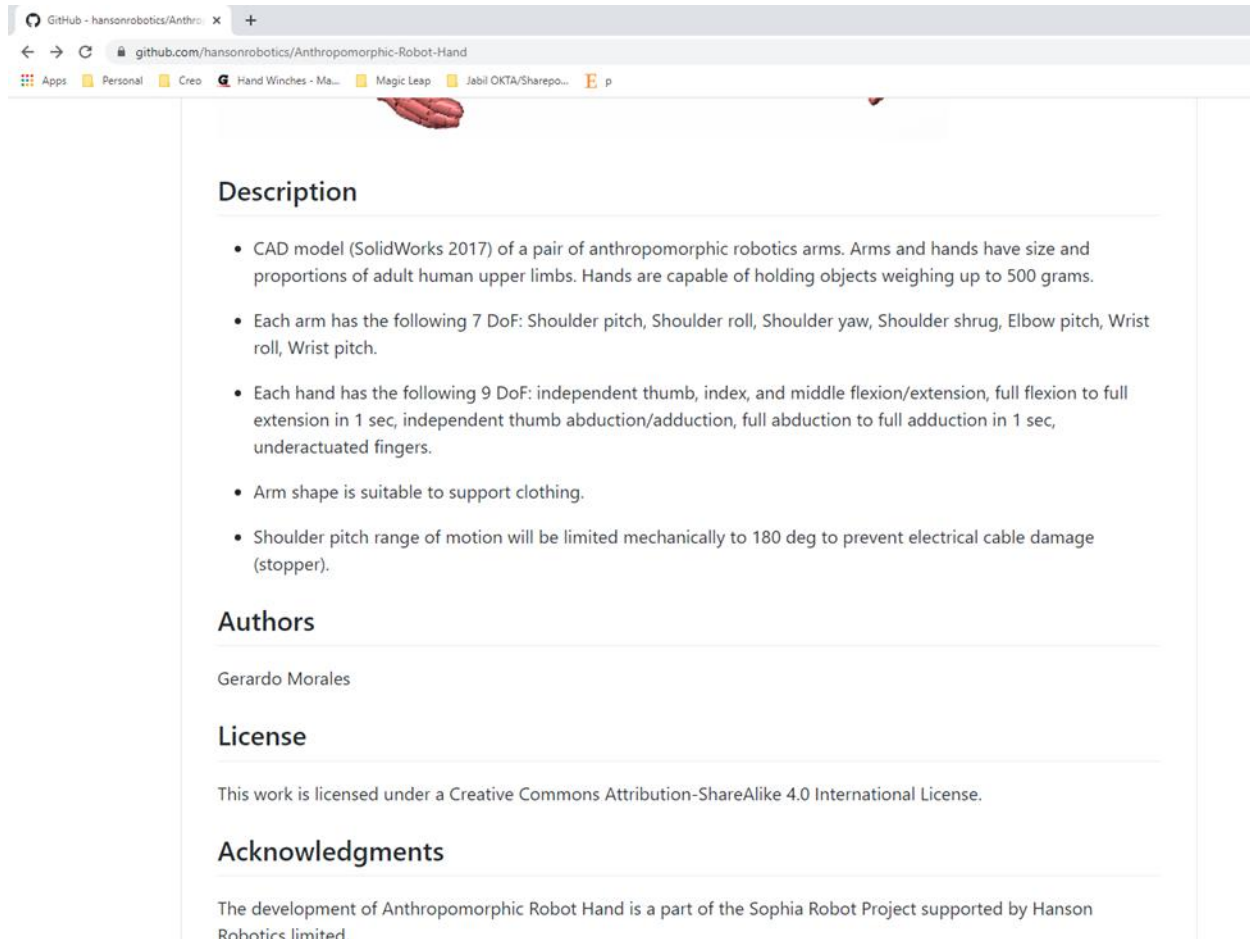
This Agreement between Mr. Carlo Canezo ("You") and Wolters Kluwer Health, Inc. ("Wolters Kluwer Health, Inc.") consists of your license details and the terms and conditions provided by Wolters Kluwer Health, Inc. and Copyright Clearance Center.

License Number	4926760395952
License date	Oct 12, 2020
Licensed Content Publisher	Wolters Kluwer Health, Inc.
Licensed Content Publication	Journal of Prosthetics & Orthotics
Licensed Content Title	Endpoint Control for a Powered Shoulder Prosthesis
Licensed Content Author	Sam Phillips, Linda Resnik, Christopher Fantini, et al
Licensed Content Date	Oct 1, 2013
Licensed Content Volume	25
Licensed Content Issue	4
Type of Use	Dissertation/Thesis
Requestor type	University/College
Sponsorship	No Sponsorship
Format	Electronic

<https://s100.copyright.com/CustomerAdmin/PLF.jsp?ref=4f317b86-feca-4751-ab99-11cb22cbe4dd>

1/5

Permission below is for to use Figure 4.1



The image shows a screenshot of a web browser displaying a GitHub repository page. The browser's address bar shows the URL 'github.com/hansonrobotics/Anthropomorphic-Robot-Hand'. The page content includes a 'Description' section with a bulleted list of specifications, an 'Authors' section listing 'Gerardo Morales', a 'License' section stating 'Creative Commons Attribution-ShareAlike 4.0 International License', and an 'Acknowledgments' section mentioning the 'Sophia Robot Project'.

## Description

- CAD model (SolidWorks 2017) of a pair of anthropomorphic robotics arms. Arms and hands have size and proportions of adult human upper limbs. Hands are capable of holding objects weighing up to 500 grams.
- Each arm has the following 7 DoF: Shoulder pitch, Shoulder roll, Shoulder yaw, Shoulder shrug, Elbow pitch, Wrist roll, Wrist pitch.
- Each hand has the following 9 DoF: independent thumb, index, and middle flexion/extension, full flexion to full extension in 1 sec, independent thumb abduction/adduction, full abduction to full adduction in 1 sec, underactuated fingers.
- Arm shape is suitable to support clothing.
- Shoulder pitch range of motion will be limited mechanically to 180 deg to prevent electrical cable damage (stopper).

## Authors

Gerardo Morales

## License

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

## Acknowledgments

The development of Anthropomorphic Robot Hand is a part of the Sophia Robot Project supported by Hanson Robotics limited.

Permission below is to use Figure 6.3, Figure 6.4, and Figure 6.5

The screenshot shows a web browser window with the URL [ifixit.com/info/Licensing](https://ifixit.com/info/Licensing). The page header includes the iFixit logo, navigation links for 'Fix Your Stuff', 'Right to Repair', and 'Store', and a search bar. The main content area is titled 'CONTENT LICENSING' and features a sidebar with a 'Licensing' section containing links for 'Content You Provide', 'Press', 'Everyone', and 'Commercial', and an 'FAQ' section with questions about Creative Commons, improving guides, embedding, and translation. The main content area has a heading 'Content Licensing' followed by a note that all content is licensed under the Creative Commons BY-NC-SA license. Below this is a section titled 'Licensing' stating that all iFixit content is licensed under the Creative Commons BY-NC-SA 3.0 license. Another section titled 'Content You Provide' lists three sources for submitted materials: original content, public domain, and compatible licensed content. A final paragraph states that users must not post content that infringes on third-party intellectual property rights and that users retain their own copyrights while granting iFixit nonexclusive rights to their submitted content.